TECHNISCHE UNIVERSITÄT DRESDEN

Dissertation

Parallelisierung von Algorithmen zur Nutzung auf Architekturen mit Teilwortparallelität

Rainer Schaffer

FAKULTÄT ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Technische Universität Dresden

Parallelisierung von Algorithmen zur Nutzung auf Architekturen mit Teilwortparallelität

Dipl.-Ing. Rainer Schaffer

der Fakultät Elektrotechnik und Informationstechnik
der Technischen Universität Dresden

zur Erlangung des akademischen Grades eines

Doktoringenieurs

(Dr.-Ing.)

genehmigte Dissertation

Vorsitzender:	Prof. Dr.-Ing. habil. W.-J. Fischer		
Gutachter:	Prof. Dr.-Ing. habil. R. Merker	Tag der Einreichung:	28.09.2009
	Prof. Dr. Ir. F. Catthoor	Tag der Verteidigung:	09.03.2010

Inhaltsverzeichnis

Zeichen, Bezeichnungen und Abkürzungen

Zeichen, Bezeichnungen

Bezeichnungen

e	Datenabhängigkeit
s	Zuweisung s
s_t	Teilzuweisung t der Zuweisung s
$f : \mathcal{X} \to \mathcal{Y}$	Abbildung f von $\mathcal{X}$ nach $\mathcal{Y}$
$y_s[.]$, $y_r[.]$, $y_a[.]$	Variablen eines Algorithmus

Skalare

m	Skaler (allgemein)

Vektoren

$\mathbf{m}$	Vektor (allgemein)
$\mathbf{0}$	Nullvektor $\mathbf{m}$ mit $\forall j : \ m_j = 0$
$\mathbf{1}$	Einsvektor $\mathbf{m}$ mit $\forall j : \ m_j = 1$
$\mathbf{i}$	Iterationsvektor
$\mathbf{i}^{\kappa}$	partitionierter Iterationsvektor
$\mathbf{d}$	Abhängigkeitsvektor (allgemein)
$\mathbf{d}(\mathbf{i}, \mathbf{i}')$	Abhängigkeitsvektor zwischen Iteration $\mathbf{i}$ und $\mathbf{i}'$
$\mathbf{d}(e)$	Abhängigkeitsvektor der Datenabhängigkeit e
$\mathbf{d}^{\kappa}$	partitionierter Abhängigkeitsvektor
$\mathbf{d}^{\mathbf{\Theta}}$	innerer Teilvektor
$\widehat{\mathbf{d}}^{\mathbf{\Theta}}$	äußerer Teilvektor
$\boldsymbol{\alpha}, \widehat{\boldsymbol{\alpha}} \in \mathbb{Z}^{1\times n}$	Abfolgevektor
$\boldsymbol{\iota}, \widehat{\boldsymbol{\iota}} \in \mathbb{Z}^{1\times n}$	Leerlaufverzögerung
$\boldsymbol{\tau}, \widehat{\boldsymbol{\tau}} \in \mathbb{Z}^{1\times n}$	Ablaufvektor

Matrizen

$\mathbf{M}$	Matrix (allgemein)
$\mathbf{E}$	Einheitsmatrix
$\mathbf{0}$	Nullmatrix
$\mathbf{\Gamma}$	Allokationsmatrix
$\mathbf{\Gamma}^M$	Datenausrichtungsmatrix
$\mathbf{\Theta}$	Partitionsmatrix (allgemein)
$\mathbf{\Theta}^P$	Partitionsmatrix (lokal parallel)
$\mathbf{\Theta}^S$	Partitionsmatrix (lokal sequenziell)

Mengen

$\mathcal{M}$	Menge (allgemein)
$\mathcal{E}$	Menge der Datenabhängigkeiten e
$\mathcal{D}^\kappa(e)$	Menge der partitionierten Abhängigkeitsvektoren $\mathbf{d}^\kappa$ resultierend aus der Datenabhängigkeit e
$\mathcal{H}$	konvexes Polyeder
$\mathcal{H}^\square$	Hyperquader
$\mathcal{I}$	Iterationsraum
$\mathcal{I}^\square$	Hyperquader, der den Iterationsraum $\mathcal{I}$ umschließt
$\mathcal{I}^\kappa$	partitionierter Iterationsraum
$\mathcal{K}$, $\widehat{\mathcal{K}}$	Partitionsraum
$\mathcal{K}^{\sigma(\mathbf{d})}$	Quellenteilraum
$\mathcal{K}^{\delta(\mathbf{d})}$	Senkenteilraum
$\mathcal{L}_s$	Indexraum der linksseitigen Variable $y_s[l(\mathbf{i})]$
$\mathcal{P}$	Rechenfeld
$\mathcal{R}^{k_{s'}}_{s's_t}$	Indexraum der rechtsseitigen Variable $y_{s'}[r^{k_{s'}}_{s's_t}(\mathbf{i})]$
$\mathcal{R}^{k_r}_{r\,s_t}$	Indexraum der Eingabevariable $y_r[r^{k_r}_{r\,s_t}]$
$\mathcal{S}$	Menge der Zuweisungen s eines Algorithmus
$\mathcal{Y}_L$	Menge aller Variablen $y_a[.]$, die auf der linken Seite einer Zuweisung $s \in \mathcal{S}$ eines Algorithmus auftreten.
$\mathcal{Y}_R$	Menge aller Variablen $y_a[.]$, die auf der rechten Seite einer Zuweisung $s \in \mathcal{S}$ eines Algorithmus auftreten.
$\mathcal{Y} = \mathcal{Y}_L \cup \mathcal{Y}_R$	Menge aller Variablen $y_a[.]$ eines Algorithmus
$\mathcal{Y}_E$	Menge aller Eingabevariablen $y_a[.]$ eines Algorithmus.
$\mathcal{Y}_A$	Menge aller Ausgabevariablen $y_a[.]$ eines Algorithmus.
$\mathcal{Y}_G = \mathcal{Y}_E \cup \mathcal{Y}_A$	Menge aller globalen Variablen $y_a[.]$ eines Algorithmus.
$\mathcal{Y}_I$	Menge aller internen Variablen $y_a[.]$ eines Algorithmus.
$\mathcal{C}^m_{\lvert\mathcal{M}\rvert}$	Menge der Kombination von m Elementen aus der Menge $\mathcal{M}$
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{R}$	Menge der reellen Zahlen

$\mathbb{Q}$	Menge der rationalen Zahlen
$\mathbb{Q}_+$	Menge der positiven rationalen Zahlen ohne Null
$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathbb{N}_+$	Menge der natürlichen Zahlen ohne Null

Funktionen

$f(\mathbf{m})$	Allgemeine Funktion angewendet auf Vektor $\mathbf{m}$
$\mathcal{M}_2 = \text{conv}(\mathcal{M}_1)$	Konvexe Hülle von $\mathcal{M}_1$
$m = \text{Rg}(\mathbf{M})$	Rang der Matrix $\mathbf{M}$
$\mathbf{m} = \text{diag}(\mathbf{M})$	Vektor der Hauptdiagonalen der Matrix $\mathbf{M}$
$\mathbf{M} = \text{diag}(\mathbf{m})$	Matrix mit Vektor $\mathbf{m}$ als Hauptdiagonale
$n = \dim(\mathcal{M})$	Dimension von $\mathcal{M}$
$f(\mathcal{M}_1)$	Menge der Elemente der Funktion $f(\mathbf{m})$ für alle $\mathbf{m} \in \mathcal{M}$
$n = \text{sign}(m)$	Vorzeichen eines Skalars m ($n \in \{-1, 0, 1\}$)
$s = \sigma(e)$	Quelle der Datenabhängigkeit e (Zuweisung)
$s_t = \delta(e)$	Ziel der Datenabhängigkeit e (Teilzuweisung)
$\text{ggT}(m_1, m2)$	größter gemeinsamer Teiler von m_1 und m_2
$\rho_s(\mathbf{i})$	Reindizierungsfunktion für Zuweisung s
$\gamma(\mathbf{i}^\kappa)$	Allokationsfunktion
$t_s(\mathbf{i})$	Ablaufplanfunktion
$\vartheta(\mathbf{i}^\kappa)$	Partitionierung
$\forall m$	Für alle m
$\exists m$	Es existiert mindestens ein m
$\exists! m$	Es existiert genau ein m
$\|m\|$	Betrag des Skalars m
$\|\mathcal{M}\|$	Anzahl der Elemente der Menge $\mathcal{M}$
$\mathcal{A} \oplus \mathcal{B}$	direkte Summe $\{\mathbf{a} + \mathbf{b}, \mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}\}$

Abkürzungen

BPA	Baum der partitionierten Abhängigkeitsvektoren
DSP	Digitaler Signalprozessor
DvB	Datenwort voller Breite
FE	Funktionseinheit
GLP	Ganzzahlig lineare Programmierung
KEA	Kantenerkennungsalgorithmus
LPGS	Lokal parallele, global sequenzielle Partitionierung
LSGP	Lokal sequenzielle, global parallele Partitionierung
MMC	Mehrstufige modifizierte Copartitionierung
PE	Prozessorelement
RAG	Reduzierter Abhängigkeitsgraph
STAF	Short Term Analysis Filtering

SPRF	Schwach programmoerbares Rechenfeld
RF	Rechenfeld
TW	Teilwort
TWP	Teilwortparallelität
UItA	Uniform-iterativer Algorithmus
const	konstant

Kapitel 1

Einleitung

Der technologische Fortschritt gestattet die Implementierung zunehmend komplexerer Prozessorarchitekturen auf einem Schaltkreis. Dabei werden sowohl immer umfangreichere Speicherarchitekturen auf den Chips implementiert als auch zusätzliche Funktionseinheiten eingebaut bzw. deren Funktionalität erweitert. Ein weiterer Schritt in dieser Entwicklung ist der Entwurf sogenannter „Systeme auf einem Schaltkreis“ (engl.: System on the Chip. kurz: SOC). Damit können nun Mehrkernsysteme (engl.: Multicore Systems) entwickelt werden, welche mehrere Prozesssorkerne besitzen. Diese Prozesssorkerne können gleiche oder unterschiedliche Aufgaben parallel abarbeiten.

Aus diesem Grund bieten heutige Prozessoren vielfältige Möglichkeiten zur Parallelverarbeitung. Abhängig von der Art der ausführbaren Operationen (gleiche oder unterschiedliche Funktionen) und deren Ansteuerungen (Einzelansteuerung, gemeinsamer Steuerbefehl oder Verwendung von VLIW-Befehlen (engl.: very large instruction word)) werden verschiedene Ansätze der Parallelverarbeitung unterschieden.

Das Ziel dieser Arbeit ist die systematische Abbildung von Algorithmen auf Rechenfelder (kurz: RF, engl.: processor arrays) mit Teilwortparallelität (kurz: TWP, engl.: sub-word parallelism). Die Abbildung 1.1 zeigt eine solche Architektur mit den drei Ebenen der Parallelverarbeitung:

(1) Parallelverarbeitung im Rechenfeld aufgrund mehrerer Prozessorelemente

(2) Parallelverarbeitung im Prozessorelement aufgrund mehrerer Funktionseinheiten

(3) Parallelverarbeitung in der Funktionseinheit aufgrund teilwortparalleler Operationen

Rechenfelder sind regulär angeordnete Prozessorelemente, die jeweils nur mit ihren Nachbarn verbunden sind. Der Aufbau (Anzahl und Art der Funktionseinheiten und Größe des lokalen Speichers) und die Funktion (ausführbare Befehle) der Prozessorelemente sind weitestgehend identisch.

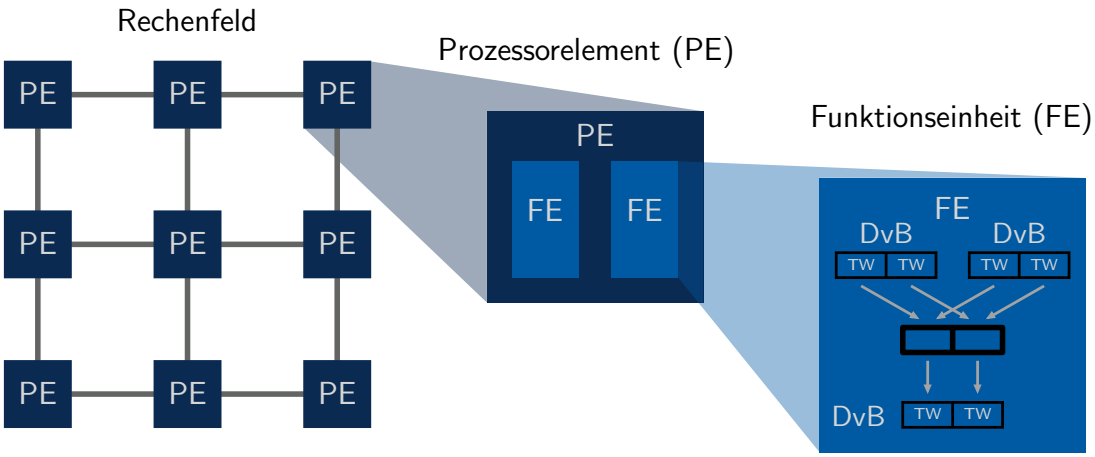


Abbildung 1.1: Rechenfeld mit Teilwortparallelität

Die Teilwortparallelität (kurz: TWP, engl.: sub-word parallelism) bezeichnet die Parallelverarbeitung innerhalb einer Funktionseinheit. Hierbei wird der Datenpfad der Funktionseinheit in mehrere schmale Datenpfade aufgeteilt, um Daten mit einer niedrigen Wortbreite auf einer Funktionseinheit mit hoher Datenpfadbreite parallel verarbeiten zu können. Die Daten niedriger Wortbreite bezeichnen wir als Teilwörter (kurz: TWs). Die Teilwörter, welche gleichzeitig verarbeitet werden sollen, werden in Datenwörtern voller Breite (kurz: DvBs) zusammengefasst. Die Berechnungen in den Funktionseinheiten werden auf Basis dieser DvBs durchgeführt. Die Operationen, welche auf die Teilwörter in einem Takt angewendet werden, sind meist identisch und hängen vom jeweiligen Befehlssatz der Funktionseinheit ab. Der Grad der Parallelität kann mit jedem Befehl geändert werden.

Da die Daten als Teilwörter in den DvBs zusammengefasst sind, müssen diese DvBs zwischen den Berechnungen meist umorganisiert werden. Dieses Umorganisieren wird auch als „Packen der Daten“ bezeichnet. Hierfür stehen spezielle Packbefehle zur Verfügung. Der Aufwand für das Packen der Daten sollte minimiert werden, denn die zusätzlichen Packoperationen können den Geschwindigkeitsgewinn aufgrund der parallelen Berechnung teilweise oder vollständig wieder zunichte machen.

Die teilwortparallele Verarbeitung von Daten wird auch in einer Vielzahl weiterer Architekturen genutzt. Als Beispiele seien hier, der Core2-Prozessor von Intel [Int07a], der Cell-Prozessor von IBM [Int07b] und der TMS320C64x-Prozessor von Texas Instruments [Ins01] erwähnt. Bei diesen Architekturen wurde die Hardware für die teilwortparallele Verarbeitung entsprechend erweitert. In [BDT06] wird eine Softwarerealisierung für die Teilwortparallelität vorgestellt, welche teilwortparallele Verarbeitung mit beliebigen Teilwortbreiten in fast jeder Funktionseinheit ermöglicht.

In dieser Arbeit wird eine systematische Abbildung von Algorithmen auf Rechenfelder mit Teilwortparallelität präsentiert. Die Abbildung basiert auf den Abbildungsmethoden für Rechenfelder, insbesondere auf der mehrstufigen modifizierten Copartitionierung [Sie03]. In dieser Arbeit wird die mehrstufige modifizierte Copartitionierung erweitert

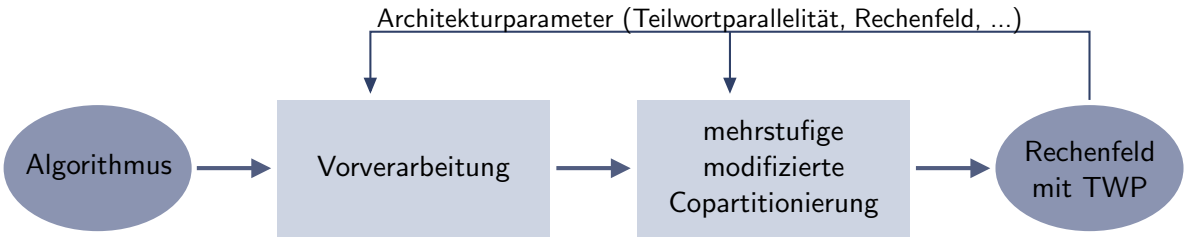


Abbildung 1.2: Systematische Abbildung eines Algorithmus

und systematisiert, um eine mehrstufige Parallelität zu realisieren und eine Anpassungen an lokale und globale Speicher zu ermöglichen. Des Weiteren wird in dieser Arbeit der Abbildungsschritt *Vorverarbeitung* entwickelt, welcher der mehrstufigen modifizierten Copartitionierung vorangestellt wird, um eine effiziente Abbildung des Algorithmus zu erzielen.

Abbildung 1.2 illustriert den Abbildungsprozess bestehend aus den zwei Schritten, der *Vorverarbeitung* und der *mehrstufigen modifizierten Copartitionierung*. Mit der Vorverarbeitung wird der Algorithmus so umformuliert, sodass die nachfolgende mehrstufige modifizierten Copartitionierung zu effizienten Abbildungen auf das Rechenfeld mit Teilwortparallelität führt. Die *Vorverarbeitung* ist zielarchitekturunabhängig. Auch wenn einzelne Parameter der Zielarchitektur mit in die *Vorverarbeitung* einbezogen werden, findet bei der *Vorverarbeitung* noch keine Anpassung an diese Parameter der Zielarchitektur statt. Erst bei der mehrstufigen modifizierten Copartitionierung wird der Algorithmus stufenweise an die Parameter der Zielarchitektur angepasst. Hierbei dient die erste Copartitionierungsstufe der Anpassung an die Teilwortparallelität. Die zweite Copartitionierungsstufe wird sowohl für die Ausnutzung der lokalen Speicher in den Prozessorelementen verwendet als auch für die Abbildung des Algorithmus auf das Rechenfeld genutzt. Nachfolgende Copartitionierungsstufen werden für die Anpassung des Algorithmus an den globalen Speicher verwendet.

Die Arbeit ist wie folgt gegliedert:

Zuerst wird in Kapitel 2 die Zielarchitektur ausführlich vorgestellt. Für die Funktionseinheiten mit Teilwortparallelität wurde eine XML-basierte Beschreibung entwickelt, aus der die Eigenschaften der Funktionseinheiten ermittelt werden können. Ein weiterer wichtiger Aspekt ist die Beschreibung der Speicherzugriffe, welche bei der Teilwortparallelität eine besondere Bedeutung haben, da die Daten auf Basis von DvBs oder auf Basis von Teilwörtern gelesen und geschrieben werden können.

Im nachfolgenden Kapitel 3 wird die Algorithmenklasse der *uniform, iterativen Algorithmen (kurz: UItA)* eingeführt. Diese Algorithmenklasse ist der Ausgangspunkt für die Abbildungsstrategie. Ein Algorithmus kann sehr unterschiedlich formuliert sein, wobei alle Formulierungen einen UItA darstellen. Diese verschiedenen Formulierungen können jedoch aufgrund ihrer Eigenschaften zu sehr unterschiedlichen Abbildungen führen. Aus diesem Grund wird eine Algorithmenkonvention präsentiert, welche der eigentliche Ausgangspunkt für die Abbildung des Algorithmus auf die Zielarchitektur ist. Man beachte, dass jeder Algorithmus der Algorithmenklasse so umformuliert werden kann, dass er die

Algorithmenkonvention erfüllt.

Für die Abbildung eines Algorithmus auf die Zielarchitektur wird der Algorithmus mehrfach transformiert. Die Grundtransformationen, das sind die Reindizierung und die mehrstufige Partitionierung, werden im Kapitel 4 vorgestellt. Bei der Vorstellung der mehrstufige Partitionierung wird im Abschnitt 4.2.2 eine neuartige Methode zur Partitionierung der Datenabhängigkeiten des Algorithmus präsentiert.

Auf Basis der mehrstufigen Partitionierung wird in Kapitel 5 die mehrstufige modifizierte Copartitionierung (kurz: MMC) vorgestellt. Die mehrstufige modifizierte Copartitionierung kombiniert die Grundversionen der Partitionierung, die lokal sequenzielle, global parallele (kurz: LSGP) und die lokal parallele, global sequenzielle (kurz: LPGS) Partitionierung miteinander. Weiterhin beinhaltet die MMC die Ablaufplanung und die Bindung. Mit dem Ablaufplan wird exakt festgelegt, zu welchem Zeitpunkt welche Zuweisung des Algoritmus in welcher Funktionseinheit von welchem Prozessorelement abgearbeitet wird. Die Zuordnung der Funktionseinheiten zu den Zuweisungen des Algorithmus wird als Bindung bezeichnet. In dieser Arbeit wird der Ablaufplan für eine allgemeinere Beschreibung durch die Leerlaufverzögerung $\boldsymbol{\iota}$ erweitert. Die Leerlaufverzögerung ermöglicht das Einfügen von Leertakten in den Ablaufplan. Eine Leerlaufverzögerung $\boldsymbol{\iota} \neq \mathbf{0}$ verringert die Effizienz des Algorithmus auf dem Rechenfeld, ist aber für die konfliktfreie Abbildung einzelner Algorithmen notwendig.

Die Ausnutzung der Teilwortparallelität im Abbildungsprozess mit dem Ziel der Beschleunigung der Abarbeitung des Algorithmus ist ein Schwerpunkt dieser Arbeit. Aus diesem Grund wird im Kapitel 6 die Transformation eines Ausgangsalgorithmus, der seine Operationen auf „einzelnen“ Daten (kurz: Einzeldatenoperationen) ausführt, in einen Algorithmus mit teilwortparallelen Operationen ausführlich abgeleitet. Diese Transformation basiert auf der ersten Stufe der mehrstufigen modifizierten Copartitionierung, wobei für diese Stufe die notwendigen Restriktionen angegeben werden. Bei der Transformation des Algorithmus werden die notwendigen Packoperationen für die teilwortparallele Version des Algorithmus systematisch ermittelt. Weiterhin wird die Transformation der Ein- und Ausgabezuweisungen des Ausgangsalgorithmus ausführlich vorgestellt. Diese Transformation hängt von der Art der Speicherung der Daten im Hauptspeicher sowie deren Lese- und Schreibmöglichkeiten ab. Beide Eigenschaften haben einen entscheidenden Einfluss auf die Beschleunigung des Algorithmus.

Die gesamte Abbildungsstrategie zur Abbildung eines Algorithmus auf ein Rechenfeld mit Teilwortparallelität wird nun in Kapitel 7 unter Verwendung der zuvor eingeführten Methoden ausführlich vorgestellt. Die Abbildungsstrategie gliedert sich, wie in Abbildung 1.2 dargestellt, in zwei Schritte, die Vorverarbeitung und die mehrstufige modifizierte Copartitionierung.

Die Vorverarbeitung (Abschnitt 7.1) wird in dieser Arbeit neu entwickelt. Sie unterteilt sich in zwei Teilschritte, die *Reindizierung* und die *Anpassung an die Algorithmenkonvention*. Ziel der Vorverarbeitung ist ein im Sinne der Abbildung auf das Rechenfeld gut formulierter Algorithmus, denn solch ein Algorithmus erlaubt eine schnelle und effiziente Abbildung auf das Rechenfeld mittels der mehrstufigen modifizierten Copartitionierung.

Für die Reindizierung wurde eine Optimierungsaufgabe basierend auf ganzzahliger linearer Optimierung entwickelt. Diese optimiert stufenweise mehrere Parameter des Algorithmus bezüglich der effizienten Abbildung des Algorithmus auf ein Rechenfeld bei der nachfolgenden mehrstufigen modifizierten Copartitionierung. Dabei fließen in die Optimierung teilweise schon Architekturparameter mit ein. Im Abschnitt „Anpassung an die Algorithmenkonvention“ wird der Algorithmus so umformuliert, dass er die Vorgaben der Algorithmenkonvention von Kapitel 3 erfüllt.

Der zweite Schritt der Abbildungsstrategie, die Anwendung der mehrstufigen modifizierten Copartitionierung, wird in Abschnitt 7.2 vorgestellt. Die mehrstufige modifizierte Copartitionierung wird in ihre einzelnen Stufen und in die abschließende Ablaufplanung unterteilt. Mit jeder Stufe der mehrstufigen modifizierten Copartitionierung erfolgt eine Anpassung des Algorithmus an Parameter der Zielarchitektur. So dient die erste Stufe der Ausnutzung der Teilwortparallelität und die zweite Stufe sowohl der Realisierung der Parallelverarbeitung im Rechenfeld als auch der Ausnutzung der lokalen Register. Weitere Copartitionierungsstufen können zur Anpassung an die Speicherarchitektur verwendet werden. Für die Anpassung an die Speicher wird eine Methode zur Bestimmung der Datenmengen vorgestellt, welche zum einen zwischen Prozessorelementen des Rechenfeldes und zum anderen zwischen Rechenfeld und Speicherarchitektur transferiert werden müssen. Diese Methode basiert auf der neuen Methode zur Partitionierung von Datenabhängigkeiten, die im Abschnitt 4.2.2 vorgestellt wird. Die abschließende Ablaufplanung ordnet den Zuweisungen des Algorithmus die Funktionseinheiten der Prozessorelemente zu und legt den exakten Ablaufplan fest.

In Kapitel 8 wird die in der Arbeit präsentierte Abbildungsstrategie auf drei praxisrelevante Algorithmen angewendet. Zwei Algorithmen werden auf ein Prozessorelement mit Teilwortparallelität abgebildet. Das dritte Beispiel zeigt die Abbildung des Kantenerkennungsalgorithmus auf eine Rechenfeld mit Teilwortparallelität. Alle Realisierungen wurden bezüglich ihres Geschwindigkeitsgewinns bewertet.

Abschließend erfolgt eine Zusammenfassung und es wird ein Ausblick auf offene Zielstellungen gegeben.

Kapitel 2

Zielarchitekturen

Als Zielarchitekturen im Prozess der Abbildung von Algorithmen auf Hardware werden Rechenfelder (engl.: processor arrays) bzw. der Spezialfall des Rechenfeldes, Ein-Prozessor-Maschinen (kurz: Prozessor) betrachtet. Das Rechenfeld besteht aus mehreren Prozessorelementen (PEs), die über Datenkanäle miteinander verbunden sind. Die Ein-/Ausgabeschnittstellen am Rand des Rechenfeldes bzw. des Prozessors verbinden die Recheneinheit mit der hierarchisch aufgebauten Speicherarchitektur.

Die Speicherarchitektur besteht aus mehreren Zwischenspeichern L_1 bis L_n und einem Hauptspeicher M. Der Hauptspeicher M mit der Größe M_M kann alle Daten des Algorithmus speichern. Die Größe M_{L_k} der Zwischenspeicher k steigt im Allgemeinen mit zunehmenden k. Der Zwischenspeicher L_1 kann auch aus mehreren kleineren Zwischenspeichern bestehen, welche direkt an das Rechenfeld angebunden sind. In dieser Arbeit betrachten wir nur Speicherarchitekturen mit einem L_1-Speicher, wie in Abbildung 2.1 dargestellt. Durch geringfügige Veränderungen an der Abbildungsstrategie, welche in Kapitel 7 vorgestellt wird, lassen sich die Algorithmen aber auch an Speicherarchitekturen mit mehreren verteilten L_1-Speichern anpassen. Hardwarenahe Eigenschaften der Zwischenspeicher und des Hauptspeichers, wie z. B. Zugriffsbeschränkungen und Speicherverfehlungen (engl.: cache misses), werden in dieser Arbeit nicht weiter betrachtet. Nur die lineare Anordnung der Daten im Speicher, siehe Abschnitt 2.4, findet Berücksichtigung bei der Abbildungsstrategie.

Der Prozessor bzw. die Prozessorelemente verfügen über lokalen Speicher L_R, deren Größe durch M_R angegeben wird, und besitzen ein oder mehrere Funktionseinheiten (FEs). Die Funktionseinheiten können Standardbefehle, wie Addition und Multiplikation, verarbeiten. Zusätzlich verfügen die FEs über Befehle, die paralleles Verarbeiten von Daten im Datenpfad ermöglichen. Diese Parallelverarbeitung im Datenpfad bezeichnen wir *Teilwortparallelität*. Sie wird im Abschnitt 2.2 näher vorgestellt.

In Abbildung 2.1 ist eine solche Architektur, bestehend aus einem 6×4 Rechenfeld und einer dreistufigen Speicherarchitektur mit den zwei Zwischenspeichern L_1 und L_2 und dem Hauptspeicher dargestellt. Die Größe der Zwischenspeicher wird mit M_{L_l}, $l = 1, 2$ und die Größe des Hauptspeichers mit M_M angegeben. Die Kommunikation zwischen

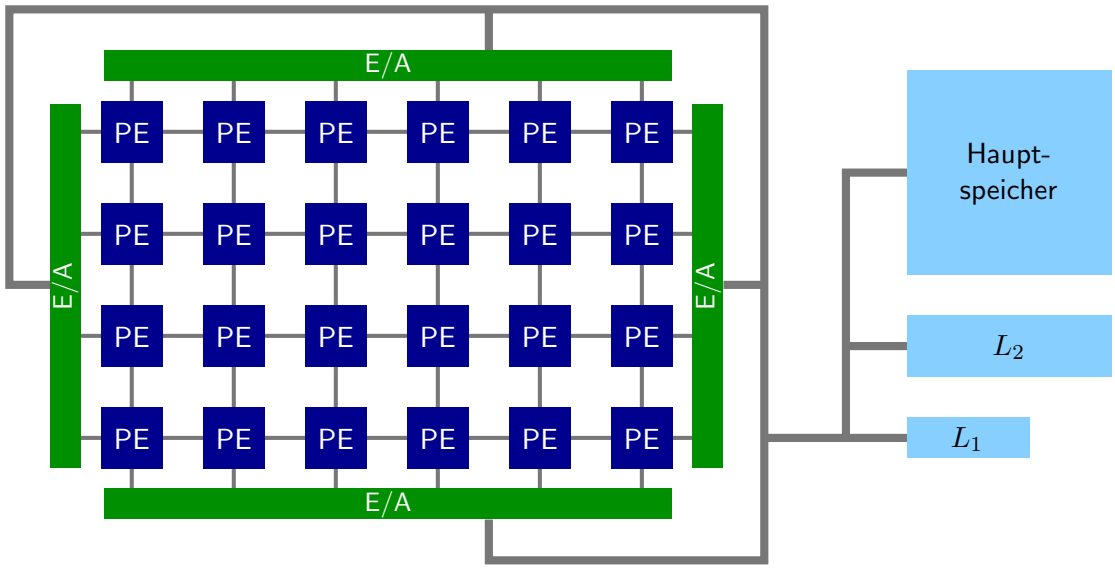


Abbildung 2.1: Rechenfeld mit Speicherarchitektur

Rechenfeld und Speicherarchitektur erfolgt über die E/A-Schnittstellen an den Rändern des Rechenfeldes. Die Prozessorelemente des Rechenfeldes sind horizontal und vertikal mit den Nachbarprozessorelementen bzw. den E/A-Schnittstellen verbunden.

Die Prozessorelemente unserer Beispielarchitektur, siehe Abbildung 2.2, enthalten vier Funktionseinheiten (ALU 1, ALU 2, MUL und SHIFT) und eine lokale Registerbank, deren Größe durch M_R gegeben ist.

Eine solche Architektur erlaubt Parallelverarbeitung auf drei Ebenen: (1) auf der Ebene der PEs des Rechenfeld, (2) auf der Ebene der Funktionseinheiten des PEs und (3) in einer FE mittels Teilwortparallelität, wie Abbildung 1.1 auf Seite 2 illustriert.

Da ein Prozessor ein Sonderfall des Rechenfeldes ist, schließt im Folgenden die Bezeichnung Rechenfeld die Ein-Prozessor-Architektur mit ein.

2.1 Rechenfeld

Ein Rechenfeld wird durch die Menge $\mathcal{P} \in \mathbb{N}^n$ beschrieben, wobei n die Dimension des Rechenfeldes ist. Für ein Rechenfeld $\mathcal{P}$ gilt:

$$\mathcal{P} = \left\{ \mathbf{p} = \begin{pmatrix} p_1 & p_2 & \dots & p_n \end{pmatrix}^t \in \mathbb{N}^n \;\middle|\; 0 \leq p_j < \vartheta_j^{\square}, 1 \leq j \leq n \right\}. \tag{2.1}$$

Das Element $\mathbf{p} \in \mathcal{P}$ gibt die Position des Prozessorelements im Rechenfeld an. Die Größe des Rechenfeldes wird durch den Vektor $\boldsymbol{\vartheta}^{\square} \in \mathbb{N}_+^n$ angegeben. So wird das Rechenfeld aus Abbildung 2.1 mit der Größe 6×4 mittels $\boldsymbol{\vartheta}^{\square} = \binom{6}{4}$ beschrieben. Im Allgemeinen ist $\boldsymbol{\vartheta}^{\square} > 1$, nur bei einem linearen Rechenfeld mit einem Prozessorelement ($\boldsymbol{\vartheta}^{\square} = \begin{pmatrix} 1 \end{pmatrix}$) erhalten wir: $\vartheta_1^{\square} = 1$.

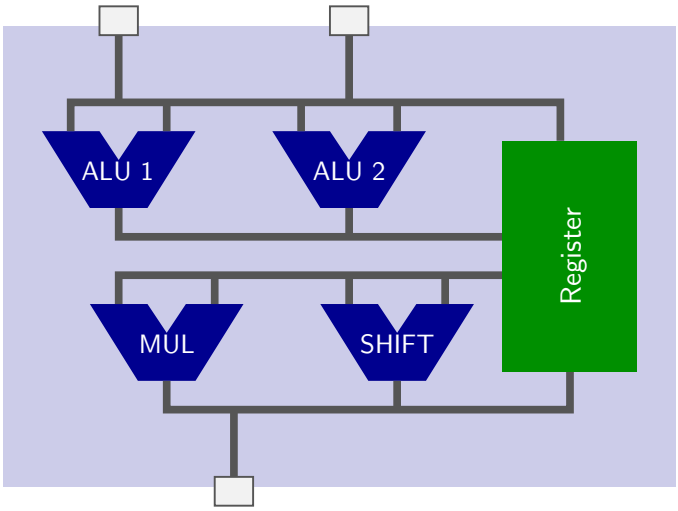


Abbildung 2.2: Prozessorelement

Die Datenkanäle, welche die Verbindung zwischen den Prozessorelementen $\mathbf{p} \in \mathcal{P}$ darstellen, werden durch den Kanaltyp $\mathbf{q} \in \mathcal{Q} \in \mathbb{Z}^n$, später meist kurz Kanal genannt, und der Anzahl $N_\mathbf{q} \in \mathbb{N}_+$ an Kanälen von diesem Typ angegeben. In Rechenfeldern gibt es meist nur Datenkanäle zwischen direkt nachbarten Prozessorlementen ($|q_i| \leq 1$), doch bei unserer Zielarchitektur sind auch Kanäle zwischen weiter entfernten Prozessorelementen ($\mathbf{q} \in \mathbb{Z}^n$) möglich.

Existiert ein Kanal zwischen den zwei PEs $\mathbf{p}_1$ und $\mathbf{p}_2$, so gilt $\mathbf{p}_1 + \mathbf{q} = \mathbf{p}_2$ und $N_\mathbf{q} \geq 1$. Ein Kanal kann unidirektional $\mathbf{q} \in \mathcal{Q}^{\text{un}} \subset \mathcal{Q}$ oder bidirektional $\mathbf{q} \in \mathcal{Q}^{\text{bi}} \subset \mathcal{Q}$ sein. Bei einem unidirektionalen Kanal $\mathbf{q} \in \mathcal{Q}^{\text{un}}$ ist ein Datentransfer von PE $\mathbf{p}_1$ nach PE $\mathbf{p}_2$ nur möglich, wenn gilt: $\mathbf{p}_1 + \mathbf{q} = \mathbf{p}_2$. Ist der Kanal $\mathbf{q} \in \mathcal{Q}^{\text{bi}} \subset \mathcal{Q}$ bidirektional, so kann der Kanal für den Transport der Daten in Richtung $\mathbf{q}$ als auch $-\mathbf{q}$ verwendet werden.

Beispiel 2.1 (Rechenfeld)
Abbildung 2.1 zeigt ein Rechenfeld der Größe $\boldsymbol{\vartheta}^\square = \binom{6}{4}$. *Das Rechenfeld ist gegeben durch* $\mathcal{P} = \{\mathbf{p} = \binom{p_1}{p_2} \mid 0 \leq p_1 < \vartheta_1^\square = 6 \wedge 0 \leq p_1 < \vartheta_2^\square = 4\}$. *Das Rechenfeld hat zwei Kanäle* $\mathcal{Q} = \{\mathbf{q}_1 = \binom{1}{0}, \mathbf{q}_1 = \binom{0}{1}\}$. *Der Kanal* $\mathbf{q}_1 \in \mathcal{Q}^{bi}$ *ist bidirektional und der Kanal* $\mathbf{q}_2 \in \mathcal{Q}^{\text{un}}$ *ist unidirektional. Jeder Kanaltyp existiert nur einmal* $N_{\mathbf{q}_1} = N_{\mathbf{q}_2} = 1$.

Die Eigenschaften der Prozessorlemente, wie Anzahl und Art der Funktionseinheiten, sowie die Verbindungsarchitektur kann für Teilbereiche $\mathcal{P}_i^p \subset \mathcal{P}$ (Funktionseinheiten) bzw. $\mathcal{P}_i^v \subset \mathcal{P}$ (Verbindungsarchitektur) des Rechenfeldes unterschiedlich sein [KHK+06a].

2.2 Teilwortparallelität

Das Prinzip der Teilwortparallelität (TWP) besteht darin, in einer Funktionseinheit mit einer Verarbeitungswortbreite von z. B. 64 Bit, Wörter niedriger Genauigkeit (z. B. acht 8 Bit breite, vier 16 Bit breite oder zwei 32 Bit breite Wörter) parallel zu verarbeiten.

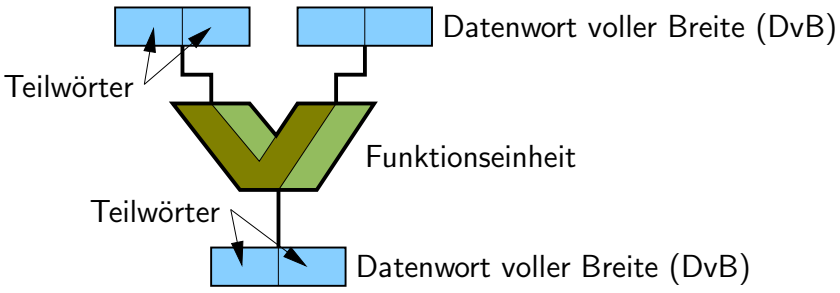


Abbildung 2.3: Funktionseinheit mit zwei Teilwörtern

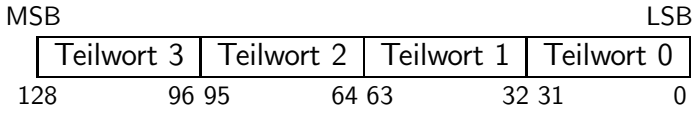


Abbildung 2.4: Datenwort voller Breite (128 Bit) mit vier Teilwörtern

Die Wörter mit niedriger Genauigkeit bezeichnen wir dabei als *Teilwörter*. Abbildung 2.3 illustriert die Parallelverarbeitung von zwei Teilwörtern in einer solchen Funktionseinheit. Die Art der Parallelverarbeitung (Operation und Anzahl der Teilwörter) ist abhängig vom auszuführenden Befehl und kann somit prinzipiell mit jedem Befehl geändert werden. Diese Befehle bezeichnen wir als teilwortparallele Befehle (kurz: TWP-Befehle).

Für die Parallelverarbeitung der Teilwörter im Datenpfad der Funktionseinheit müssen diese in Datenwörtern voller Breite (kurz: DvBs) *gepackt* sein. Ein Datenwort voller Breite enthält ein oder mehrere (meist 2^n) Teilwörter. Die Wortbreite der Teilwörter ist im Allgemeinen gleich, es wurde aber auch schon ein Befehlssatz für DvBs mit 1x 16 Bit und 2x 8 Bit Teilwörtern definiert[1]. Die Teilwörter sind im DvB immer von rechts nach links angeordnet, wie in Abbildung 2.4 dargestellt.

Im Allgemeinen muss die Anordnung der Teilwörter im DvB von einem TWP-Befehl zu einem anderen TWP-Befehl verändert werden. Hierfür stehen die sogenannten Packbefehle zur Verfügung. Die Verarbeitung dieser Packbefehle erfordert Zeit und entsprechende Ressourcen. Bei der Parallelisierung ist somit darauf zu achten, dass der Geschwindigkeitsgewinn und die Energieeinsparung durch die Nutzung von TWP-Befehlen nicht durch die notwendigen Packbefehle wieder zunichte gemacht werden.

Neben der Hardwarerealisierung der Teilwortparallelität wurde in [BDT06] eine Implementierung von TWP in Software vorgestellt. Diese Softwarerealisierung hat den Vorteil der höheren Flexibilität, da die Wortbreite der Teilwörter frei gewählt werden kann. So kann ein DvB mit 32 Bit Breite in 3 Teilwörter mit je 9 Bit Breite unterteilt werden. Weiterhin kann die Wortbreite je Teilwort unterschiedlich sein, so kann das DvB mit 32 Bit Breite 2 Teilwörter mit je 4 Bit Breite und 2 Teilwörter mit 9 Bit Breite aufgeteilt werden.[2] Doch diese Softwarerealisierung der teilwortparallelen Operationen ist

[1] siehe Quantized Color Pack eXtension (QCPX) von [KBW03] für 16 Bit Farbdatentyp in YCbCr.

[2] Zwischen den Teilwörtern in einem DvB müssen bei der Softwarerealisierung ein bis zwei „Überlauf"-Bits reserviert werden.

gegenüber der Hardwarerealisierung deutlich aufwändig.

In der wissenschaftlichen Literatur wird die Teilwortparallelität nach der Flynnsche Klassifikation [Fly66][Dun90] in die „Ein Befehlsstrom, mehrere Datenströme“ (engl.: single instruction stream, multiple data streams, kurz: SIMD) Architekturklasse eingeordnet. Die Teilwortparallelität unterscheidet sich aber bezüglich der Befehle und der Verarbeitung von Datenströmen gegenüber einem Vektorprozessor oder Rechenfeld, welche ebenfalls zu der SIMD Architekturklasse zählen. Bei letztgenannten Architekturen bedeutete ein Befehl meist *eine* Operation, welche parallel auf einen Satz von Daten angewendet wurde. Die Verarbeitung der Daten erfolgt in den Prozessorelementen des Rechenfeldes oder des Vektorprozessors *unabhängig* voneinander.

Auch die Teilwortparallelität verarbeitet, wie das SIMD-Prinzip vorschreibt, mit einem Befehl mehrere Datenströme, doch bieten die Befehle weitere Varianten der Parallelverarbeitung, wie die Beispielarchitekturen und ihre Befehlssätze in Abschnitt 2.5 zeigen. Mit den TWP-Befehlen ist es möglich,

- unterschiedliche Operationen mit einem Befehl auf die Datenströme anzuwenden (z. B. `addsub` Addition des niederwertigen Teilworts und Subtraktion des höherwertigen Teilworts).
- dass die Datenströme nicht unabhängig voneinander verarbeitet werden (z. B. `sad` Summe der absoluten Differenzen von 8x 8 Bit Bildpunkten).
- dass die Anzahl der Eingabedatenströme ungleich der Ausgabedatenströme ist (z. B. `muladd` Multiplikation der 4x 8 Bit Eingabeteilwörter miteinander und Addition der beiden niederwertigen und der beiden höherwertigen Multiplikationsergebnisse. Das Ausgabedatenwort ist ein 2x 16 Bit Datenwort.).

Des Weiteren müssen die Eingabedatenströme, welche durch die Teilwörter in den DvBs gegeben sind, mit den Packbefehlen so aufbereitet werden, dass eine teilwortparallele Verarbeitung in der Funktionseinheit möglich wird. Deshalb unterscheiden wir bei den TWP-Befehlen folgende Gruppen:

- **SOMD-Befehle (Single Operation Multiple Data):** Befehle bei denen die Anzahl an Teilwörtern in den Ein- und Ausgabe-DvBs gleich ist und bei denen die gleiche Operation (z. B. Addition) auf die Teilwörter des DvBs angewendet wird.
- **Sonderbefehle:** Befehle mit Rechenoperationen, die keine SOMD-Befehle sind. Diese Befehle haben keine gleichen Operationen für die Teilwörter (z. B. AddSub) und/oder keine gleiche Anzahl an Eingabe- und Ausgabeteilwörtern (z. B. MulAdd mit 2x 4 Eingabeteilwörter und 1x 2 Ausgabeteilwörter).
- **Packbefehle:** Befehle zum Umsortieren der Teilwörter eines DvBs oder zum Variieren der Teilwörter zweier DvBs.

- **Datenformatänderungsbefehle:** Befehle, welche das Datenformat eines Datenworts ändern. Das kann die Erhöhung oder die Reduzierung der Wortbreite der Teilwörter sein. Es gibt aber auch Befehlssätze, die eine Transformation von Ganzzahlen in Fließkommazahlen und umgekehrt ermöglichen [Int04].
- **Übertragungsbefehle:** Befehle zum Lesen und Schreiben der Teilwörter bzw. DvBs vom bzw. in den Speicher. Abhängig vom Befehlssatz ist das Lesen bzw. Schreiben von ein oder mehereren Teilwörtern oder nur von DvBs möglich. Auch gibt es Architekturen, die nur ausgerichtete DvBs[3] lesen bzw. schreiben können.

2.3 Funktionseinheiten

In der Arbeit werden die Funktionseinheiten in einer XML-Notation beschrieben. Der Algorithmus 2.1 zeigt die Beschreibung einer Funktionseinheit (`FunctionalUnit`) strukturell. Eine Funktionseinheit $m \in \mathcal{M}$ hat einen eindeutigen Namen (`name`), eine In-

Algorithmus 2.1 Beschreibung einer Funktionseinheit in XML

```
<FunctionalUnit name="Name der FE" instance="Instanz" size="Fläche">
    <Instruction name="Name des Befehls"
                                    latency="Latenz" offset="Eingabeverzögerung" />
    ....
</FunctionalUnit>
```

stanz n_m (`instance`), eine Fläche c_m (`size`) und eine Menge von Befehlen $F_s \in \mathcal{F}(m)$ (`Instruction`), wobei der Befehl F_s die Funktion F_s ausführt. Die Angabe der Instanz n_m einer Funktionseinheit ist notwendig, um mehrere gleiche Funktionseinheiten in einem PE unterscheiden zu können. Ein Befehl hat einen Befehlsnamen F_s (`name`). Die Abarbeitungsdauer eines Befehls F_s durch die Funktionseinheit m wird mit der Latenz $l_{m(s)}$ (`latency`) angegeben. Hat die Funktionseinheit eine Pipelinestruktur, so kann die Ausführung des nachfolgenden Befehls begonnen werden, bevor die aktuelle beendet ist. Die Eingabeverzögerung $o_{m(s)}$ (`offset`) gibt die Zeit an, nach der die Ausführung des nächsten Befehls frühestens beginnen darf. Einfache Funktionseinheiten sind Addierer und Multiplizierer. Eine komplexere Funktionseinheit, welche mehrere Befehle ausführen kann, ist zum Beispiel die ALU-Einheit. Funktionseinheiten können auch TWP-Befehle ausführen.

Beispiel 2.2
Die Funktionseinheiten des PEs in Abbildung 2.2 sollen die folgenden Befehle ausführen können:

ALU: $c_1 = 5$ *SHIFT:* $c_3 = 2$

[3] Ausgerichtete DvBs sind Datenwörter voller Breite, deren niederwertigstes Teilwort auch das niederwertigste Teilwort des DvBs im Speicher ist. Das Lesen eines DvBs, dessen Teilwörter im Speicher nebeneinander angeordnet sind, welche aber zu zwei DvBs gehören, ist nicht möglich.

add	$(o_{1(1)} = 2,\ l_{1(1)} = 2)$	*shiftLogicLeft*	$(o_{3(1)} = 1,\ l_{3(1)} = 1)$
add2x16	$(o_{1(2)} = 2,\ l_{1(2)} = 2)$	*shiftLogicRight*	$(o_{3(2)} = 1,\ l_{3(2)} = 1)$
add4x8	$(o_{1(3)} = 2,\ l_{1(3)} = 2)$	*shiftArithLeft*	$(o_{3(3)} = 2,\ l_{3(3)} = 2)$
mul	$(o_{1(4)} = 2,\ l_{1(4)} = 2)$	*shiftArithRight*	$(o_{3(4)} = 2,\ l_{3(4)} = 2)$

MUL: $c_2 = 8$

mul	$(o_{2(1)} = 4,\ l_{2(1)} = 5)$
mul2x16	$(o_{2(2)} = 4,\ l_{2(2)} = 5)$
mul4x8	$(o_{2(3)} = 4,\ l_{2(3)} = 5)$

Alle Zahlenangaben sind normiert und somit ohne Maßeinheit. Der Funktionseinheit ALU1 *aus Abbildung 2.2 gehört zur Funktionseinheit* ALU *und hat die Instanz* $n = 1$. *Die Funktionseinheit* ALU2 *gehört ebenfalls zur Funktionseinheit* ALU, *deren Instanz ist jedoch* $n = 2$.

Die Beschreibung der Eigenschaften einer Funktionseinheit m über ihre Menge der Befehle ist nicht ausreichend für die Untersuchung von Teilwortparallelität. Denn ein entscheidender Schritt bei der Verwendung von TWP ist die Zuordnung der Operationen eines Algorithmus zu den Operationen, welche sich hinter den TWP-Befehlen verbergen. Hierzu ist es notwendig zu wissen, welche Grundoperationen die TWP-Befehle besitzen und welches Datenformat die Ein- und Ausgabedaten haben müssen. Da die Eigenschaften der Operationen unabhängig von den Funktionseinheiten sind, welche diese Operationen realisieren, werden die Eigenschaften aller Befehle in einer Befehlsliste (`InstructionList`) zusammengefasst, wie in Algorithmus 2.2 gezeigt.

Ein Befehl (`Instruction`) hat einen Namen (`name`) und einen Maschinenkode (`code`). Name und Maschinenkode müssen jeweils eindeutig sein. Jeder Befehl muss von mindestends einer Funktionseinheit abgearbeitet werden können, deshalb muss der Name des Befehls in mindestens einer Funktionseinheit vorkommen. Da ein Befehl mehrere funktionelle Beschreibungen (`FunctionalDescription`) besitzen kann, werden diese nacheinander aufgeführt. Die funktionelle Beschreibung gibt zuerst das Ein- und Ausgabedatenformat (`InputData` und `OutputData`) an, bevor für jedes Teilwort des Ausgabedatenformats die Ausgabefunktion (`FunctionalDescription`) beschrieben wird.

Für das Ein- und Ausgabedatenformat wird die Anzahl der Ein- und Ausgabedatenwörter (`number`), die Anzahl der Teilwörter eines Datenworts (`subwords`) und der Typ jedes Teilworts angegeben. Der Typ eines Teilworts (`Subword`) beinhaltet die Informationen: vorzeichenbehaftet ja oder nein (`sign=<yes,no>`), ganzzahlig oder Fließkommazahl (`type=<int,float>`), Wortbreite des Teilworts in Bits (`width`) und Breite der Mantisse (`mantissa`) bei einer Fließkommazahl.

Beispiel 2.3
Verschiedene Ein- und Ausgabedatenformate sollen im Folgenden beschrieben werden:

- *2 Eingabedaten mit je einem Teilwort, die Teilwörter sind 64 bit Gleitkommazahlen mit 52 Bit breiter Mantisse (IEEE 754 Standard).*

Algorithmus 2.2 Beschreibung der Operationen eines Befehls

```
<InstructionList>
    <Instruction name="Name des Befehls" code="Maschinenkode">
        <FunctionalDescription>
            <InputData number="Anzahl">
                <Word subwords="Anzahl">
                    <Subword sign="<yes,no>" type="<int,float>"
                             width="Bits" mantissa="Bits" />
                </Word>
                ...
            </InputData>
            <OutputData number="Anzahl">
                <Word subwords="Anzahl">
                    <Subword sign="<yes,no>" type="<int,float>"
                             width="Bits" mantissa="Bits" />
                </Word>
                ...
            </OutputData>
            <OutputFunction>
                <Function word="Position(en)", subword="Position(en)">
                    Ausgabefunktion
                </Function>
                ...
            </OutputFunction>
        </FunctionalDescription>
        <FunctionalDescription>
            ...
        </FunctionalDescription>
        ...
    </Instruction>
    <Instruction>
        ...
    </Instruction>
    ...
</InstructionList>
```

```
<InputData number="2">
    <Word subwords="1">
        <Subword sign="yes" type="float" width="64" mantissa="52"/>
    </Word>
</InputData>
```

- *2 Eingabedaten mit je vier Teilwörtern, die Teilwörter sind vorzeichenbehaftete 16 Bit Ganzzahlen.*

```
<InputData number="2">
    <Word subwords="4">
        <Subword sign="yes" type="int" width="16"/>
    </Word>
</InputData>
```

- *1 Ausgabedatum mit drei Teilwörtern, das erste Teilwort ist eine vorzeichenbehaftete 16 Bit Ganzzahl, die beiden anderen Teilwörter sind vorzeichenbehaftete 8 Bit Ganzzahlen (Datentyp für QCPX-Befehlssatz [KW03]).*

```
<OutputData number="1">
    <Word subwords="3">
        <Subword sign="yes" type="int" width="16"/>
        <Subword sign="yes" type="int" width="8"/>
        <Subword sign="yes" type="int" width="8"/>
    </Word>
</OutputData>
```

Die Beschreibung der Ausgabefunktionen kann für jedes Teilwort einzeln (z. B. `subword=2`) oder für Teilwortbereiche (z. B. `subword=2-5`) erfolgen. Die Funktion wird als C-Code beschrieben. Die Eingabedaten werden mit `inX[sw]` gekennzeichnet, wobei `X` die Nummer des Eingabedatenworts ist und `sw` die Nummer des Teilworts. Der Zähler für die Datenwörter und Teilwörter startet mit „0“.

Beispiel 2.4
In Folgenden sollen einige Ausgabefunktionen vorgestellt werden.

- *Einfache Addition von zwei 32 Bit Ganzzahlen ohne TWP.*

```
<OutputFunction>
    <Function word="0", subword="0"> in0[i] + in1[i] </Function>
</OutputFunction>
```

- *Parallele Addition von vier 8 Bit Teilwörter.*

```
<OutputFunction>
    <Function word="0", subword="0-3"> in0[i] + in1[i] </Function>
</OutputFunction>
```

- *Packbefehl, bei dem die höherwertigen zwei Teilwörter des Eingabedatenworts auf die Position der niederwertigen zwei Teilwörter und der höherwertigen zwei Teilwörter des Ausgabedatenworts kopiert werden.*

$x_{1,1}$	$x_{1,2}$	...	$x_{1,n}$	$x_{2,1}$	$x_{2,2}$	...	$x_{2,n}$	$x_{3,1}$	...	$x_{m,1}$	$x_{m,2}$	...	$x_{m,n}$

Abbildung 2.5: Anordnung von Daten eines zweidimensionalen Datenraums im Speicher

```
<OutputFunction>
    <Function word="0" subword="0-1"> in0[i+2] </Function>
    <Function word="0" subword="2-3"> in0[i] </Function>
</OutputFunction>
```

- *Multiplikations- und Additions-Operation. Die vier Teilwörter des einen Eingabedatenworts werden paarweise mit den Teilwörtern des anderen Eingabedatenworts multipliziert. Die zwei niederwertigen und die zwei höherwertigen Teilwörter des Ergebnis-DvBs der Multiplikation werden miteinander addiert, sodass das Ausgabedatenwort nur noch zwei Teilwörter besitzt.*

```
<OutputFunction>
    <Function word="0" subword="0-1">
        in0[2*i]*in1[2*i]+in0[2*i+1]*in1[2*i+1]
    </Function>
</OutputFunction>
```

Die Beschreibung der Funktionseinheiten und der Befehle basiert auf der MAML-Beschreibung [KHK+06a, KHK+06b].

2.4 Speicherzugriff

Die Addressierung der Daten im Speicher ist linear organisiert. Unsere Algorithmenbeschreibung erlaubt aber die Beschreibung von mehrdimensionalen Ein- und Ausgabedaten. Ist der Datenraum mehrdimensional, so werden die Daten nebeneinander im Speicher angeordnet, wie Abbildung 2.5 für die Anordnung von zweidimensionalen Daten zeigt.

Aufgrund der linearen Adressierung der Daten im Speicher können die Daten in maximal einer Raumrichtung so angeordnet sein, dass die Teilwörter eines DvBs nebeneinander liegen. Für die Beschreibung der Raumrichtung, in der die Daten nebeneinander liegen, soll die Datenausrichtungsmatrix $\mathbf{\Gamma}^M$ definiert werden.

Definition 2.1 (Datenausrichtungsmatrix $\mathbf{\Gamma}^M$)
Die Datenausrichtungsmatrix $\mathbf{\Gamma}^M$ sei eine eindimensionale Matrix $\mathbf{\Gamma}^M \in \mathbb{N}^{1\times n}$, mit den Elementen $\gamma^M_{1,j} \in \{0,1\}$ und $j = 1,2,\ldots,n$. Die Größe der Matrix $1 \times n$ ist durch die Dimension n des Datenraums gegeben. Die Matrix $\mathbf{\Gamma}^M$ besitzt genau ein Nicht-Null-Element, d. h. $\exists!\, \gamma^M_{1,j} = 1$, $1 \leq j \leq n$. Das Element $\gamma^M_{1,j} = 1$ gibt die Raumrichtung j an, in der die Daten nebeneinander im Speicher angeordnet sind.

Je nachdem ob die Speicheradressierung bezüglich der Teilwörter oder der DvBs organisiert ist, muss beim DvB-Speicherzugriff zwischen *ausgerichteten* und *unausgerichteten*

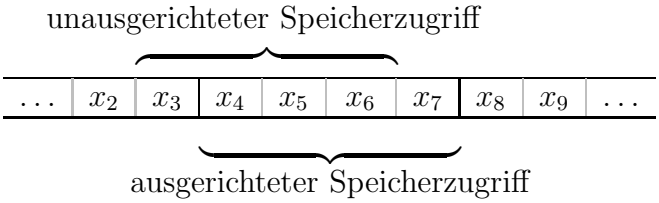


Abbildung 2.6: Ausgerichtete und unausgerichtete DvBs (4 Teilwörter) im Speicher

Speicherzugriff unterschieden werden. Der unausgerichtete Speicherzugriff erlaubt das Lesen bzw. Schreiben von B Teilwörtern, also eines DvBs, ausgehend von einem beliebigen Teilwort (Startadresse). Beim ausgerichteten Speicherzugriff ist hingegen nur das Lesen bzw. Schreiben **eines** kompletten DvBs aus dem Speicher möglich. Das erste Teilwort, von dem aus das DvB gelesen wird, ist also immer ein Vielfaches von B. Abbildung 2.6 illustriert die beiden Speicherzugriffsvarianten.

Für Allzweckprozessoren, wie Athlon64 und Intel Core2, stehen meist Lese- und Schreiboperationen zur Verfügung, die sowohl den ausgerichteten als auch den unausgerichteten Speicherzugriff erlauben. Die Abarbeitungsdauer für einen unausgerichtete Speicherzugriff dauert aber länger als für einen ausgerichteten Speicherzugriff [Adv05, Int07a].

Andere Architekturen, wie zum Beispiel der Cell Prozessor [Int07b], erlauben dagegen nur das ausgerichtete Lesen und Schreiben von DvBs. Für solche Architekturen müssen die unausgerichteten Speicherzugriffe in ausgerichtete Speicherzugriffe umgewandelt werden, wofür meist zusätzliche Packoperationen notwendig sind.

2.5 Zielarchitekturen für diese Arbeit

Teilwortparallelität hat mittlerweile in viele Prozessorarchitekturen Einzug gehalten. Aktuelle Mehrzweckprozessoren, wie Intel Core i7, AMD Phenom und PowerPC, bieten eine Vielzahl an TWP-Befehlen zur parallelen Verarbeitung von Daten mit niedriger Wortbreite. Auch wenn die aktuellen Prozessoren mehrere Prozessorkerne besitzen, können diese Prozessoren nicht als Rechenfelder betrachtet werden, denn die Parallelverarbeitung ist aufgabenbasiert und nicht datenbasiert. Somit werden die Mehrzweckprozessoren in unserer Arbeit als Rechenfelder mit einem Prozessorelement interpretiert.

Der CELL-Prozessor [EOO+05] von IBM besteht aus einem PowerPC Prozessorelement (kurz: PPE) und acht synergetischen Prozessorelementen (engl.: synergistic processing element, kurz: SPE). Alle PEs bieten Möglichkeiten der Teilwortparallelität. Der CELL-Prozessor ist aber in unserem Sinne ebenfalls kein Rechenfeld, denn der PPE steuert die SPEs und die Parallelverarbeitung ist aufgaben- und nich datenbasiert.

In Digitalen Signalprozessoren (kurz: DSPs), wie TigerSHARC [Fri00], NXP Trimedia [VEP99] und Texas Instruments TMS320C64x [Ins01] wurden ebenfalls TWP-Befehle integriert, um den Datendurchsatz zu erhöhen. Der TI TMS320C64x mit seinen zwei Datenpfaden kann als ein lineares Rechenfeld mit zwei PEs betrachtet werden. Die anderen beiden DSPs werden als Rechenfelder mit einem PE angesehen.

Der ARM Cortex-M3-Mikroprozessor [Sad08] bietet mit dem Befehlssatz *Thumb-2* ebenfalls gute Möglichkeiten der teilwortparallelen Verarbeitung von Daten.

Ein aktueller Trend ist die Entwicklung von Architekturen für *Software Defined Radio* (kurz: SDR). Software Defined Radio bezeichnet die Softwarerealisierung der Signalverarbeitung von Hochfequenzsendern und -empfängern in anwendungsspezifischen Prozessoren (engl.: application-specific instruction-set processor. kurz: ASIP) oder Mehrzweckprozessoren. Die Sandblaster Architektur von Sandbridge [GMI+05] und die in [NBB+08] beschriebene Architektur für die Turbo- und Low-Density-Parity-Check-Dekodierung (kurz: LDPC-Dekodierung) sind zwei Beispiele für SDR ASIPs. Diese Architekturen bieten in ihrem anwendungsspezifischen Befehlssatz teilwortparallele Befehle, um die benötigte Verarbeitungsgeschwindigkeit bei gleichzeitig niedrigen Energieverbrauch zu erreichen.

Weitere Rechenfelder mit mehreren Prozessorelementen und mit Teilwortparallelität sind der M5-DSP [HHF03][HHF04], der HiveFlex Moustique-IC2 Processor von Silicon Hive [Sil] und das schwach programmierbare Rechenfeld (engl.: weackly programmable processor array, kurz: SPRF) [KHKT06]. Alle drei Prozessoren basieren auf skalierbaren Architekturkonzepten. Die ersten beiden Architekturkonzepte erlauben den Entwurf von linearen Rechenfeldern, welche nach dem SIMD Prinzip arbeiten. Das Konzept des SPRF zielt auf den Entwurf mehrdimensionaler Rechenfelder, i. A. zweidimensional, ab.

Nachfolgend werden die Mehrzweckprozessoren von Intel und AMD sowie das schwach programmierbare Rechenfeld etwas ausführlicher vorgestellt, denn diese Architekturen sind die hauptsächlichen Zielarchitekturen in dieser Arbeit.

2.5.1 Mehrzweckprozessoren von Intel und AMD

Mehrzweckprozessoren (engl.: General Purpose Processors) wie die der Intel Pentium, Intel Core, AMD Athlon und AMD Phenom besitzen seit Jahren Befehlssätze für die Teilwortparallelität. Die ersten TWP-Befehle wurden im Form des MMX-Befehlssatzes [PW96] im Jahr 1997 eingeführt. Mit SSE [Int04], SSE2 [Int04, AMD02], SSE3 [Int04], SSE4 [Ram06] (Intel), 3DNow! [AMD02] und SSE5 [Adv07] (AMD) wurden weitere Befehlssätze für Mehrzweckprozessoren entwickelt und realisiert. Diese Entwicklung setzt sich weiter fort.

Auch wenn aktuelle Mehrzweckprozessoren mehrere Kerne besitzen, betrachten wir diese Prozessoren als Rechenfelder mit einem Prozessorelement, denn die Kerne dieser Architekturen sind nur lose miteinander verbunden.

Bei den oben genannten Prozessoren muss zwischen Befehlen für 64 Bit DvBs und Befehlen für 128 Bit DvBs unterschieden werden. Für die Berechnungen der 64 Bit DvBs stehen acht Register und für die Berechnungen der 128 Bit DvBs stehen 16 Register zur Verfügung. Die acht 64 Bit Register werden ebenfalls für die Speicherung von Fließkommazahlen benutzt, daher ist es nach der Verwendung der MMX/3DNow!-Befehle notwendig, mittels des Befehls `emms` diese Register wieder freizugeben. Der Zeitbedarf für den Befehl `emms` ist sehr hoch, sodass nicht zu oft zwischen Fließkommazahloperationen und teilwortparallelen Operationen umgeschaltet werden sollte.

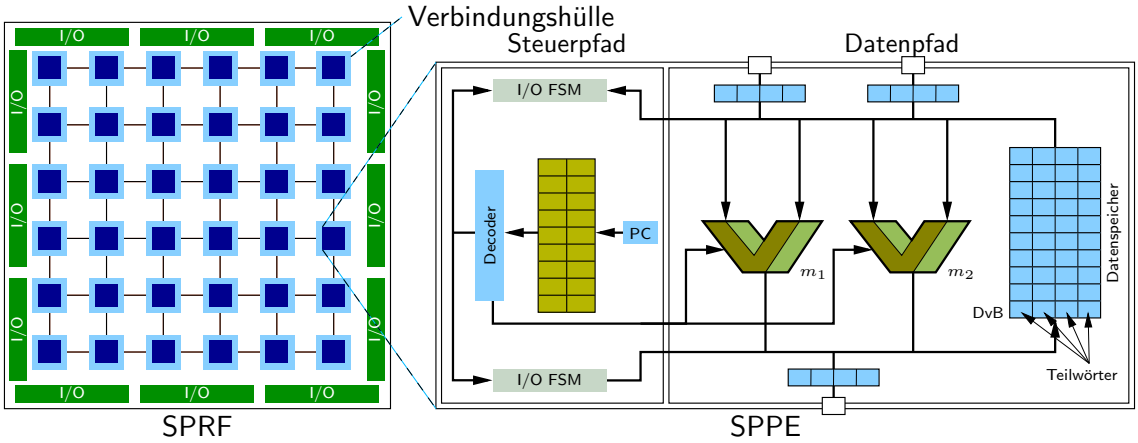


Abbildung 2.7: Schwach programmierbares Rechenfeld (SPRF)

Tabelle 2.1: Eigenschaften der Funktionseinheiten (Eingabeverzögerung $o_{m(s)} = 1$)

FE	Fläche c_m	Funktionen F_s mit Latenz $l_{m(s)}$ in der Klammer
m_1	4	abs(2), min(3), shr(1), add/sub(1), pack1(1), pack2(1)
m_2	2	shr(1), pack1(1), pack2(1)

Eine Speicherarchitektur, an der das SPRF angebunden ist, wurde noch nicht definiert.

Die oben genannten Befehlssätze erlauben es, praktisch alle Datenformate ob Ganzzahl (8 Bit, 16 Bit, 32 Bit, 64 Bit) oder Fließkommazahl (32 Bit, 64 Bit) teilwortparallel zu verarbeiten. Entsprechende Befehle zur Berechnung stehen zur Verfügung. Die Möglichkeiten der Veränderung der Positionen der Teilwörter innerhalb der DvBs sind hingegen sehr beschränkt.

2.5.2 Schwach programmierbares Rechenfeld

Eine Referenzarchitektur für ein Rechenfeld mit Teilwortparallelität ist das schwach programmierbare Rechenfeld (kurz: SPRF) [KHKT06]. Das SPRF besteht aus einem Feld von schwach programmierbaren Prozessorelementen (kurz: SPPEs). Ein SPPE besitzt ein oder mehrere Funktionseinheiten (FE) mit einem begrenzten Befehlssatz. Die FEs erlauben die Verarbeitung von Daten mit mehreren Teilwörtern (Teilwortparallelität). Die Prozessorelemente sind durch programmierbare Kanäle miteinander verbunden.

Abbildung 2.7 zeigt beispielhaft ein solches SPRF. Das Rechenfeld hat die Größe $\boldsymbol{\vartheta}^{\square} = \binom{6}{6}$. Die SPPE sind durch die Kanäle $\mathbf{q}_1 = \binom{1}{0} \in \mathcal{Q}^{\text{bi}}$, $N_{\mathbf{q}_1} = 1$ und $\mathbf{q}_2 = \binom{0}{1} \in \mathcal{Q}^{\text{bi}}$, $N_{\mathbf{q}_2} = 1$ miteinander verbunden. Jedes SPPE besitzt zwei Funktionseinheiten $m_1, m_2 \in \mathcal{M}$. Die Funktionseinheiten mit 64 Bit Wortbreite erlauben Teilwortparallelität mit vier Teilwörtern. Das heißt, vier Teilwörter mit 16 Bit Wortbreite werden parallel in den Funktionseinheiten berechnet. Tabelle 2.1 zeigt die Funktionen F_s, welche die Funktionseinheiten m_i, $i = 1, 2$ ausführen können. Der lokale Speicher des SPPE hat die Größe $M_R = 10$.

Kapitel 3

Algorithmenklasse

In diesem Kapitel wird die Algorithmenklasse vorgestellt, welche Ausgangspunkt für unsere Algorithmentransformationen und den Abbildungsprozess auf ein paralleles Rechenfeld ist. Die Algorithmen dieser Klasse bezeichnen wir als uniform-iterative Algorithmen (UItA). Sie sind wie folgt definiert.

Definition 3.1 (Uniform-iterativer Algorithmus (UItA))
Ein uniform-iterativer Algorithmus ist eine Menge von Zuweisungen $s \in \mathcal{S}$ der Form

$$s: \quad y_s[l_s(\mathbf{i})] = \begin{cases} F_{s_1}(\ldots), & \mathbf{i} \in \mathcal{I}_{s_1}, \\ \ldots & \\ F_{s_t}\left(\ldots, y_{s'}[r^{k_{s'}}_{s's_t}(\mathbf{i})], \ldots, y_r[r^{k_r}_{r\,s_t}(\mathbf{i})], \ldots\right), & \mathbf{i} \in \mathcal{I}_{s_t}, \\ \ldots & \\ F_{s_{T_s}}(\ldots\ldots), & \mathbf{i} \in \mathcal{I}_{s_{T_s}}, \end{cases}$$
$$s, s' \in \mathcal{S}, \quad r \in \mathcal{S}_r, \quad 1 \le k_{s'} \le K_{s's_t}, \quad 1 \le k_r \le K_{r\,s_t}. \tag{3.1}$$

Jede Zuweisung s besteht aus T_s Teilzuweisungen. Jede Teilzuweisung s_t, $1 \le t \le T_s$ besitzt eine einwertige Funktion F_{s_t}.

Die indizierte Variable $y_s[.]$ hat die Indexfunktion

$$l_s(\mathbf{i}) = \mathbf{L}_s\mathbf{i} + \mathbf{l}_s \in \mathcal{L}_s \subset Z^{n_s} \quad \textit{oder} \quad r^{k_{s'}}_{s's_t}(\mathbf{i}) = \mathbf{R}^{k_{s'}}_{s's_t}\mathbf{i} + \mathbf{r}^{k_{s'}}_{s's_t} \in \mathcal{R}^{k_{s'}}_{s's_t} \subset Z^{n_{s's_t}} \tag{3.2}$$

und die indizierte Variable $y_r[.]$ hat die Indexfunktion

$$r^{k_r}_{r\,s_t}(\mathbf{i}) = \mathbf{R}^{k_r}_{r\,s_t}\mathbf{i} + \mathbf{r}^{k_r}_{r\,s_t} \in \mathcal{R}^{k_r}_{r\,s_t} \subset Z^{n_{r\,s_t}}. \tag{3.3}$$

Der Vektor $\mathbf{i} \in \mathbb{Z}^n$ der Indexfunktionen wird Iterationsvektor genannt. Die Indexräume $\mathcal{L}_s$, $\mathcal{R}^{k_{s'}}_{s's_t}$ und $\mathcal{R}^{k_r}_{r\,s_t}$ der indizierten Variable $y_s[.]$ oder $y_r[.]$ sind konvexe Polyeder.

Die Indexfunktionen $l_s(\mathbf{i})$ mit der Definitionsmenge $\mathcal{I}_s$ und die Indexfunktionen $r^{k_{s'}}_{s's_t}(\mathbf{i})$ und $r^{k_r}_{r\,s_t}(\mathbf{i})$ mit der Definitionsmenge $\mathcal{I}_{s_t}$ müssen bijektive Abbildungen sein. Die Definitionsmengen $\mathcal{I}_s$ und $\mathcal{I}_{s_t}$ werden im Folgenden als Iterationsräume bezeichnet.

Für alle Indexfunktionen $l_s(\mathbf{i})$ *und* $r^{k_s}_{s\,s'_t}(\mathbf{i})$ *der Variablen* $y_s[.] \in \mathcal{Y}_{L\cap R} = \mathcal{Y}_L \cap \mathcal{Y}_R$ *mit* $\mathcal{L}_s \cap \mathcal{R}^{k_s}_{s\,s'_t} \neq \emptyset$*, welche sowohl auf der linken Seite einer Zuweisung* $s \in \mathcal{S}$ *(*$y_s[.] \in \mathcal{Y}_L$*) als auch auf der rechten Seite einer Zuweisung* $s' \in \mathcal{S}$ *(*$y_s[.] \in \mathcal{Y}_R$*) des Algorithmus vorhanden sind und dessen Indexräume* $\mathcal{L}_s$ *und* $\mathcal{R}^{k_s}_{s\,s'_t}$ *nicht disjunkt sind (*$\mathcal{L}_s \cap \mathcal{R}^{k_s}_{s\,s'_t} \neq \emptyset$*), gelte:* $\mathbf{L}_s = \mathbf{R}^{k_s}_{s\,s'_t} = \mathbf{B}$ *und* $\mathbf{B}$ *sei invertierbar, denn diese Bedingung stellt sicher, dass die Indexfunktionen* $l_s(\mathbf{i})$ *und* $r^{k_s}_{s\,s'_t}(\mathbf{i})$ *bijektive Abbildungen sind.*

Bemerkung: Ist $l_s(\mathbf{i})$ *mit* $\mathbf{i} \in \mathcal{I}_s$ *bijektiv, so wird jede Variable nur einmal geschrieben. Ist* $r^{k_r}_{r\,s_t}(\mathbf{i})$ *mit* $\mathbf{i} \in \mathcal{I}_{s_t}$ *bijektiv, so wird jede Variable nur einmal gelesen.*

Jede Teilzuweisung s_t *ist in einem Iterationsraum* $\mathcal{I}_{s_t}$ *definiert, welcher ein konvexes Polyeder* $\mathcal{H}$ *darstellt. Die Iterationsräume* $\mathcal{I}_{s_t}$ *sind paarweise disjunkt (*$\forall t_1, t_2 : \mathcal{I}_{s_{t_1}} \cap \mathcal{I}_{s_{t_2}} = \emptyset,\ 1 \leq t_1, t_2 \leq T_s \wedge t_1 \neq t_2$*). Der Iterationsraum* $\mathcal{I}_s = \bigcup_{t=1}^{T_s} \mathcal{I}_{s_t}$ *der Zuweisung* s *ergibt sich aus der Vereinigung der Iterationsräume* $\mathcal{I}_{s_t}$ *der Teilzuweisungen* s_t *dieser Zuweisung. Der Iterationsraum* $\mathcal{I}_s$ *muss ebenfalls ein konvexes Polyeder* $\mathcal{H}$ *sein.*[1]

Der gemeinsame Iterationsraum $\mathcal{I}$ *eines UItA ergibt sich aus* $\mathcal{I} = conv\,(\cup_{s\in\mathcal{S}}\,\mathcal{I}_s)$*. Der Operator 'conv' bestimmt die konvexe Hülle der Vereinigung der Iterationsräume* $\mathcal{I}_s$*. Somit ist der gemeinsame Iterationsraum* $\mathcal{I}$ *auch wieder ein konvexes Polyeder* $\mathcal{H}$*.*

Für einen UItA wird angenommen,

(1) dass er eine *Einmal-Zuweisungs-Form* (engl. Single Assignment Form) hat. Das heißt, jede Instanz der Variable $y_s[.]$ wird nur einmal im Algorithmus definiert.[2]

(2) dass es eine partielle Ordnung der Instanzen der Zuweisungen $(s, \mathbf{i})$ mit $s \in \mathcal{S}$ und $\mathbf{i} \in \mathcal{I}_s$ gibt, die den Datenabhängigkeiten genügt.

Die Definition der uniform-iterativen Algorithmen basiert auf den *regulär iterativen Algorithmen* (engl.: regular iterative algorithm, kurz: RIA) [Rao85] sowie den *stückweise regulären Algorithmen* (engl.: piecewise regular algorithm, kurz: PRA) [Thi92]. Die *stückweise linearen Algorithmen* (engl.: piecewise linear algorithm, kurz: PLA) [TT93][Thi89] sind eine Verallgemeinerung der stückweise regulären Algorithmen. Die Indexmatrizen $\mathbf{L}_s$ und $\mathbf{R}^{k_{s'}}_{s'\,s_t}$ der stückweise linearen Algorithmen können frei gewählt werden, während für diese bei den stückweise regulären Algorithmen $\mathbf{L}_s = \mathbf{R}^{k_{s'}}_{s'\,s_t} = \mathbf{E}$ gilt. Durch Algorithmentransformationen [Thi89][RTRK89][DCDM00] lassen sich PLAs meist in PRAs und damit in UItA umwandeln.

Algorithmen aus der allgemeineren Klasse der *affin-indizierten Algorithmen* (engl.: affine indexed algorithm, kurz: AIA) [Eck01] lassen sich ebenfalls mittels bekannter Lokalisierungsmethoden [EM99][RCDM94][RTRK89] in uniform-indizierte Algorithmen transformieren.

[1] Die Vereinigung von konvexen Polyedern ergibt nicht zwangsläufig wieder ein konvexes Polyeder.

[2] In der Fachliteratur wird die bei den UItAs verwendete Einmal-Zuweisungs-Form als *dynamische Einmal-Zuweisungs-Form* [VJB05][Fea88] bezeichnet.

Programm 3.1 FIR-Filter in C-Syntax

```
for( unsigned i = 0; i < 1024; i++ ){
    y[i] = 0;
    for( unsigned j = 0; j < 20; j++ ){
        y[i] += a[j]*x[i-j];
    }
}
```

Algorithmus 3.1 FIR-Filter

$$s_1: \quad y_a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} a[j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{1_1} \\ y_a\left[\begin{smallmatrix} i-1 \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{1_2} \end{cases}$$

$$s_2: \quad y_x\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} x[i-j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{2_1} \\ x[i-j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{2_2} \\ y_x\left[\begin{smallmatrix} i-1 \\ j-1 \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{2_3} \end{cases}$$

$$s_3: \quad y_y\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} y_a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \cdot y_x\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{3_1} \\ y_y\left[\begin{smallmatrix} i \\ j-1 \end{smallmatrix}\right] + y_a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \cdot y_x\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{3_2} \end{cases}$$

$$s_4: \quad y[i] = y_y\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], \qquad \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{4_1}$$

mit $\mathcal{I}_{1_1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = 0 \wedge 0 \leq j \leq J-1\}$,
$\mathcal{I}_{1_2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \leq i \leq I-1 \wedge 0 \leq j \leq J-1\}$,
$\mathcal{I}_{2_1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = 0 \wedge 0 \leq j \leq J-1\}$,
$\mathcal{I}_{2_2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \leq i \leq I-1 \wedge j = 0\}$,
$\mathcal{I}_{2_3} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \leq i \leq I-1 \wedge 1 \leq j \leq J-1\}$,
$\mathcal{I}_{3_1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \leq i \leq I-1 \wedge j = 0\}$,
$\mathcal{I}_{3_2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \leq i \leq I-1 \wedge 1 \leq j \leq J-1\}$,
$\mathcal{I}_{4_1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \leq i \leq I-1 \wedge j = J-1\}$,
$I = 1024$, $J = 20$.

Beispiel 3.1 (Finite Impulse Response Filter)
Das Finite Impulse Response (kurz: FIR) Filter aus Programm 3.1 ist in Algorithmus 3.1 als uniform iterativer Algorithmus dargestellt.

Hierbei wurden die Indexfunktionen in Kurzform formuliert. Beispielhaft sind nachfolgend alle Indexfunktionen der Zuweisung s_2 und s_4 ausführlich präsentiert:

$$s_2: \quad l_{s_2}(\mathbf{i}) = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \cdot \mathbf{i} + \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right), \qquad r^1_{r_2 s_{2_1}}(\mathbf{i}) = \begin{pmatrix} 1 & -1 \end{pmatrix} \cdot \mathbf{i} + \begin{pmatrix} 0 \end{pmatrix},$$
$$r^1_{r_2 s_{2_2}}(\mathbf{i}) = \begin{pmatrix} 1 & -1 \end{pmatrix} \cdot \mathbf{i} + \begin{pmatrix} 0 \end{pmatrix},$$
$$r^1_{s_2 s_{2_3}}(\mathbf{i}) = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \cdot \mathbf{i} - \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right),$$

$$s_4: \quad l_{s_4}(\mathbf{i}) = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \mathbf{i} + \begin{pmatrix} 0 \end{pmatrix}, \qquad r^1_{s_3 s_{4_1}}(\mathbf{i}) = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \cdot \mathbf{i} + \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right).$$

3.1 Räume

Die Iterationsräume des UItA sind konvexe Polyeder. Konvexe Polyeder sind wie folgt definiert:

Definition 3.2 (Konvexes Polyeder)
Ein konvexes Polyeder $\mathcal{H}$, meist nur Polyeder bezeichnet, ist eine Punktmenge, die sich durch ein lineares Ungleichungssystem darstellen lässt

$$\mathcal{H} = \{\mathbf{i} \,|\, \mathbf{i} \in \mathbb{Z}^n \wedge \mathbf{A} \cdot \mathbf{i} \geq \mathbf{a}_0\}, \qquad \mathbf{A} \in \mathbb{Q}^{m \times n}, \quad \mathbf{a}_0 \in \mathbb{Q}^m. \tag{3.4}$$

Ein Polyeder wird durch Hyperebenen $\mathbf{a}_i \cdot \mathbf{i} \geq a_{0,i}$ begrenzt, die durch die Ungleichungen des Ungleichungssystems (3.4) beschrieben werden. Neben der impliziten Definition *entsprechend (3.4) ist eine parametrische Darstellung, auch Minkowski Charakterisierung [Wil93] genannt, möglich:*

$$\mathcal{H} = \left\{\mathbf{i} \,|\, \mathbf{i} = \mathbf{L}\boldsymbol{\lambda} + \mathbf{R}\boldsymbol{\mu} + \mathbf{V}\boldsymbol{\nu},\ \boldsymbol{\lambda} \in \mathbb{Q}^{n_\lambda},\ \mathbf{0} \leq \boldsymbol{\mu} \in \mathbb{Q}^{n_\mu},\ \mathbf{0} \leq \boldsymbol{\nu} \in \mathbb{Q}^{n_\nu},\ \mathbf{1}^t \cdot \boldsymbol{\nu} = 1\right\}. \tag{3.5}$$

Dabei sind die Spalten der Matrix $\mathbf{L} \in \mathbb{Q}^{n \times n_\lambda}$ die n_λ Linien, die Spalten der Matrix $\mathbf{R} \in \mathbb{Q}^{n \times n_\mu}$ die n_μ Strahlen und die Spalten der Matrix $\mathbf{V} \in \mathbb{Q}^{n \times n_\nu}$ die n_ν Ecken des Polyeders.

Die Menge $\mathcal{V} = \{\mathbf{v} | \mathbf{v} = \mathbf{V} \cdot \mathbf{m},\ m_j \in \{0, 1\},\ \sum_{j=1}^{n} m_j = 1\}$ enthält alle Ecken $\mathbf{v}$ des Polyeders.

Ein Polyeder, das beschränkt ist, also von allen Seiten durch Hyperebenen begrenzt ist, heißt Polytop. Ein Polytop besitzt keine Linien und keine Strahlen. Die Iterationsräume $\mathcal{I}_{s_t}$ der Teilzuweisungen s_t als auch die Iterationsräume $\mathcal{I}_s$ der Zuweisungen $s \in \mathcal{S}$ sind Polytope. Anderenfalls würden die den Iterationsräumen $\mathcal{I}_{s_t}$ und $\mathcal{I}_s$ zugrundeliegenden Teilzuweisungen s_t bzw. Zuweisungen $s \in \mathcal{S}$ unendlich viele Instanzen besitzen.

Jedes beschränkte Polyeder $\mathcal{H}$ kann von einen Hyperquader $\mathcal{H}^\square$ eingeschlossen werden. Ein Hyperquader ist wie folgt definiert:

Definition 3.3 (Hyperquader)
Ein Hyperquader $\mathcal{H}^\square$ ist ein konvexes Polyeder $\mathcal{H} = \{\mathbf{i} \,|\, \mathbf{A}\mathbf{i} \geq \mathbf{a}_0,\ \mathbf{A} \in \mathbb{Q}^{m \times n}, \mathbf{a}_0 = \mathbb{Q}^m\}$ für das gilt:

$$\mathbf{A} = \begin{pmatrix} \mathbf{E} \\ -\mathbf{E} \end{pmatrix} \qquad \textit{und} \qquad \mathbf{a}_0 = \begin{pmatrix} \mathbf{a}_0^{\sqcup} \\ -\mathbf{a}_0^{\sqcap} \end{pmatrix}, \qquad \mathbf{a}_0^{\sqcup}, \mathbf{a}_0^{\sqcap} \in \mathbb{Z}^n. \tag{3.6}$$

Die Vektoren $\mathbf{a}_0^{\sqcap}$ und $\mathbf{a}_0^{\sqcup}$ definieren die obere bzw. untere Begrenzung des Hyperquaders in jeder Raumrichtung.

Die Ecken $\mathbf{v} \in \mathcal{V}$ des Polyeders $\mathcal{H}$ können rationale Punkte im Raum sein. Da das Polyeder $\mathcal{H}$ aber eine Punktmenge im $\mathbb{Z}^n$ ist, besitzen die oberen und unteren Grenzen $a_{0,j}^{\sqcap}$ und $a_{0,j}^{\sqcup}$ des umschließenden Hyperquaders $\mathcal{H}^\square$ ganzzahlige Werte (siehe Definition 3.3).

Die Grenzen des umschließenden Hyperquaders $\mathcal{H}^{\square}$ lassen sich aus den Ecken $\mathbf{v} \in \mathcal{V}$ des Polyeder $\mathcal{H}$ bestimmen:

$$a_{0,j}^{\sqcap} = \left\lfloor \max_{\forall \mathbf{v} \in \mathcal{V}} v_j \right\rfloor \qquad \text{und} \qquad a_{0,j}^{\sqcup} = \left\lceil \min_{\forall \mathbf{v} \in \mathcal{V}} v_j \right\rceil. \tag{3.7}$$

Die auf- bzw. abgerundeten Minimal- bzw. Maximalwerte der Eckelemente v_j sind die oberen $a_{0,j}^{\sqcap}$ bzw. unteren Grenzen $a_{0,j}^{\sqcup}$ des Hyperquaders $\mathcal{H}^{\square}$, wie die Gleichungen 3.7 zeigen.

Das Volumen des Hyperquaders $V(\mathcal{H}^{\square})$[3] kann somit mit

$$V(\mathcal{H}^{\square}) = \prod_{j=1}^{n} (a_{0,j}^{\sqcap} - a_{0,j}^{\sqcup} + 1) \tag{3.8}$$

bestimmt werden.

Im zweidimensionalen Raum wird der Hyperquader als *Rechteck* und im dreidimensionalen Raum als *Quarder* bezeichnet. In der weiteren Arbeit benutzen wir immer die allgemeine Bezeichnung *Hyperquader*.

Beispiel 3.2 (Hyperquader)
Es sei ein Polyeder $\mathcal{H}$ der Größe

$$\mathcal{H} = \left\{ \mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 1 & -2 \end{pmatrix} \mathbf{i} \geq \begin{pmatrix} 1 \\ 0 \\ -5 \\ -2 \end{pmatrix} \right\} \tag{3.9}$$

gegeben. Das Polyeder hat vier Ecken $\mathbf{v} \in \mathcal{V}_P = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 3/2 \end{pmatrix}, \begin{pmatrix} 5 \\ 7/2 \end{pmatrix} \right\}$.

Der Hyperquader $\mathcal{H}^{\square}$ soll das Polyeder $\mathcal{H}$ umschließen. Für die obere und untere Begrenzung des Hyperquaders erhalten wir aus den Gleichungen 3.7: $\mathbf{a}_0^{\sqcap} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$ und $\mathbf{a}_0^{\sqcup} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Somit lautet die implizite Definition für den Hyperquader

$$\mathcal{H}^{\square} = \left\{ \mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \mathbf{i} \geq \begin{pmatrix} 1 \\ 0 \\ -5 \\ -3 \end{pmatrix} \right\}. \tag{3.10}$$

Der Hyperquader $\mathcal{H}^{\square}$ hat vier Ecken: $\mathbf{v} \in \mathcal{V}_H = \{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \begin{pmatrix} 5 \\ 3 \end{pmatrix} \}$.

Man beachte, die rationale Ecke $\mathbf{v} = \begin{pmatrix} 5 \\ 7/2 \end{pmatrix} \in \mathcal{V}_P$ des Polyeders $\mathcal{H}$ liegt außerhalb des Hyperquaders $\mathcal{H}^{\square}$. Alle ganzzahligen Iterationspunkte $\mathbf{i} \in \mathcal{H} \subset \mathbb{Z}^n$ des Polyeders $\mathcal{H}$ liegen aber auch innerhalb des Hyperquaders $\mathcal{H}^{\square}$ ($\mathcal{H} \backslash \mathcal{H}^{\square} = \emptyset$). Abbildung 3.1 zeigt das Polyeder $\mathcal{H}$ und den Hyperquader $\mathcal{H}^{\square}$ im zweidimensionalen Raum.

Das Volumen des Hyperquaders lautet: $V(\mathcal{H}^{\square}) = (5 - 1 + 1) \cdot (3 - 0 + 1) = 20$.

[3] Im zweidimesionalen Raum ist es die Fläche des Rechtecks.

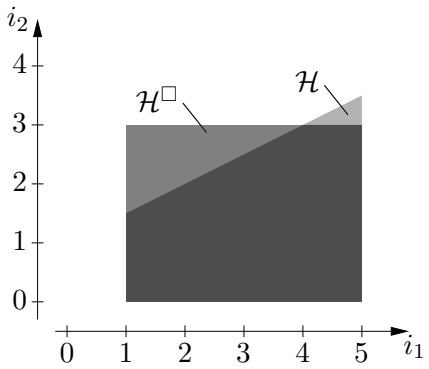


Abbildung 3.1: Polyeder mit umschließenden Hyperquader

Beispiel 3.3 (Iterationsräume des FIR-Filters)
Die Beschreibung der Iterationsräume $\mathcal{I}_{1_1}$ und $\mathcal{I}_{1_2}$ des FIR-Filters aus Beispiel 3.1 in der implizierten Definition lautet:

$$\mathcal{I}_{1_1} = \left\{ \mathbf{i} = \binom{i}{j} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -J+1 \end{pmatrix} \right\}, \tag{3.11}$$

$$\mathcal{I}_{1_2} = \left\{ \mathbf{i} = \binom{i}{j} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 1 \\ 0 \\ -I+1 \\ -J+1 \end{pmatrix} \right\}. \tag{3.12}$$

Die Ecken der Iterationsräume $\mathcal{I}_{1_1}$ und $\mathcal{I}_{1_2}$ sind in den Mengen $\mathcal{V}_{1_1}$ und $\mathcal{V}_{1_2}$ zusammengefasst:

$$\mathcal{V}_{1_1} = \left\{ \binom{0}{0}, \binom{0}{J-1} \right\}, \qquad \mathcal{V}_{1_2} = \left\{ \binom{1}{0}, \binom{1}{J-1}, \binom{I-1}{0}, \binom{I-1}{J-1} \right\}. \tag{3.13}$$

Der Iterationsraum $\mathcal{I}_1 = \mathcal{I}_{1_1} \cup \mathcal{I}_{1_2}$ der Zuweisung s_1 lautet:

$$\mathcal{I}_1 = \left\{ \mathbf{i} = \binom{i}{j} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 0 \\ 0 \\ -I+1 \\ -J+1 \end{pmatrix} \right\}. \tag{3.14}$$

Der gemeinsame Iterationsraum $\mathcal{I} = \operatorname{conv} \bigcup_{\forall s \in \mathcal{S}} \mathcal{I}_s$ des Algorithmus ist gleich dem Iterationsraum $\mathcal{I}_1$ der Zuweisung s_1. Der den gemeinsamen Iterationsraum $\mathcal{I}$ umschließende Hyperquader $\mathcal{I}^{\square}$ hat die gleiche Größe wie der gemeinsamen Iterationsraum $\mathcal{I}$, denn der gemeinsame Iterationsraum $\mathcal{I}$ ist ein Hyperquader.

3.2 Datenabhängigkeiten

Im Allgemeinen ist eine Teilzuweisung s_t der Zuweisung $s \in \mathcal{S}$ von einer Zuweisung $s' \in \mathcal{S}$ abhängig, wenn für die Variable $y_{s'}[.]$ gilt:

$$r_{s's_t}^{k_{s'}}(\mathbf{i}) = l_{s'}(\mathbf{i}') \tag{3.15}$$

mit $\mathbf{i} \in \mathcal{I}_{s_t}$ und $\mathbf{i}' \in \mathcal{I}_{s'}$. Diese Abhängigkeit wird durch den Abhängigkeitsvektor $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i})$ mit $\mathbf{i}' + \mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i}) = \mathbf{i}$, $\forall(\mathbf{i}', \mathbf{i}) \in \mathcal{G}_{s's_t}^{k_{s'}}$ beschrieben, wobei $\mathcal{G}_{s's_t}^{k_{s'}}$ das Abhängigkeitsgebiet des Abhängigkeitsvektors $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i})$ ist. Das Abhängigkeitsgebiet $\mathcal{G}_{s's_t}^{k_{s'}}$ kann wie folgt bestimmt werden:

$$\mathcal{G}_{s's_t}^{k_{s'}} = \left\{ (\mathbf{i}', \mathbf{i}) \,|\, \mathbf{i}' \in \mathcal{I}_{s'} \wedge \mathbf{i} \in \mathcal{I}_{s_t} \wedge l_{s'}(\mathbf{i}') = r_{s's_t}^{k_{s'}}(\mathbf{i}) \right\}. \tag{3.16}$$

Hilfssatz 3.1 (Uniforme Datenabhängigkeit)
Gilt für die Matrizen der Indexfunktionen $l_{s'}(\mathbf{i})$ und $r_{s's_t}^{k_{s'}}(\mathbf{i})$

$$\mathbf{L}_{s'} = \mathbf{R}_{s's_t}^{k_{s'}} = \mathbf{B} \in \mathbb{Z}^{n \times n} \qquad \textit{und} \qquad \mathrm{Rg}(\mathbf{B}) = n,$$

dann ist der Abhängigkeitsvektor $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i})$ für alle $(\mathbf{i}, \mathbf{i}') \in \mathcal{G}_{s's}^{k_{s'}}$ uniform. Das heißt, der Abhängigkeitsvektor $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i})$ ist unabhängig von den Iteration $\mathbf{i}$ und $\mathbf{i}'$. Der Abhängigkeitsvektor $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i})$ nimmt somit den konstanten Wert $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i}) = \mathbf{d}_{s's_t}^{k_{s'}}$ an.

Beweis 3.1
Zwischen der Zuweisung s' und der Teilzuweisung s_t der Zuweisung s besteht eine Datenabhängigkeit $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i}) = \mathbf{i} - \mathbf{i}'$, wenn für die Indexfunktionen $l_{s'}(\mathbf{i}')$ und $r_{s's_t}^{k_{s'}}(\mathbf{i})$ der Variablen $y_{s'}[.]$ gilt:

$$l_{s'}(\mathbf{i}') = r_{s's_t}^{k_{s'}}(\mathbf{i}) \tag{3.17}$$

$$\mathbf{L}_{s'}\mathbf{i}' + \mathbf{l}_{s'} = \mathbf{R}_{s's_t}^{k_{s'}}\mathbf{i} + \mathbf{r}_{s's_t}^{k_{s'}}. \tag{3.18}$$

Ist $\mathbf{L}_{s'}$ eine reguläre Matrix ($\mathrm{Rg}(\mathbf{L}_{s'}) = n$, $\mathbf{L}_{s'} \in \mathbb{Z}^{n \times n}$), so gilt:

$$\mathbf{i}' = \mathbf{L}_{s'}^{-1}\mathbf{R}_{s's_t}^{k_{s'}}\mathbf{i} + \mathbf{L}_{s'}^{-1}(\mathbf{r}_{s's_t}^{k_{s'}} - \mathbf{l}_{s'}). \tag{3.19}$$

Setzt man (3.19) in $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i}) = \mathbf{i} - \mathbf{i}'$ ein, folgt:

$$\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i}) = (\mathbf{E} - \mathbf{L}_{s'}^{-1}\mathbf{R}_{s's_t}^{k_{s'}})\mathbf{i} + \mathbf{L}_{s'}^{-1}(\mathbf{l}_{s'} - \mathbf{r}_{s's_t}^{k_{s'}}). \tag{3.20}$$

Die Datenabhängigkeit $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i})$ ist durch die Invertierbarkeit von $\mathbf{L}_{s'}$ nur noch von Iterationsvektor $\mathbf{i}$ abhängig. Wenn zusätzlich $\mathbf{L}_{s'}^{-1}\mathbf{R}_{s's_t}^{k_{s'}} = \mathbf{E}$ gilt, ist die Datenabhängigkeit $\mathbf{d}_{s's_t}^{k_{s'}}(\mathbf{i}', \mathbf{i}) = \mathbf{d}_{s's_t}^{k_{s'}}$ konstant. Durch die Bedingung $\mathbf{L}_{s'}^{-1}\mathbf{R}_{s's_t}^{k_{s'}} = \mathbf{E}$ muss gelten $\mathbf{L}_{s'} = \mathbf{R}_{s's_t}^{k_{s'}}$, denn $\mathbf{L}_{s'}^{-1}\mathbf{L}_s = \mathbf{E}$ und $\mathbf{R}_{s's_t}^{k_{s'},-1}\mathbf{R}_{s's_t}^{k_{s'}} = \mathbf{E}$.

Im Weiteren betrachten wir entsprechend der Definition 3.1 für UItAs nur uniforme Abhängigkeitsvektoren. Das Abhängigkeitsgebiet $\mathcal{G}_{s's_t}^{k_{s'}}$ kann für UItAs mittels

$$\mathcal{G}_{s's_t}^{k_{s'}} = \left\{ (\mathbf{i}', \mathbf{i}) \,|\, \mathbf{i} \in \mathcal{I}_{s_t} \wedge \mathbf{i}' = \mathbf{i} - \mathbf{d}_{s's_t}^{k_{s'}} \right\} \tag{3.21}$$

bestimmt werden.

Es soll nun ein Graph definiert werden, welcher die Datenabhängigkeiten im Algorithmus repräsentiert.

Definition 3.4 (Reduzierter Abhängigkeitsgraph (RAG))
Im reduzierten Abhängigkeitsgraph $\langle \mathcal{S}, \mathcal{E} \rangle$ bilden die Zuweisungen s des UItA die Menge der Knoten $\mathcal{S}$. Eine Kante $e \in \mathcal{E}$ des RAG stellt die Datenabhängigkeit zwischen der Zuweisung $\sigma(e) = s'$ und der Zuweisung $\delta(e) = s$ dar, wobei $\sigma(e)$ die Quelle und $\delta(e)$ die Senke der Kante e bezeichnet. Jede Kante $e \in \mathcal{E}$ wird mit dem Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{d}^{k_{s'}}_{s's_t}$ und der Menge $\mathcal{T}(e) = \{t|...\}$ gewichtet, wobei die Menge $\mathcal{T}(e)$ die Teilzuweisungen t enthält, für die diese Datenabhängigkeit existiert. Somit können mehrere Abhängigkeitsvektoren $\mathbf{d}^{k_{s'}}_{s's_t}$ durch eine Kante e dargestellt werden. Aus den Iterationsräumen $\mathcal{I}_{s_t}$, $t \in \mathcal{T}(e)$ und dem Abhängigkeitsvektor $\mathbf{d}(e)$ kann entsprechend (3.21) das Abhängigkeitsgebiet $\mathcal{G}(e) = \bigcup_{t \in \mathcal{T}(e)} \mathcal{G}^{k_{s'}}_{s's_t}$ ermittelt werden.

Beispiel 3.4
Das FIR Filter aus Beispiel 3.1 hat sechs Datenabhängigkeiten $e_1, e_2, \ldots, e_6$ mit den Abhängigkeitsvektoren:

$$\mathbf{d}(e_1) = \mathbf{d}^1_{s_1 s_{1_2}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad \mathbf{d}(e_2) = \mathbf{d}^1_{s_1 s_{3_1}} = \mathbf{d}^1_{s_1 s_{3_2}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$
$$\mathbf{d}(e_3) = \mathbf{d}^1_{s_2 s_{2_3}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \mathbf{d}(e_4) = \mathbf{d}^1_{s_2 s_{3_1}} = \mathbf{d}^1_{s_2 s_{3_2}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$
$$\mathbf{d}(e_5) = \mathbf{d}^1_{s_3 s_{3_2}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad \mathbf{d}(e_6) = \mathbf{d}^1_{s_3 s_{4_1}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Die Abbildungen 3.2(a) und 3.2(b) zeigen die Abhängigkeitsvektoren im Iterationsraum $\mathcal{I}$ und im reduzierten Abhängigkeitsgraph. Die Teilräume $\mathcal{I}_{s_t}$ der Zuweisungen $s \in \mathcal{S}$ sind rechts neben dem Iterationsraum $\mathcal{I}$ dargestellt. Die Teilzuweisungen $t \in \mathcal{T}(e)$ der Kante e wurden in Abbildung 3.2(b) in geschweiften Klammern angegeben.

Aus der Darstellung der Abhängigkeitsvektoren im Iterationsraum sind die Datenabhängigkeiten zwischen den Iterationen erkennbar. Der RAG verdeutlicht die Datenabhängigkeiten zwischen den Zuweisungen.

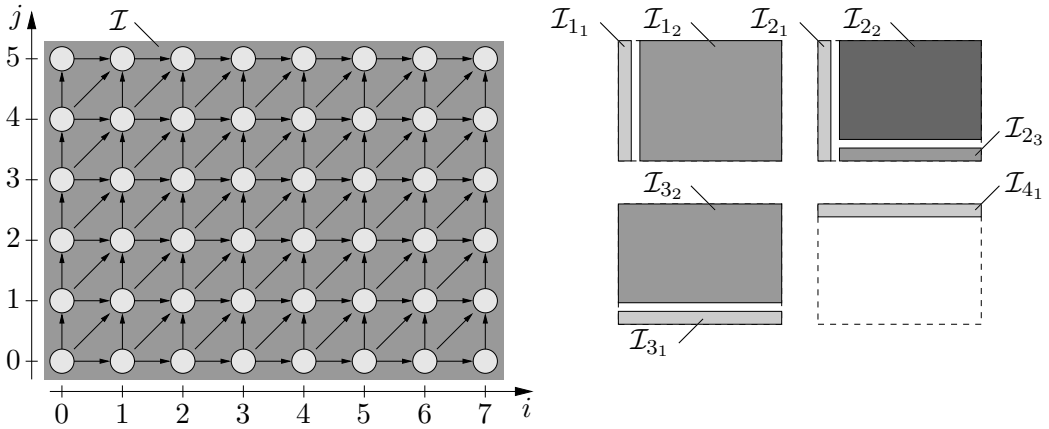
3.3 Variablen- , Teilzuweisungs- und Zuweisungstypen

Die Variablen $y_s[.]$ bzw. $y_r[.]$, die Teilzuweisungen s_t und die Zuweisungen s können nach ihren Eigenschaften kategorisiert werden. In den nachfolgenden Unterkapiteln werden die unterschiedlichen Variablen-, Teilzuweisungs- und Zuweisungstypen vorgestellt.

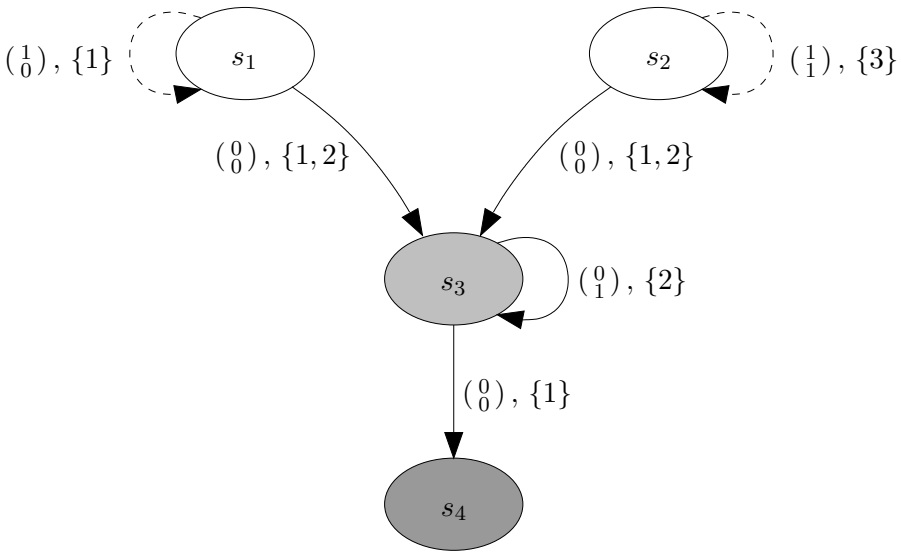
3.3.1 Variablentypen

Die Variablen können nach ihren Verwendung in *interne Variablen* $y_s[.] \in \mathcal{Y}_I$ und *globale Variablen* $y_s[.], y_r[.] \in \mathcal{Y}_G$ unterteilt werden. Interne Variablen werden nur zur Abarbeitung des Algorithmus benötigt. Die globalen Variablen sind dagegen die Schnittstelle zur Anwendung, in welcher der Algorithmus verwendet wird.

Die globalen Variablen können wir in *Eingabevariablen* $y_s[.], y_r[.] \in \mathcal{Y}_E \subset \mathcal{Y}_G$ und *Ausgabevariablen* $y_s[.] \in \mathcal{Y}_A \subset \mathcal{Y}_G$ unterteilen. Die Mengen $\mathcal{Y}_E$ und $\mathcal{Y}_A$ müssen nicht disjunkt sein. Ist eine Variable sowohl Eingabevariable $y_s[.] \in \mathcal{Y}_E$ als auch Ausgabevariable $y_s[.] \in \mathcal{Y}_E$, dann muss entsprechend Definition 3.1 gelten $\mathcal{L}_s \cap \mathcal{R}^{k_s}_{s\,s'_t} = \emptyset$.



(a) Iterationsraum mit $I = 8$ und $J = 6$



(b) RAG

Abbildung 3.2: Iterationsraum und RAG des FIR-Filters

Programm 3.2 IIR-Filter in C-Syntax

```
for( unsigned i = 0; i < 1024; i++ ){
    y_x[i] = 0;
    for( unsigned j = 0; j < 20; j++ ){
        y_x[i] += a[j]*x[i-j];
    }
    y_y[i] = 0
    for( unsigned l = 0; l < 16; l++ ){
        y_y[i] += b[l]*y[i-l];
    }
    y[i] = y_x[i] + y_y[i];
}
```

Beispiel 3.5 (Algorithmus mit Variable $y_s[.] \in \mathcal{Y}_E$ und $y_s[.] \in \mathcal{Y}_A$)
Die Variable $y[.]$ des Infinite Inpulse Response (kurz: IIR) Filters ist ein Beispiel für eine globale Variable, welche sowohl Eingabe- als auch Ausgabevariable ist. Programm 3.2 zeigt den Algorithmus in C-Syntax. Wird der Algorithmus in einen UItA umgewandelt, so müssen die Variablen $y[-11], y[-10], \ldots, y[-1]$ global gelesen werden, wogegen die Variablen $y[0], y[1], \ldots, y[1023]$ global geschrieben werden. Die Variablen $y[0], y[1], \ldots, y[1023]$ dürfen nicht global gelesen werden, sondern müssen intern zur Verfügung gestellt werden.

Der Algorithmus 3.2 zeigt den diskutierten Teil des IIR-Filters. In der Teilzuweisung $s_{6,1}$ wird die globale Variable $y[.]$ gelesen. Das Schreiben der globalen Variable $y[.]$ findet in Zuweisung s_{10} statt. Die Indexräume $\mathcal{R}^1_{y\,s_{6_1}}$ der Teilzuweisung s_{6_1} und $\mathcal{L}_y = \mathcal{L}_{s_{10}}$ der Zuweisung s_{10} sind disjunkt. Die internen Variablen $y_y[.]$ und $y_{y_t}[.]$ stellen den Wert der globalen Variable $y[.]$ für die Berechnung zur Verfügung.

Beispiel 3.6 (Variablentypen des FIR-Filters)
In Algorithmus 3.1 wurde das FIR-Filter als UItA vorgestellt. Dieser Algorithmus hat sechs Variablen, welche in der Menge $\mathcal{Y} = \{a[.], x[.], y[.], y_a[.], y_x[.], y_y[.]\}$ zusammengefasst sind. Die Menge der Eingabevariablen des Algorithmus ist $\mathcal{Y}_E = \{a[.], x[.]\}$ und die Menge der Ausgabevariablen ist $\mathcal{Y}_A = \{y[.]\}$. Somit ist die Menge der globalen Variablen $\mathcal{Y}_G = \mathcal{Y}_E \cup \mathcal{Y}_A = \{a[.], x[.], y[.]\}$. Die Menge der internen Variablen beinhaltet folglich $\mathcal{Y}_I = \mathcal{Y} \backslash \mathcal{Y}_G = \{y_a[.], y_x[.], y_y[.]\}$ die drei Variablen $y_a[.]$, $y_x[.]$ und $y_y[.]$.

Eine besondere Eigenschaft der internen Variablen $y_s[.] \in \mathcal{Y}_I$ ist, dass der Index dieser Variablen verändert werden kann. Ist diese Indextransformation $\mathcal{L}_s \to \mathcal{L}'_s$ eine bijektive Abbildung, so erhalten wir wieder einen gültigen Algorithmus, denn für den Algorithmus ist es unerheblich, ob die internen Daten im Indexraum $\mathcal{L}_s$ oder Indexraum $\mathcal{L}'_s$ zwischengespeichert werden.

Somit können die Indexfunktionen der internen Variablen $y_s[.] \in \mathcal{Y}_I$ immer so verändert werden, dass die Indexfunktionen $l_s(\mathbf{i})$ und $r^{k_s}_{s\,s'_t}(\mathbf{i})$ die folgende Form haben:

$$l_s(\mathbf{i}) = \mathbf{L}_s\mathbf{i} + \mathbf{l}_s = \mathbf{i} \qquad \text{bzw.} \qquad r^{k_s}_{s\,s'_t}(\mathbf{i}) = \mathbf{R}^{k_s}_{s\,s'_t}\mathbf{i} + \mathbf{r}^{k_s}_{s\,s'_t} = \mathbf{i} - \mathbf{d}^{k_s}_{s\,s'_t}. \tag{3.22}$$

Algorithmus 3.2 IIR-Filter

$$\dots$$

$$s_6: \quad y_{y_t}\begin{bmatrix} i \\ j \end{bmatrix} = \begin{cases} y[i-j], & \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{6_1} \\ y_y\begin{bmatrix} i \\ j \end{bmatrix}, & \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{6_2} \\ y_{y_t}\begin{bmatrix} i-1 \\ j-1 \end{bmatrix}, & \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{6_3} \end{cases}$$

$$s_7: \quad y_{s_y}\begin{bmatrix} i \\ j \end{bmatrix} = y_b\begin{bmatrix} i \\ j \end{bmatrix} \cdot y_{y_t}\begin{bmatrix} i \\ j \end{bmatrix}, \qquad \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{7_1}$$

$$s_8: \quad y_{y_y}\begin{bmatrix} i \\ j \end{bmatrix} = \begin{cases} y_{s_y}\begin{bmatrix} i \\ j \end{bmatrix}, & \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{8_1} \\ y_{y_y}\begin{bmatrix} i \\ j+1 \end{bmatrix} + y_{s_y}\begin{bmatrix} i \\ j \end{bmatrix}, & \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{8_2} \end{cases}$$

$$s_9: \quad y_y\begin{bmatrix} i \\ j \end{bmatrix} = y_{y_x}\begin{bmatrix} i \\ j \end{bmatrix} + y_{y_y}\begin{bmatrix} i \\ j+1 \end{bmatrix}, \qquad \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{9_1}$$

$$s_{10}: \quad y[i] = y_y\begin{bmatrix} i \\ j \end{bmatrix}, \qquad \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{10_1}$$

mit
$$\mathcal{I}_{6_1} = \{\mathbf{i} = \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 \mid i = 0 \wedge 1 \le j \le J_y - 1\}$$
$$\mathcal{I}_{6_2} = \{\mathbf{i} = \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge j = 0\}$$
$$\mathcal{I}_{6_3} = \{\mathbf{i} = \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 \mid 1 \le i \le I - 1 \wedge 1 \le j \le J_y - 1\}$$
$$\mathcal{I}_{7_1} = \{\mathbf{i} = \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge 1 \le j \le J_y - 1\}$$
$$\mathcal{I}_{8_1} = \{\mathbf{i} = \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge j = J_y - 1\}$$
$$\mathcal{I}_{8_2} = \{\mathbf{i} = \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge 1 \le j \le J_y - 2\}$$
$$\mathcal{I}_{9_1} = \mathcal{I}_{10_1} = \{\mathbf{i} = \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge j = 0\}$$

Erfüllen die Indexfunktionen $l_s(\mathbf{i})$ und $r^{k_s}_{s\,s'_t}(\mathbf{i})$ der internen Variable $y_s[.] \in \mathcal{Y}_I$ nicht die Form aus (3.22), so können sie durch

$$l_s(\mathbf{i}) = \mathbf{L}_s(\mathbf{L}_s^{-1}\mathbf{i} - \mathbf{L}_s^{-1}\mathbf{l}_s) + \mathbf{l}_s \qquad = \mathbf{i} \qquad \text{und} \tag{3.23}$$

$$r^{k_s}_{s\,s'_t}(\mathbf{i}) = \mathbf{R}^{k_s}_{s\,s'_t}(\mathbf{L}_s^{-1}\mathbf{i} - \mathbf{L}_s^{-1}\mathbf{l}_s) + \mathbf{r}^{k_s}_{s\,s'_t} \qquad = \mathbf{i} - \mathbf{d}^{k_s}_{s\,s'_t} \tag{3.24}$$

ersetzt werden. Diese Transformation ist eine bijektive Abbildung.

Im Weiteren werden in dieser Arbeit die Indexfunktionen der internen Variablen $y_s[.] \in \mathcal{Y}_I$ immer die Form entsprechend (3.22) haben.

3.3.2 Teilzuweisungstypen

Die einfachste Teilzuweisung ist eine *direkte Datenteilzuweisung*

$$s_t: \qquad y_s[l_s(\mathbf{i})] = y_{s'}[r^{k_{s'}}_{s'\,s_t}(\mathbf{i})], \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.25}$$

Hierbei hat die Teilzuweisung s_t nur eine rechtsseitige Variable und die Funktion F_{s_t} ist eine identische Funktion[4].

[4] identische Funktion f: $f(x) = x$

Die direkte Datenteilzuweisung s_t kann nach dem Typ der links- bzw. rechtsseitigen Variablen unterschieden werden in:

- *Eingabedatenteilzuweisung*:

$$s_t : \qquad y_s[\mathbf{i}] = y_{s'}[r^{k_{s'}}_{s's_t}(\mathbf{i})], \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.26}$$

  Die linksseitige Variable $y_s[.]$ ist eine interne Variable $y_s[.] \in \mathcal{Y}_I$ und die rechtsseitige Variable $y_{s'}[.]$ ist eine Eingabevariable $y_{s'}[.] \in \mathcal{Y}_E \subset \mathcal{Y}_G$.

- *Ausgabedatenteilzuweisung*:

$$s_t : \qquad y_s[l_s(\mathbf{i})] = y_{s'}[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.27}$$

  Die linksseitige Variable $y_s[.]$ ist eine Ausgabevariable $y_s[.] \in \mathcal{Y}_A \subset \mathcal{Y}_G$ und rechtsseitige Variable $y_{s'}[.]$ ist eine interne Variable $y_{s'}[.] \in \mathcal{Y}_I$. Der Abhängigkeitsvektor $\mathbf{d}^{k_{s'}}_{s's_t}$ der internen Variable $y_{s'}[.]$ ist $\mathbf{d}^{k_{s'}}_{s's_t} = \mathbf{0}$.

- *Interndatenteilzuweisung*:

$$s_t : \qquad y_s[\mathbf{i}] = y_{s'}[\mathbf{i} - \mathbf{d}^{k_{s'}}_{s',s_t}], \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.28}$$

  Sowohl die linksseitige Variable y_s als auch die rechtsseitige Variable $y_{s'}[.]$ sind interne Variablen $y_s, y_{s'}[.] \in \mathcal{Y}_I$.

Zwei besondere Interndatenteilzuweisungen sollen nun eingeführt werden:

- *Lokale Interndatenteilzuweisung*

$$s_t : \qquad y_s[\mathbf{i}] = y_{s'}[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.29}$$

  Die Indexfunktion $r^{k_{s'}}_{s's_t}(\mathbf{i})$ der rechtsseitigen Variable $y_{s'}[.]$ ist $r^{k_{s'}}_{s's_t}(\mathbf{i}) = \mathbf{i}$.

- *Propagierungsteilzuweisung*

$$s_t : \qquad y_s[\mathbf{i}] = y_s[\mathbf{i} - \mathbf{d}^{k_s}_{s\,s_t}], \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.30}$$

  Bei dieser Zuweisung ist die linksseitige Variable gleich der rechtsseitigen Variable. Propagierungsteilzuweisung entstehen unter anderem durch die Lokalisierung eines AInA oder AItA.

  Der Anhängigkeitsvektor $\mathbf{d}^{k_s}_{s\,s_t}$ der Propagierungsteilzuweisung s_t beschreibt den Transfer der Variable $y_s[.]$ im Iterationsraum. Dieser Vektor wird nachfolgend auch als Propagierungsvektor bezeichnet.

  In dieser Arbeit beschränken wir die Elemente $d^{k_s}_{s\,s_t,i}$ des Propagierungsvektors $\mathbf{d}^{k_s}_{s\,s_t}$ auf $|d^{k_s}_{s\,s_t,i}| \leq 1$. Dadurch wird sichergestellt, dass allen Variablen $y_s[.]$, die sich in der durch den Propagierungsvektor $\mathbf{d}^{k_s}_{s\,s_t}$ vorgegebenen Raumrichtung befinden, der gleiche Wert zugewiesen wird.

  Im RAG wird ein Propagierungsvektor mit einer gestrichelten Linie dargestellt, wie in Abbildung 3.2(b) für das FIR-Filter zu sehen ist.

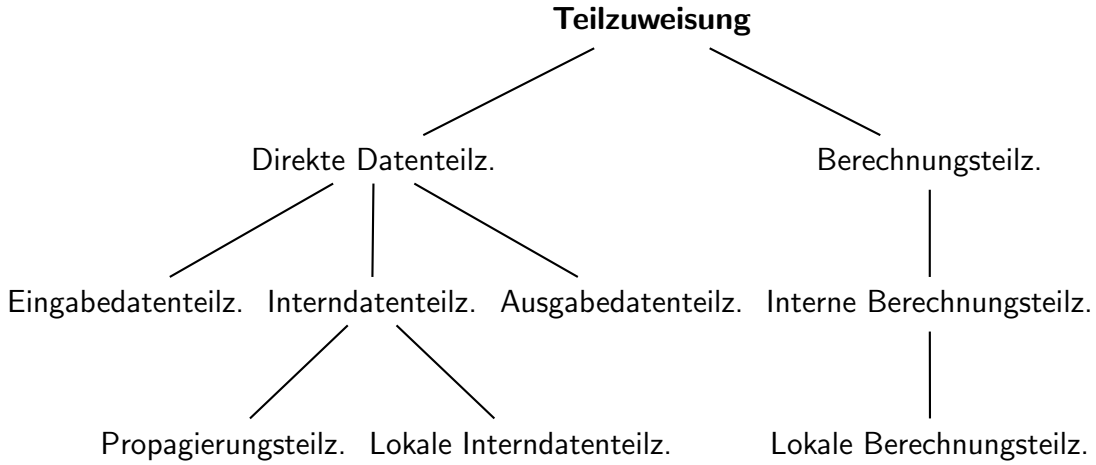


Abbildung 3.3: Typen von Teilzuweisungen

Neben der direkten Datenteilzuweisung gibt es noch die *Berechnungsteilzuweisung*

$$s_t: \qquad y_s[l_s(\mathbf{i})] = F_{s_t}\left(\ldots, y_{s'}[r^{k_s}_{s's_t}(\mathbf{i})], \ldots\right), \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.31}$$

Bei ihr ist die Funktion F_{s_t} keine identische Funktion. Eine Berechnungsteilzuweisung, bei der alle Variablen interne Variablen sind, bezeichnen wir als *interne Berechnungsteilzuweisung*

$$s_t: \qquad y_s[\mathbf{i}] = F_{s_t}\left(\ldots, y_{s'}[\mathbf{i} - \mathbf{d}^{k_{s'}}_{s's_t}], \ldots\right) \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.32}$$

Haben alle rechtsseitigen Variablen weiterhin nur Indexfunktionen mit $\mathbf{d}^{k_{s'}}_{s's_t} = \mathbf{0}$, so nennen wir diese Teilzuweisung *lokale Berechnungsteilzuweisung*

$$s_t: \qquad y_s[\mathbf{i}] = F_{s_t}\left(\ldots, y_{s'}[\mathbf{i}], \ldots\right), \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{3.33}$$

Abschließend sind alle in diesem Kapitel vorgestellten Teilzuweisungstypen in Abbildung 3.3 in einem Baum dargestellt. Dieser Baum zeigt sehr gut die Beziehungen zwischen den Teilzuweisungstypen.

Beispiel 3.7 (Teilzuweisungstypen beim FIR-Filteralgorithmus)
Das FIR-Filter entsprechend Algorithmus 3.1 hat neun Teilzuweisungen. Die Teilzuweisungen s_{1_1}, s_{2_1} und s_{2_2} sind Eingabedatenteilzuweisungen für die Variablen $a[.]$ und $x[.]$. Die Teilzuweisungen s_{1_2} und s_{2_3} sind Propagierungsteilzuweisungen. Die Teilzuweisungen s_{3_1} und s_{3_2} gehören zur Gruppe der Berechnungsteilzuweisungen, wobei s_{3_1} eine lokale Berechnungsteilzuweisung und s_{3_1} eine interne Berechnungsteilzuweisung. Die Ausgabe der Daten für die Variable $y[.]$ erfolgt mit Hilfe der Ausgabedatenteilzuweisung s_{4_1}.

3.3.3 Zuweisungstypen

Basierend auf den Teilzuweisungstypen sollen nun Zuweisungstypen definiert werden. Insgesamt sechs Zuweisungstypen sollen unterschieden werden.

- Eine *Eingabedatenzuweisung* ist eine Zuweisung s, welche mindestens eine Eingabedatenteilzuweisung s_t enthält und deren andere Teilzuweisungen Interndatenteilzuweisungen sind. Die Knoten von Eingabedatenzuweisungen werden im RAG mit weißer Hintergrundfarbe gezeichnet.

- Eine *Ausgabedatenzuweisung* ist eine Zuweisung s, welche nur Ausgabedatenteilzuweisungen s_t enthält. Die Knoten von Ausgabedatenzuweisungen werden im RAG mit dunkelgrauer Hintergrundfarbe dargestellt.

- Eine *Interndatenzuweisung* ist eine Zuweisung s, welche nur Interndatenteilzuweisungen s_t enthält. Die Knoten von Interndatenzuweisungen werden im RAG mit hellgrauer Hintergrundfarbe dargestellt.

- Eine *interne Berechnungszuweisung* ist eine Zuweisung s, welche mindestens eine interne Berechnungsteilzuweisung s_t enthält und deren andere Teilzuweisungen Interndatenteilzuweisungen sind. Die Knoten von internen Berechnungszuweisungen werden im RAG mit grauer Hintergrundfarbe dargestellt.

- Eine *lokale Berechnungszuweisung* ist eine Zuweisung s, welche mindestens eine lokale Berechnungsteilzuweisung s_t enthält und deren andere Teilzuweisungen lokale Interndatenteilzuweisungen sind. Die Knoten von lokalen Berechnungszuweisungen werden im RAG wie die Knoten von internen Berechnungszuweisungen mit grauer Hintergrundfarbe dargestellt. Den Typ der Berechnungszuweisung ermittelt man im RAG anhand der ankommenden Kanten $e \in \mathcal{E}$ des Knotens s ($\delta(e) = s$). Sind die Abhängigkeitsvektoren aller dieser Kanten $\mathbf{d}(e) = \mathbf{0}$, so ist die Zuweisung s eine lokale Berechnungszuweisung, anderenfalls ist die Zuweisung eine interne Berechnungszuweisung.

- Eine *Propagierungszuweisung* ist eine Zuweisung s, welche aus Interndatenteilzuweisung oder Eingabeteilzuweisung und ein oder mehreren Propagierungsteilzuweisungen besteht. Die Abhängigkeitsvektoren der Propagierungsteilzuweisungen sind linear unabhängig. Eine Propagierungszuweisung ist eine Sonderform der Interndatenzuweisung oder der Eingabezuweisung, abhängig davon, ob eine Teilzuweisung eine Eingabeteilzuweisung ist oder nicht. Propagierungszuweisungen, welche gleichzeitig Interndatenzuweisungen sind, werden als interne Propagierungszuweisungen bezeichnet und Propagierungszuweisungen, welche gleichzeitig Eingabezuweisungen sind, werden als Eingabe- und Propagierierungszuweisungen bezeichnet. Eine Eingabe- und Propagierierungszuweisung kann leicht in eine Eigabezuweisung mit einer Teilzuweisung und eine interne Propagierungszuweisung umgewandelt werden.

Interndatenzuweisungen, interne Berechnungszuweisungen und lokale Berechnungszuweisungen werden in der Arbeit zukünftig unter dem Begriff *interne Zuweisungen* zusammengefasst. Die Eingabedatenzuweisungen und die Ausgabedatenzuweisungen werden kurz als *E/A-Zuweisungen* bezeichnet.

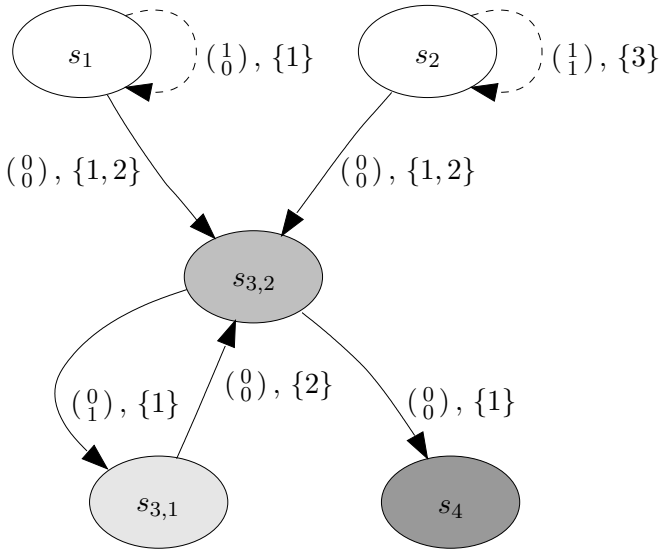


Abbildung 3.4: RAG des FIR-Filters mit lokaler Berechnungszuweisung

Beispiel 3.8 (Zuweisungstypen beim FIR-Filteralgorithmus)
Das FIR-Filter entsprechend Algorithmus 3.1 hat vier Zuweisungen. Die Zuweisungen s_1 und s_2 sind Eingabe- und Propagierungszuweisungen, Zuweisung s_3 ist eine interne Berechnungszuweisung und Zuweisung s_4 ist eine Ausgabedatenzuweisung.

Ersetzt man die Zuweisung s_3 des Algorithmus 3.1 durch die folgenden beiden Zuweisungen

$$s_{3,1}: \quad y_{y_t}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = y_y\left[\begin{smallmatrix} i \\ j-1 \end{smallmatrix}\right], \qquad \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{3,1_2}$$

$$s_{3,2}: \quad y_y\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} y_a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \cdot y_x\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{3,2_1} \\ y_{y_t}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] + y_a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \cdot y_x\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{3,2_2} \end{cases}$$

mit $\mathcal{I}_{3,1_1} = \mathcal{I}_{3,2_2} = \{\mathbf{i} = \binom{i}{j} \in \mathbb{Z}^2 \mid 0 \leq i \leq I-1 \wedge 1 \leq j \leq J-1\}$, $\mathcal{I}_{3,2_1} = \{\mathbf{i} = \binom{i}{j} \in \mathbb{Z}^2 \mid 0 \leq i \leq I-1 \wedge j = 0\}$, $I = 128$ und $J = 8$, so ist Zuweisung $s_{3,1}$ eine Interndatenzuweisung und Zuweisung $s_{3,2}$ eine lokale Berechnungszuweisung. In Abbildung 3.4 ist der RAG des modifizierten FIR-Filteralgorithmus dargestellt.

Eine interne Berechnungszuweisung kann immer in eine lokale Berechnungszuweisung umgewandelt werden. Hierzu fügen wir für jede rechtsseitige Variable $y_s[.]$, für die gilt: $\mathbf{d}^{k_s}_{s\,s'_t} \neq \mathbf{0}$, eine neue Zuweisung h mit $y_h[\mathbf{i}] = y_s[\mathbf{i} - \mathbf{d}^{k_s}_{s\,s'_t}]$ zu dem Algorithmus hinzu. Die Variable $y_h[.]$ ersetzt nun die Variable $y_s[.]$ in Teilzuweisung s'_t. Der Iterationsraum der Zuweisung h wird mit $\mathcal{I}_h$ bezeichnet. Dieser Raum ist gleich dem Iterationsraum $\mathcal{I}_{s'_t}$ der Teilzuweisung s'_t. In Beispiel 3.8 wurde dieses Vorgehen bereits für Variable $y_y[.]$ vorgestellt.

Durch das Hinzufügen der Zuweisungen h erhöht sich der lokale Datentransfer[5], welcher zu erhöhtem Energieverbrauch in der Zielarchitektur führen kann. Dieser lokale

[5]Datentransfer innerhalb einer Iteration

Datentransfer kann nach der Abbildung des Algorithmus auf die Zielarchitektur, wenn notwendig, wieder beseitigt werden. Für die in den folgenden Kapiteln vorgestellte Abbildungsstrategie bieten die lokalen Berechnungszuweisungen große Vorteile. So zerfällt der Iterationsraum von lokalen Berechnungszuweisungen bei der Partitionierung nicht in mehrere Teilräume, siehe Abschnitt 4.2.3. Weiterhin lassen sich redundante Datentransfers zwischen Iterationen schneller ermitteln und beseitigen, siehe Abschnitt 7.1.2.

3.4 Algorithmuskonvention

Ein Algorithmus kann als UItA in verschiedenen Versionen beschrieben werden. Für diese Arbeit soll eine Algorithmuskonvention festgelegt werden, die Ausgangspunkt für unseren Abbildungsprozess ist.

Ein UItA soll in dieser Arbeit nur aus Eingabe-, Ausgabe-, Interndatenzuweisungen und internen Berechnungszuweisungen bestehen. Die Ausgabezuweisungen besitzen dabei immer nur eine Teilzuweisung. Eine noch strengere Konvention ist die Beschränkung der Berechnungszuweisungen auf lokale Berechnungszuweisungen.

Im Allgemeinen kann jeder UItA durch Hinzufügen von internen Variablen und Interndatenzuweisungen in einen UItA mit oben beschriebener Konvention überführt werden.

Kapitel 4

Algorithmentransformationen

In diesem Abschnitt sollen zwei Algorithmentransformationen, die Reindizierung und die mehrstufige Partitionierung, vorgestellt werden, welche eine wesentliche Grundlage für unsere Abbildungmethode bilden.

Die möglichen Algorithmentransformationen für die Klasse der uniform-iterativen Algorithmen (UItA) sind sehr eingeschränkt, da der transformierte Algorithmus wieder ein UItA sein soll. Allgemeinere Algorithmentransformationen wie z. B. Lokalisierung [Thi89][RTRK89] und globale Schleifentransformationen [WL98][CDWD01][DCDM00] dienen im Zusammenhang mit dieser Arbeit vorrangig der Umwandlung von stückweise linearen bzw. affin-indizierten Algorithmen in die Klasse der uniform-iterativen Algorithmen. Diese Umwandlung soll in dieser Arbeit nicht weiter betrachtet werden.

Die Reindizierung verändert die Interationsräume $\mathcal{I}_s$ der Zuweisungen $s \in \mathcal{S}$ eines Algorithmus und die Abhängigkeitsvektoren $\mathbf{d}(e)$ der Datenabhängigkeiten $e \in \mathcal{E}$, welche zwischen den Zuweisungen $s' = \sigma(e)$ und $s = \delta(e)$ existieren. Dabei bleibt die Dimension des Algorithmus unverändert. Bei der Reindizierung wird zwischen der Verschiebung der Iterationsräume zueinander bzw. absolut sowie der Veränderung der Form des gemeinsamen Iterationsraums $\mathcal{I}$ unterschieden.

Die mehrstufige Partitionierung transformiert einen Algorithmus mit einem n-dimensionalen Iterationsraum $\mathcal{I} \subset \mathbb{Z}^n$ in einen Algorithmus mit einem $(p+1)n$-dimensionalen Iterationsraum $\mathcal{I}^\kappa \subset \mathbb{Z}^{(p+1)n}$, wobei p die Anzahl der Partitionierungsstufen angibt. Durch die Restriktionen der Partitionierung ist sichergestellt, dass die Partitionierung eine bijektive Abbildung ist.

Die beiden Algorithmentransformationen Reindizierung und mehrstufige Partitionierung können auch als globale Schleifentransformationen interpretiert werden. So kann die Reindizierung als Schleifenverzerrung (eng.: loop skewing) und die Partitionierung als Schleifenfaltung (eng.: loop tiling) gedeutet werden [WL98].

4.1 Reindizierung

Mit der Reindizierung kann die Lage und die Form der Iterationsräume $\mathcal{I}_s \in \mathbb{Z}^n$ der Zuweisungen $s \in \mathcal{S}$ im Raum $\mathbb{Z}^n$ verändert werden. Die Definition für die Reindizierung eines UItA lautet:

Definition 4.1 (Reindizierung)
Ein UItA wird durch die Reindizierung $\rho_s : \mathbb{Z}^n \to \mathbb{Z}^n$ in einen neuen UItA transformiert. Für die Abbildung ρ_s muss gelten:

$$\widetilde{\mathbf{i}} = \rho_s(\mathbf{i}) = \mathbf{C}_s\mathbf{i} + \mathbf{c}_s, \qquad \mathbf{C}_s = \mathbf{C}, \quad \forall s \in \mathcal{S}, \tag{4.1}$$

mit $\mathbf{C} \in \mathbb{Z}^{n\times n}$, $\mathbf{c}_s \in \mathbb{Z}^n$, und $|\det(\mathbf{C})| = 1$.

Die Unimodularität der Matrix $\mathbf{C}$, d. h. $|\det(\mathbf{C})| = 1$, stellt sicher, dass die transformierten Iterationsräume

$$\widetilde{\mathcal{I}}_s = \left\{\widetilde{\mathbf{i}} \,|\, \widetilde{\mathbf{A}}_s\widetilde{\mathbf{i}} \geq \widetilde{\mathbf{a}}_s,\ \widetilde{\mathbf{A}}_s = \mathbf{A}_s\mathbf{C}_s^{-1},\ \widetilde{\mathbf{a}}_s = \mathbf{a}_s + \mathbf{A}_s\mathbf{C}_s^{-1}\mathbf{c}_s\right\} \tag{4.2}$$

wieder Polyeder sind.

Die Reindizierungsfunktion $\rho_s(\mathbf{i}) = \widetilde{\mathbf{i}} = \mathbf{C}_s\mathbf{i} + \mathbf{c}_s$ reindiziert die Iterationsräume $\mathcal{I}_s$ des Ausgangsalgorithmus. Für die Indexfunktionen $\widetilde{l}_s(\widetilde{\mathbf{i}})$ und $\widetilde{r}^k_{s's}(\widetilde{\mathbf{i}})$ des transformierten Algorithmus gilt dann:

$$\widetilde{l}_s(\widetilde{\mathbf{i}}) = \widetilde{\mathbf{L}}_s\widetilde{\mathbf{i}} + \widetilde{\mathbf{l}}_s \qquad \widetilde{\mathbf{L}}_s = \mathbf{L}_s\mathbf{C}_s^{-1} \qquad \widetilde{\mathbf{l}}_s = \mathbf{l}_s - \mathbf{L}_s\mathbf{C}_s^{-1}\mathbf{c}_s \tag{4.3}$$

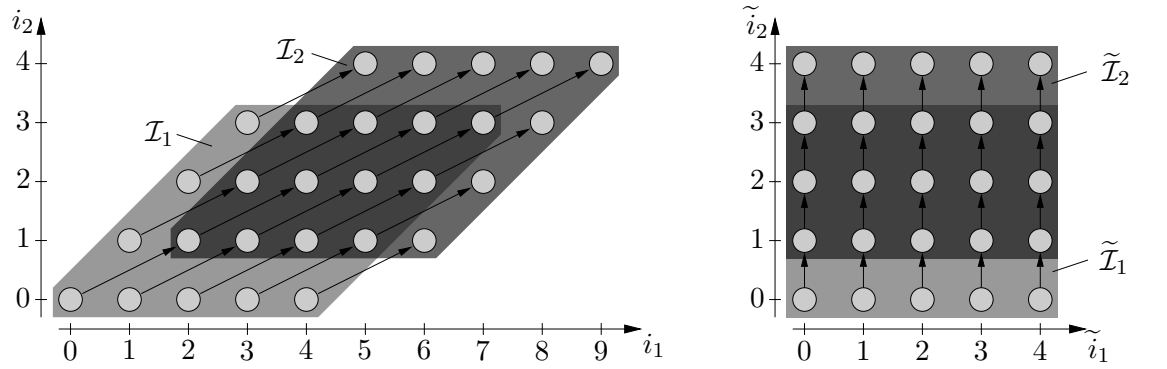
$$\widetilde{r}^{k_{s'}}_{s's_t}(\widetilde{\mathbf{i}}) = \widetilde{\mathbf{R}}^{k_{s'}}_{s's_t}\widetilde{\mathbf{i}} + \widetilde{\mathbf{r}}^{k_{s'}}_{s's_t} \qquad \widetilde{\mathbf{R}}^{k_{s'}}_{s's_t} = \mathbf{R}^{k_{s'}}_{s's_t}\mathbf{C}_s^{-1} \qquad \widetilde{\mathbf{r}}^{k_{s'}}_{s's_t} = \mathbf{r}^{k_{s'}}_{s's_t} - \mathbf{R}^{k_{s'}}_{s's_t}\mathbf{C}_s^{-1}\mathbf{c}_s. \tag{4.4}$$

Mit den in Definition 4.1 gegebenen Restiktionen für die Reindizierungsmatrizen ($\mathbf{C}_s = \mathbf{C}$ und $|\det(\mathbf{C})| = 1$) bleiben uniforme Datenabhängigkeiten $\mathbf{d}^{k_{s'}}_{s's_t}$ des Ausgangsalgorithmus uniforme Datenabhängigkeiten $\widetilde{\mathbf{d}}^{k_{s'}}_{s's_t}$ im reindizierten Algorithmus. Wenn für den Ausgangsalgorithmus $\mathbf{L}_s = \mathbf{R}^{k_{s'}}_{s's_t}$ und $\mathrm{Rg}(\mathbf{L}_s) = n$ gilt, dann folgt für den reindizierten Algorithmus: $\widetilde{\mathbf{L}}_s = \widetilde{\mathbf{R}}^{k_{s'}}_{s's_t}$ und $\mathrm{Rg}(\widetilde{\mathbf{L}}_s) = n$. Somit sind entsprechend Hilfssatz 3.1 sowohl für die Datenabhängigkeiten $\mathbf{d}^{k_{s'}}_{s's_t}$ des Ausgangsalgorithmus als auch für Datenabhängigkeiten $\widetilde{\mathbf{d}}^{k_{s'}}_{s's_t}$ des reindizierten Algorithmus die Bedingungen für uniforme Datenabhängigkeiten erfüllt.

Die Indexfunktionen der internen Variablen des reindizierten Algorithmus können, wie in Abschnitt 3.3.1 beschrieben, wieder so verändert werden, dass die Indexfunktionen dieser Variablen die Form $\widetilde{l}_s(\widetilde{\mathbf{i}}) = \widetilde{\mathbf{i}}$ bzw. $\widetilde{r}^{k_s}_{s\,s'_t}(\widetilde{\mathbf{i}}) = \widetilde{\mathbf{i}} - \widetilde{\mathbf{d}}^{k_s}_{s\,s'_t}$ haben.

Beispiel 4.1 (Reindizierung)
Ausgangspunkt ist ein Algorithmus mit zwei Zuweisungen s_1 und s_2:

$$s_1: \quad y_{s_1}[\mathbf{i}] = F_{s_1}(\ldots), \qquad \mathbf{i} \in \mathcal{I}_1$$
$$s_2: \quad y_{s_2}[\mathbf{i}] = F_{s_2}(\ldots, y_{s_1}[\mathbf{i} - \tbinom{2}{1}], \ldots), \qquad \mathbf{i} \in \mathcal{I}_2$$



(a) Iterationsräume vor Reindizierung (b) Iterationsräume nach Reindizierung

Abbildung 4.1: Reindizierung eines Algorithmus

mit $\mathcal{I}_1 = \{\mathbf{i} = \binom{i_1}{i_2} \in \mathbb{Z}^2 \mid 0 \leq i_1 - i_2 \leq 4 \wedge 0 \leq i_2 \leq 3\}$ *und*
$\mathcal{I}_2 = \{\mathbf{i} = \binom{i_1}{i_2} \in \mathbb{Z}^2 \mid 1 \leq i_1 - i_2 \leq 5 \wedge 1 \leq i_2 \leq 4\}$.

Die Variable y_{s_1} *sei eine interne Variable und die Variable* y_{s_2} *eine globale Variable. Zwischen den beiden Zuweisungen existiert eine Datenabhängigkeit* e*, für die gilt:* $\mathbf{d}(e) = \binom{2}{1}$.

Der Algorithmus soll mit der Reindizierungsmatrix $\mathbf{C} = \left(\begin{smallmatrix}1 & -1\\ 0 & 1\end{smallmatrix}\right)$ *und den Reindizierungsvektoren* $\mathbf{c}_{s_1} = \binom{0}{0}$ *und* $\mathbf{c}_{s_2} = \binom{-1}{0}$ *reindiziert werden. Die Reindizierungsmatrix erfüllt die Bedingung* $|\det(\mathbf{C})| = 1$ *aus Definition 4.1. Damit erhalten wir folgenden reindizierten Algorithmus:*

$$s_1: \quad y_{s_1}[\left(\begin{smallmatrix}1 & 1\\ 0 & 1\end{smallmatrix}\right)\widetilde{\mathbf{i}}] = F_{s_1}(\ldots), \qquad \widetilde{\mathbf{i}} \in \widetilde{\mathcal{I}}_1,$$
$$s_2: \quad y_{s_2}[\left(\begin{smallmatrix}1 & 1\\ 0 & 1\end{smallmatrix}\right)\widetilde{\mathbf{i}} + \binom{1}{0}] = F_{s_2}(\ldots, y_{s_1}[\left(\begin{smallmatrix}1 & 1\\ 0 & 1\end{smallmatrix}\right)\widetilde{\mathbf{i}} + \binom{1}{1}], \ldots), \qquad \widetilde{\mathbf{i}} \in \widetilde{\mathcal{I}}_2,$$

bzw.

$$s_1: \quad y_{s_1}[\widetilde{\mathbf{i}}] = F_{s_1}(\ldots), \qquad \widetilde{\mathbf{i}} \in \widetilde{\mathcal{I}}_1,$$
$$s_2: \quad y_{s_2}[\left(\begin{smallmatrix}1 & 1\\ 0 & 1\end{smallmatrix}\right)\widetilde{\mathbf{i}} + \binom{1}{0}] = F_{s_2}(\ldots, y_{s_1}[\widetilde{\mathbf{i}} + \binom{1}{1}], \ldots), \qquad \widetilde{\mathbf{i}} \in \widetilde{\mathcal{I}}_2.$$

wenn die Indexfunktionen der internen Variable y_{s_1} *ebenfalls reindiziert wurden.*

Für die Iterationsräume gilt: $\widetilde{\mathcal{I}}_1 = \{\widetilde{\mathbf{i}} = \binom{i}{j} \in \mathbb{Z}^2 \mid 0 \leq i \leq 4 \wedge 0 \leq j \leq 4\}$ *und*
$\widetilde{\mathcal{I}}_2 = \{\widetilde{\mathbf{i}} = \binom{i}{j} \in \mathbb{Z}^2 \mid 0 \leq i \leq 4 \wedge 1 \leq j \leq 5\}$.

Der Abhängigkeitsvektor $\mathbf{d}(e) = \binom{2}{1}$ *des Ausgangsalgorithmus beträgt nun* $\widetilde{\mathbf{d}}(e) = \binom{0}{1}$. *Abbildung 4.1 zeigt die Iterationsräume und den Abhängigkeitsvektor des Ausgangsalgorithmus und des reindizierten Algorithmus.*

Anhand des Beispiels ist gut zu erkennen, dass mit der Reindizierung die Form und die Lage der Iterationsräume $\mathcal{I}_s$ und die Länge und die Richtung der Abhängigkeitsvektoren $\mathbf{d}(e)$ beeinflusst werden kann.

Ein reindizierter Algorithmus kann durch eine weitere Reindizierung $\rho_s(\widetilde{\mathbf{i}})$ nochmals reindiziert werden. Somit ist es möglich, die gewünschten Eigenschaften für einen Algorithmus stufenweise zu erreichen.

4.2 Mehrstufige Partitionierung

Die mehrstufige Partitionierung ist die Basis für die Anpassung eines Algorithmus an eine parallele Architektur. Sie transformiert einen Algorithmus von einem n-dimensionalen Iterationsraum $\mathcal{I} \in \mathbb{Z}^n$ in einen $(p+1) \cdot n$-dimensionalen Iterationsraum $\mathcal{I}^\kappa \in \mathbb{Z}^{(p+1)\cdot n}$. Der Parameter p gibt dabei die Anzahl der Partitionierungsstufen an.

Zuerst wird ausgehend von einer einstufigen Partitionierung die mehrstufige Partitionierung des Iterationsraums vorgestellt. Anschließend wird im Abschnitt 4.2.2 die mehrstufige Partitionierung von Abhängigkeitsvektoren $\mathbf{d}(e)$ beschrieben. Im letzten Abschnitt wird dann die Transformation des gesamten Algorithmus vorgestellt.

In diesem Abschnitt wird nur die Methode der mehrstufigen Partitionierung vorgestellt. Die Interpretation und die Bestimmung der Parameter der mehrstufigen Partitionierung erfolgt in den Kapiteln 5 und 7.

4.2.1 Partitionierung des Iterationsraums

Einstufige Partitionierung

Mittels Partitionierung wird der Iterationsraum $\mathcal{I}$ in Partitionen zerlegt, welche auch als Kacheln bezeichnet werden. Die Definition für die einstufige Partitionierung lautet:

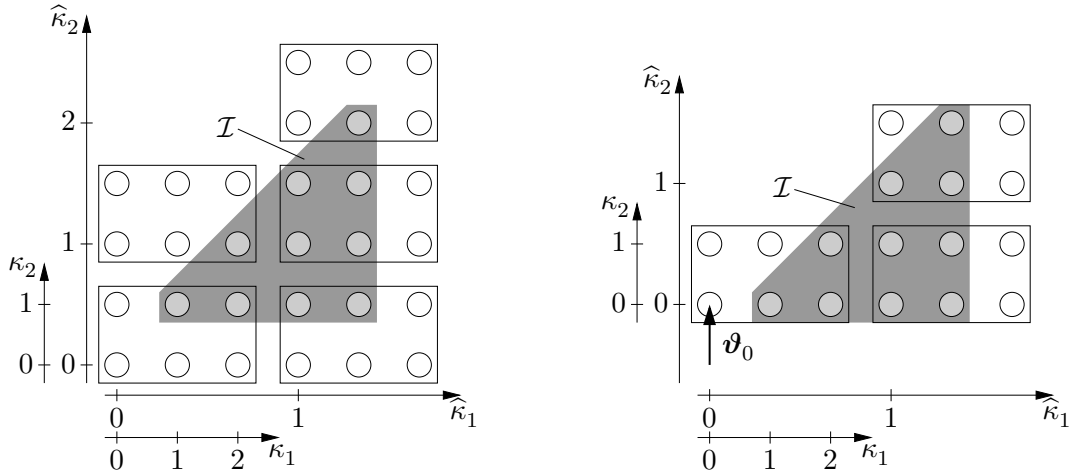
Definition 4.2 (Einstufige Partitionierung)
Ein Iterationsraum $\mathcal{I}$ kann entsprechend der Gleichung

$$\mathbf{i} = \vartheta(\mathbf{i}^\kappa) = \boldsymbol{\Theta}^\kappa \mathbf{i}^\kappa + \boldsymbol{\vartheta}_0 = \begin{pmatrix} \mathbf{E} & \boldsymbol{\Theta} \end{pmatrix} \begin{pmatrix} \boldsymbol{\kappa} \\ \widehat{\boldsymbol{\kappa}} \end{pmatrix} + \boldsymbol{\vartheta}_0 = \boldsymbol{\kappa} + \boldsymbol{\Theta}\widehat{\boldsymbol{\kappa}} + \boldsymbol{\vartheta}_0, \tag{4.5}$$

mit $\boldsymbol{\vartheta} = diag(\boldsymbol{\Theta}) \in \mathbb{N}_+^n$, $\boldsymbol{\kappa} \in \mathcal{K} \subset \mathbb{N}^n$, $0 \leq \kappa_j < \vartheta_j$, $\widehat{\boldsymbol{\kappa}} \in \widehat{\mathcal{K}} \subset \mathbb{Z}^n$, $\boldsymbol{\vartheta}_0 \in \mathbb{Z}^n$ und $0 \leq \vartheta_{0,j} < \vartheta_j$ partitioniert werden.

Durch die einstufige Partitionierung wird jeder Iteration $\mathbf{i} \in \mathcal{I}$ eineindeutig eine partitionierte Iteration $\mathbf{i}^\kappa = \begin{pmatrix} \boldsymbol{\kappa} \\ \widehat{\boldsymbol{\kappa}} \end{pmatrix}$ durch die Funktion $\vartheta(\mathbf{i}^\kappa)$ zugeordnet. Dabei beschreibt $\widehat{\boldsymbol{\kappa}}$ die Partition, in der Iteration $\mathbf{i}$ liegt, und $\boldsymbol{\kappa}$ die Lage der Iteration $\mathbf{i}$ innerhalb der Iteration. Die Partitionsmatrix $\boldsymbol{\Theta}$ beschreibt die Größe der rechtwinkligen Partition. Die Anzahl der Elemente einer Partition lässt sich mittels $\det(\boldsymbol{\Theta})$ bestimmen.

Durch den Verschiebungsvektor $\boldsymbol{\vartheta}_0$ kann die Position der Partitionen im Raum verändert werden. Die Verschiebung dient vor allem der Minimierung der Anzahl der Partitionen, die den Iterationsraum $\mathcal{I}$ vollständig überdecken.



(a) Partitionierung 1 (b) Partitionierung 2

Abbildung 4.2: Partitionierung

Der partitionierte Iterationsraum $\mathcal{I}^\kappa$ ergibt sich durch

$$\mathcal{I}^\kappa = \left\{ \mathbf{i}^\kappa = \begin{pmatrix} \boldsymbol{\kappa} \\ \widehat{\boldsymbol{\kappa}} \end{pmatrix} \mid \begin{pmatrix} \mathbf{E} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} \\ \mathbf{A} & \mathbf{A}\boldsymbol{\Theta} \end{pmatrix} \mathbf{i}^\kappa \geq \begin{pmatrix} \mathbf{0} \\ -\boldsymbol{\vartheta} + 1 \\ \mathbf{a}_0 - \mathbf{A}\boldsymbol{\vartheta}_0 \end{pmatrix} \right\} = \mathcal{K} \oplus \widehat{\mathcal{K}}. \tag{4.6}$$

Der partitionierte Iterationsraum $\mathcal{I}^\kappa$ ist die direkte Summe aus den Partitionsteilräumen $\mathcal{K}$ und $\widehat{\mathcal{K}}$. Die Partitionsteilräume werden wie folgt bestimmt:

$$\mathcal{K} = \{ \boldsymbol{\kappa} \mid \mathbf{0} \leq \boldsymbol{\kappa} < \boldsymbol{\vartheta} \}, \tag{4.7}$$

$$\widehat{\mathcal{K}} = \left\{ \widehat{\boldsymbol{\kappa}} \mid \underset{\kappa_1,\kappa_2,\ldots\kappa_n}{\mathrm{FME}} (\mathcal{I}^\kappa) \right\}, \tag{4.8}$$

wobei $\mathrm{FME}_{\kappa_1,\kappa_2,\ldots\kappa_n}(\mathcal{I}^\kappa)$ die Fourier-Motzkin Elimination der Variablen $\kappa_1, \kappa_2, \ldots \kappa_n$ aus dem Raum $\mathcal{I}^\kappa$ ist.

Beispiel 4.2 (Einstufige Partitionierung des Iterationsraums)
Abbildung 4.2 zeigt zwei Versionen der Partitionierung des Iterationsraums $\mathcal{I} = \{\mathbf{i} = \left(\begin{smallmatrix} i_1 \\ i_2 \end{smallmatrix}\right) \mid i_1 \leq 4 \wedge i_2 \geq 1 \wedge i_2 \leq i_1\}$.
Die Partitionsmatrix für beide Versionen ist $\boldsymbol{\Theta} = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 2 \end{smallmatrix}\right)$. *Sie unterscheiden sich nur durch ihren Verschiebungsvektor* $\boldsymbol{\vartheta}_0$. *Für Partitionierung 1 gilt* $\boldsymbol{\vartheta}_0 = \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)$ *und für Partitionierung 2* $\boldsymbol{\vartheta}_0 = \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$. *Der Verschiebungsvektor* $\boldsymbol{\vartheta}_0 = \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$ *hat zur Folge, dass bei Partitionierung 2 nur* $|\widehat{\mathcal{K}}_2| = 3$ *Partitionen benötigt werden, um den Iterationsraum* $\mathcal{I}$ *zu überdecken. Bei Partitionierung 1 sind hierfür* $|\widehat{\mathcal{K}}_1| = 5$ *Partitionen notwendig. Die Räume* $\mathcal{K}$ *und* $\widehat{\mathcal{K}}$ *für die beiden Partitionierungen lauten wie folgt:*

$$\mathcal{K}_1 = \mathcal{K}_2 = \left\{ \boldsymbol{\kappa} = \left(\begin{smallmatrix} \kappa_1 \\ \kappa_2 \end{smallmatrix}\right) \mid 0 \leq \kappa_1 < 3 \wedge 0 \leq \kappa_2 < 2 \right\} \tag{4.9}$$

$$\widehat{\mathcal{K}}_1 = \left\{ \widehat{\boldsymbol{\kappa}} = \left(\begin{smallmatrix} \widehat{\kappa}_1 \\ \widehat{\kappa}_2 \end{smallmatrix}\right) \mid 0 \leq \widehat{\kappa}_1 \leq 1 \wedge 0 \leq \widehat{\kappa}_2 \leq 2 \wedge \widehat{\kappa}_2 \leq \widehat{\kappa}_1 + 1 \right\} \tag{4.10}$$

$$\widehat{\mathcal{K}}_2 = \left\{ \widehat{\boldsymbol{\kappa}} = \left(\begin{smallmatrix} \widehat{\kappa}_1 \\ \widehat{\kappa}_2 \end{smallmatrix}\right) \mid 0 \leq \widehat{\kappa}_1 \leq 1 \wedge 0 \leq \widehat{\kappa}_2 \leq 1 \wedge \widehat{\kappa}_2 \leq \widehat{\kappa}_1 \right\}. \tag{4.11}$$

Das Ergebnis, das bei der Verschiebung der Partitionen mittels des Vektors $\boldsymbol{\vartheta}_0$ erreicht wird, kann auch durch eine Reindizierung des Iterationsraums mit $\mathbf{C}_s = \mathbf{E}$ und $\mathbf{c}_s = -\boldsymbol{\vartheta}_0$ erreicht werden. Aus diesem Grund wird, wenn nichts anders angegeben ist, der Verschiebungsvektor gleich $\boldsymbol{\vartheta}_0 = \mathbf{0}$ gesetzt.

Die Partitionierung (4.5) wurde in [Eck01] durch das Hinzufügen einer Basis $\mathbf{B}$ [Eck01] verallgemeinert:

$$\mathbf{i} = \vartheta(\mathbf{i}^\kappa) = \mathbf{B}\boldsymbol{\Theta}^\kappa \mathbf{i}^\kappa + \boldsymbol{\vartheta}_0 = \mathbf{B}\boldsymbol{\kappa} + \mathbf{B}\boldsymbol{\Theta}\widehat{\boldsymbol{\kappa}} + \boldsymbol{\vartheta}_0, \tag{4.12}$$

wobei die Basis $\mathbf{B}$ eine unimodulare Matrix sein muss. Durch diese Verallgemeinerung sind auch Kacheln in Form eines Parallelogramms (zweidimensionaler Iterationsraum) bzw. eines Parallelotops (n-dimensionaler Iterationsraum) möglich.

In dieser Arbeit werden nur Kacheln in der Form eines Hyperquarders betrachtet. Das ist legitim, denn durch eine geeignete Reindizierung, siehe Abschnitt 4.1, kann der Iterationsraum $\mathcal{I}$ des Algorithmus so verändert werden, dass die Partitionierung des reindizierten Iterationsraums $\widetilde{\mathcal{I}}$ mit Basis $\mathbf{B} = \mathbf{E}$ immer zur besten Lösung führt. Die Reindizierungsmatrix $\mathbf{C}$ für die Veränderung des Iterationsraums entspricht dabei genau jener Basis $\mathbf{B}$, welche bei der Partitionierung des Iterationsraums $\mathcal{I}$ des Ausgangsalgorithmus zur besten Kachelung führen würde. In Abschnitt 7.1.1 wird eine Methode vorgestellt, wie diese Reindizierungsmatrix $\mathbf{C}$ automatisch ermittelt werden kann.

Mehrstufige Partitionierung

Im Allgemeinen kann ein Iterationsraum $\mathcal{I}$ mehrfach partitioniert werden. Hierbei wird der äußere Teilraum $\widehat{\mathcal{K}}$ des partitionierten Iterationsraums $\mathcal{I}^{\boldsymbol{\kappa}}$ der i-ten Partitionierungsstufe erneut mit $\boldsymbol{\Theta}^{i+1}$ der $i+1$-ten Partitionierungsstufe partitioniert. Der Teilvektor $\widehat{\boldsymbol{\kappa}}^i$ der i-ten Partitionierungsstufe wird durch die neue Partitionierung $\boldsymbol{\kappa}^{i+1} + \boldsymbol{\Theta}^{i+1}\widehat{\boldsymbol{\kappa}}^{i+1}$ ersetzt. Für die zweistufige Partitionierung ergibt sich folgende Gleichung:

$$\mathbf{i} = \boldsymbol{\kappa}^1 + \boldsymbol{\Theta}^1 \left(\boldsymbol{\kappa}^2 + \boldsymbol{\Theta}^2 \widehat{\boldsymbol{\kappa}}^2\right) + \boldsymbol{\vartheta}_0, \tag{4.13}$$

$$\mathbf{i} = \begin{pmatrix} \mathbf{E} & \boldsymbol{\Theta}^1 & \boldsymbol{\Theta}^1\boldsymbol{\Theta}^2 \end{pmatrix} \mathbf{i}^\kappa + \boldsymbol{\vartheta}_0 = \boldsymbol{\Theta}^\kappa \mathbf{i}^\kappa + \boldsymbol{\vartheta}_0, \tag{4.14}$$

$$\mathbf{i} = \vartheta(\mathbf{i}^\kappa), \qquad \mathbf{i}^\kappa = \begin{pmatrix} \boldsymbol{\kappa}^1 & \boldsymbol{\kappa}^2 & \widehat{\boldsymbol{\kappa}}^2 \end{pmatrix}^t, \tag{4.15}$$

mit $\boldsymbol{\vartheta}^k = \operatorname{diag}(\boldsymbol{\Theta}^k) \in \mathbb{N}_+^n$, $\boldsymbol{\kappa}^k \in \mathcal{K}^k \subset N^n$, $0 \leq \kappa_j^k < \vartheta_j^k$, $k = 1, 2$, $\widehat{\boldsymbol{\kappa}}^2 \in \widehat{\mathcal{K}}^2 = \widehat{\mathcal{K}} \subset \mathbb{Z}^n$ und $0 \leq \vartheta_{0,j} < \vartheta_j^1 \cdot \vartheta_j^2$. Der partitionierte Iterationsraum $\mathcal{I}^\kappa = \mathcal{K}^1 \oplus \mathcal{K}^2 \oplus \widehat{\mathcal{K}}$ lautet:

$$\mathcal{I}^\kappa = \left\{ \begin{pmatrix} \mathbf{E} & \mathbf{0} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} & \mathbf{0} \\ \mathbf{0} & -\mathbf{E} & \mathbf{0} \\ \mathbf{A} & \mathbf{A}\boldsymbol{\Theta}^1 & \mathbf{A}\boldsymbol{\Theta}^1\boldsymbol{\Theta}^2 \end{pmatrix} \begin{pmatrix} \boldsymbol{\kappa}^1 \\ \boldsymbol{\kappa}^2 \\ \widehat{\boldsymbol{\kappa}}^2 \end{pmatrix} \geq \begin{pmatrix} \mathbf{0} \\ -\boldsymbol{\vartheta}^1 + 1 \\ \mathbf{0} \\ -\boldsymbol{\vartheta}^2 + 1 \\ \mathbf{a}_0 - \mathbf{A}\boldsymbol{\vartheta}_0 \end{pmatrix} \right\}. \tag{4.16}$$

Durch Fourier-Motzkin-Elimination der Variablen κ_j^i, mit $i \in \{1,\ 2\}$ und $1 \leq j \leq n$, lässt sich aus dem Raum $\mathcal{I}^\kappa$ der Raum $\widehat{\mathcal{K}} = \mathrm{FME}_{\kappa_j^i}(\mathcal{I}^\kappa)$ ermitteln.

Für den allgemeinen Fall einer p-stufigen Partitionierung gilt:

$$\mathbf{i} = \boldsymbol{\kappa}^1 + \boldsymbol{\Theta}^1 \left(\boldsymbol{\kappa}^2 + \boldsymbol{\Theta}^2 \left(\boldsymbol{\kappa}^3 + \boldsymbol{\Theta}^3 \left(\boldsymbol{\kappa}^4 + \ldots + \boldsymbol{\Theta}^{p-1} \left(\boldsymbol{\kappa}^p + \boldsymbol{\Theta}^p \widehat{\boldsymbol{\kappa}}^p\right)\right)\right)\right) + \boldsymbol{\vartheta}_0, \tag{4.17}$$

$$\mathbf{i} = \sum_{k=1}^{p} \left(\prod_{l=1}^{k-1} \boldsymbol{\Theta}^l\right) \boldsymbol{\kappa}^k + \left(\prod_{l=1}^{p} \boldsymbol{\Theta}^l\right) \widehat{\boldsymbol{\kappa}}^p + \boldsymbol{\vartheta}_0, \tag{4.18}$$

$$\mathbf{i} = \begin{pmatrix} \mathbf{E} & \prod_{l=1}^{1} \boldsymbol{\Theta}^l & \prod_{l=1}^{2} \boldsymbol{\Theta}^l & \ldots & \prod_{l=1}^{p} \boldsymbol{\Theta}^l \end{pmatrix} \mathbf{i}^\kappa + \boldsymbol{\vartheta}_0 = \boldsymbol{\Theta}^\kappa \mathbf{i}^\kappa + \boldsymbol{\vartheta}_0, \tag{4.19}$$

$$\mathbf{i} = \vartheta(\mathbf{i}^\kappa), \qquad \mathbf{i}^\kappa = \begin{pmatrix} \boldsymbol{\kappa}^1 & \boldsymbol{\kappa}^2 & \ldots & \boldsymbol{\kappa}^p & \widehat{\boldsymbol{\kappa}}^p \end{pmatrix}^t \tag{4.20}$$

mit $\boldsymbol{\vartheta}^k = \operatorname{diag}(\boldsymbol{\Theta}^k) \in \mathbb{N}_+^n$, $\boldsymbol{\kappa}^k \in \mathcal{K}^k \subset N^n$, $0 \leq \kappa_j^k < \vartheta_j^k$, $k = 1, \ldots, p$, $\widehat{\boldsymbol{\kappa}}^p \in \widehat{\mathcal{K}}^p = \widehat{\mathcal{K}} \subset \mathbb{Z}^n$ und $0 \leq \vartheta_{0,j} < \prod_{l=1}^{p} \vartheta_j^l$. Der partitionierte Iterationsraum $\mathcal{I}^\kappa = \bigoplus_{k=1}^{p} (\mathcal{K}^k) \oplus \widehat{\mathcal{K}}$ ergibt sich durch:

$$\mathcal{I}^\kappa = \left\{ \begin{pmatrix} \mathbf{E} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{E} & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & -\mathbf{E} & \mathbf{0} \\ \mathbf{A} & \mathbf{A}\boldsymbol{\Theta}^1 & \ldots & \mathbf{A}\prod_{k=1}^{p-1} \boldsymbol{\Theta}^k & \mathbf{A}\prod_{k=1}^{p} \boldsymbol{\Theta}^k \end{pmatrix} \begin{pmatrix} \boldsymbol{\kappa}^1 \\ \boldsymbol{\kappa}^2 \\ \vdots \\ \boldsymbol{\kappa}^n \\ \widehat{\boldsymbol{\kappa}}^p \end{pmatrix} \geq \begin{pmatrix} \mathbf{0} \\ -\boldsymbol{\vartheta}^1 + 1 \\ \mathbf{0} \\ -\boldsymbol{\vartheta}^2 + 1 \\ \vdots \\ \mathbf{0} \\ -\boldsymbol{\vartheta}^n + 1 \\ \mathbf{a}_0 - \mathbf{A}\boldsymbol{\vartheta}_0 \end{pmatrix} \right\}. \tag{4.21}$$

Den Raum $\widehat{\mathcal{K}}$ erhalten wir wieder durch Fourier-Motzkin-Elimination $\widehat{\mathcal{K}} = \mathrm{FME}_{\kappa_j^i}(\mathcal{I}^\kappa)$, mit $1 \leq i \leq p$ und $1 \leq j \leq n$.

Beispiel 4.3 (Zweistufige Partitionierung eines Iterationsraums)
Abbildung 4.3 zeigt eine zweistufige Partitionierung des Iterationsraums $\mathcal{I} = \{\mathbf{i} = \binom{i_1}{i_2} \mid i_1 \leq 4 \wedge i_2 \geq 1 \wedge i_2 \leq i_1\}$ *mit den Partitionsmatrizen*

$$\boldsymbol{\Theta}^1 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \qquad \textit{und} \qquad \boldsymbol{\Theta}^2 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}.$$

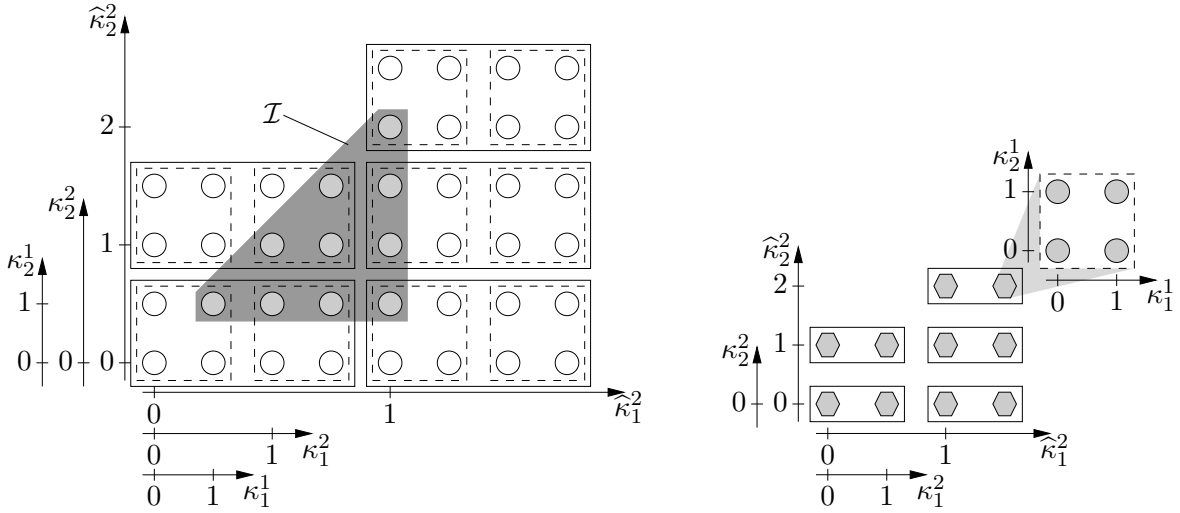
Die Partitionierung wurde in Abbildung 4.3(a) in der einfachen und in Abbildung 4.3(b) in der kompakten Form dargestellt. In der kompakten Darstellung repräsentiert jedes Sechseck eine Partition der ersten Partitionierung ($\boldsymbol{\Theta}^1 = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 2 \end{smallmatrix}\right)$).

Für die Räume $\mathcal{K}^1$, $\mathcal{K}^2$ *und* $\widehat{\mathcal{K}}$ *gilt:*

$$\mathcal{K}^1 = \left\{ \boldsymbol{\kappa}^1 = \begin{pmatrix} \kappa_1^1 \\ \kappa_2^1 \end{pmatrix} \mid 0 \leq \kappa_1^1 < 2 \wedge 0 \leq \kappa_2^1 < 2 \right\} \tag{4.22}$$

$$\mathcal{K}^2 = \left\{ \boldsymbol{\kappa}^2 = \begin{pmatrix} \kappa_1^2 \\ \kappa_2^2 \end{pmatrix} \mid 0 \leq \kappa_1^2 < 2 \wedge \kappa_2^2 = 0 \right\} \tag{4.23}$$

$$\widehat{\mathcal{K}} = \left\{ \widehat{\boldsymbol{\kappa}}^2 = \begin{pmatrix} \widehat{\kappa}_1^2 \\ \widehat{\kappa}_2^2 \end{pmatrix} \mid \widehat{\kappa}_1^2 \leq 1 \wedge \widehat{\kappa}_2^2 \geq 0 \wedge \widehat{\kappa}_2^2 = \widehat{\kappa}_1^2 + 1 \right\} \tag{4.24}$$



(a) einfachen Darstellung (b) kompakte Darstellung

Abbildung 4.3: Zweistufige Partitionierung

4.2.2 Partitionierung der Abhängigkeitsvektoren

Für die Transformation eines Algorithmus von einem n-dimensionalen Iterationsraum in einen $(p+1)n$-dimensionalen Iterationsraum müssen auch die Abhängigkeitsvektoren $\mathbf{d}(e) \in \mathbb{Z}^n$ des Algorithmus in den $(p+1)n$-dimensionalen Raum transformiert werden.

Einstufig partitionierte Abhängigkeitsvektoren

Der Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{d} = \mathbf{i}_{\delta(e)} - \mathbf{i}_{\sigma(e)}$ im Indexraum $\mathcal{I}$ wird im Allgemeinen durch eine Partitionierung mit Partitionsmatrix $\boldsymbol{\Theta}$ in mehrere partitionierte Abhängigkeitsvektoren $\mathbf{d}_i^\kappa$ aufgeteilt:

$$\mathbf{d} = \mathbf{i}_{\delta(e)} - \mathbf{i}_{\sigma(e)}, \tag{4.25}$$

$$\mathbf{d} = \boldsymbol{\kappa}_\delta + \boldsymbol{\Theta}\widehat{\boldsymbol{\kappa}}_\delta + \boldsymbol{\vartheta}_0 - (\boldsymbol{\kappa}_\sigma + \boldsymbol{\Theta}\widehat{\boldsymbol{\kappa}}_\sigma + \boldsymbol{\vartheta}_0), \tag{4.26}$$

$$\mathbf{d} = \boldsymbol{\kappa}_\delta - \boldsymbol{\kappa}_\sigma + \boldsymbol{\Theta}\widehat{\boldsymbol{\kappa}}_\delta, \qquad \boldsymbol{\kappa}_\sigma \in \mathcal{K}_i^{\sigma(\mathbf{d})}, \tag{4.27}$$

$$\mathbf{d} = \mathbf{d}_i^{\boldsymbol{\Theta}} + \boldsymbol{\Theta}\widehat{\mathbf{d}}_i^{\boldsymbol{\Theta}}. \tag{4.28}$$

$$\mathbf{d} = \begin{pmatrix}\mathbf{E} & \boldsymbol{\Theta}\end{pmatrix} \mathbf{d}_i^\kappa, \qquad \mathbf{d}_i^\kappa = \begin{pmatrix}\mathbf{d}_i^{\boldsymbol{\Theta}} \\ \widehat{\mathbf{d}}_i^{\boldsymbol{\Theta}}\end{pmatrix} \in \mathcal{D}^\kappa, \tag{4.29}$$

wobei o. B. d. A. $\widehat{\boldsymbol{\kappa}}_\sigma = \mathbf{0}$ gesetzt wurde. Der innere Teilvektor $\mathbf{d}_i^{\boldsymbol{\Theta}}$ beschreibt die Abhängigkeiten zwischen den Elementen in einer Partition und der äußere Teilvektor $\widehat{\mathbf{d}}_i^{\boldsymbol{\Theta}}$ die Abhängigkeiten zwischen den Partitionen. Die partitionierten Abhängigkeitsvektoren $\mathbf{d}_i^\kappa = (\mathbf{d}_i^{\boldsymbol{\Theta}}, \widehat{\mathbf{d}}_i^{\boldsymbol{\Theta}})^t \in \mathcal{D}^\kappa$ sind abhängig von dem Ausgangselement $\boldsymbol{\kappa}_\sigma \in \mathcal{K}_i^{\sigma(\mathbf{d})}$ in der Partition $\mathcal{K}$.

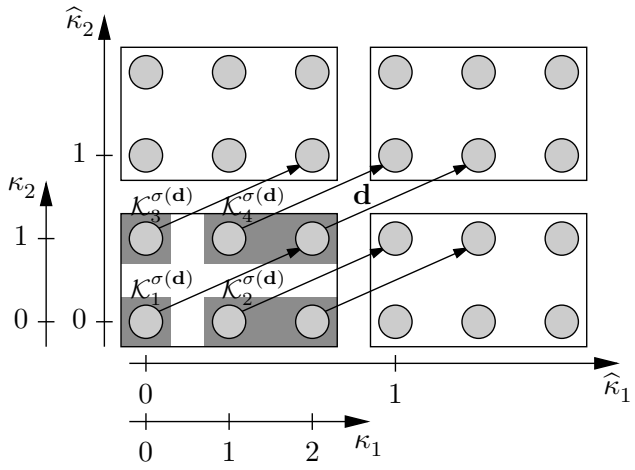
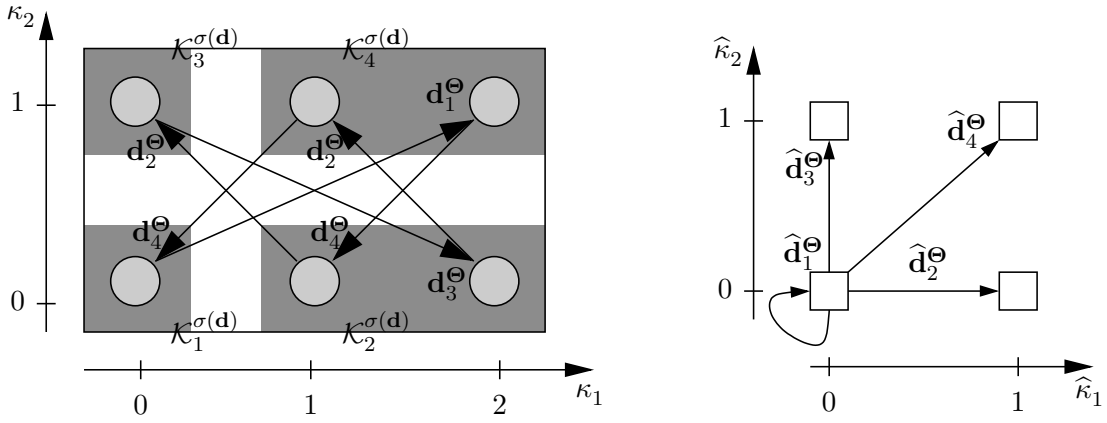


Abbildung 4.4: Abhängigkeitsvektor $\mathbf{d} = (2,\ 1)^t$ im partitionierten Iterationsraum



(a) in der Partition (b) zwischen Partitionen

Abbildung 4.5: Abhängigkeitsvektor $\mathbf{d} = (2,\ 1)^t$ im partitionierten Iterationsraum

Beispiel 4.4 (Einstufige Partitionierung eines Abhängigkeitsvektors)
Abbildung 4.4 zeigt den Abhängigkeitsvektor $\mathbf{d} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *in einem partitionierten Iterationsraum mit* $\mathbf{\Theta} = \left(\begin{smallmatrix}3 & 0\\0 & 2\end{smallmatrix}\right)$. *Vier Teilbereiche* $\mathcal{K}_1^{\sigma(\mathbf{d})}, \ldots, \mathcal{K}_4^{\sigma(\mathbf{d})}$ *müssen nach der Partitionierung unterschieden werden. Die Unterscheidung erfolgt nach der Lage des Zielknotens* $\boldsymbol{\kappa}_\delta$. *Die Startknoten* $\boldsymbol{\kappa}_\sigma$, *deren Zielknoten* $\boldsymbol{\kappa}_\delta$ *sich in der gleichen Partition befinden, werden zu einem Teilbereich* $\mathcal{K}_i^{\sigma(\mathbf{d})}$ *zusammengefasst.*

In Abbildung 4.5 sind die inneren und äußeren Teilvektoren des partitionierten Abhängigkeitsvektors dargestellt, welche zwischen den Elementen einer Partition (a) bzw. zwischen den Partitionen (b) existieren.

Zur Bestimmung der partitionierten Abhängigkeitsvektoren $\mathbf{d}_i^{\kappa} = (\mathbf{d}_i^{\mathbf{\Theta}}\ \widehat{\mathbf{d}}_i^{\mathbf{\Theta}})^t$ und deren zugehöriger Teilbereiche $\mathcal{K}_i^{\sigma(\mathbf{d})}$ bzw. $\mathcal{K}_i^{\delta(\mathbf{d})}$, welche wir im Weiteren als Quellen- bzw. Senkenteilräume bezeichnen, aus dem Abhängigkeitsvektor $\mathbf{d} \in \mathbb{Z}^n$ kann wie folgt vorgegangen werden:

(1) Bestimmung der Mengen $\mathcal{X}^1(\mathbf{d})$ und $\mathcal{X}^2(\mathbf{d})$, welche die Indizes j der Elemente d_j des Abhängigkeitsvektors $\mathbf{d}$ enthalten sollen, bei denen das Vektorelement d_j in dieser Raumrichtung in eine bzw. zwei Partitionen zeigt.

$$\mathcal{X}^1(\mathbf{d}) = \left\{ j \mid d_j \mod \vartheta_j = 0 \right\} \tag{4.30}$$

$$\mathcal{X}^2(\mathbf{d}) = \left\{ j \mid d_j \mod \vartheta_j \neq 0 \right\} \tag{4.31}$$

Nur wenn das Vektorelement d_j gleich der Größe ϑ_j der Partition in dieser Raumrichtung oder ein Vielfaches davon ist, zeigen alle Abhängigkeitsvektoren dieser Raumrichtung in **eine** Partition. Für die Mengen $\mathcal{X}^1(\mathbf{d})$ und $\mathcal{X}^2(\mathbf{d})$ gilt dann:

$$\mathcal{X}(\mathbf{d}) = \mathcal{X}^1(\mathbf{d}) \cup \mathcal{X}^2(\mathbf{d}) = \{ j \mid 1 \leq j \leq n \} \quad \text{und} \quad \mathcal{X}^1(\mathbf{d}) \cap \mathcal{X}^2(\mathbf{d}) = \emptyset. \tag{4.32}$$

Die Anzahl der partitionierten Abhängigkeitsvektoren, die aus dem Abhängigkeitsvektor $\mathbf{d}$ entstehen, lässt sich bestimmen mit: $m = 2^{|\mathcal{X}^2|}$.

(2) Bestimmung der Menge $\widehat{\mathcal{D}}^{\mathbf{d}}$ von äußeren Teilvektoren $\widehat{\mathbf{d}}^{\boldsymbol{\Theta}}$:

$$\widehat{\mathcal{D}}^{\mathbf{d}} = \left\{ \widehat{\mathbf{d}}^{\boldsymbol{\Theta}} \mid \widehat{\mathbf{d}}^{\boldsymbol{\Theta}} = \left(\widehat{d}_1^{\boldsymbol{\Theta}}, \ldots, \widehat{d}_j^{\boldsymbol{\Theta}}, \ldots, \widehat{d}_n^{\boldsymbol{\Theta}} \right)^t \right\}, \tag{4.33}$$

$$\widehat{d}_j^{\boldsymbol{\Theta}} \in \left\{ \widehat{d}_j^{1,\boldsymbol{\Theta}} \right\}, \quad \text{wenn } j \in \mathcal{X}^1(\mathbf{d}), \tag{4.34}$$

$$\widehat{d}_j^{\boldsymbol{\Theta}} \in \left\{ \widehat{d}_j^{1,\boldsymbol{\Theta}}, \widehat{d}_j^{2,\boldsymbol{\Theta}} \right\}, \quad \text{wenn } j \in \mathcal{X}^2(\mathbf{d}), \tag{4.35}$$

mit

$$\widehat{d}_j^{1,\boldsymbol{\Theta}} = \left\lfloor \frac{d_j}{\vartheta_j} \right\rfloor, \qquad \widehat{d}_j^{2,\boldsymbol{\Theta}} = \left\lceil \frac{d_j}{\vartheta_j} \right\rceil. \tag{4.36}$$

(3) Bestimmung der Menge $\mathcal{D}^{\mathbf{d}}$ von inneren Teilvektoren $\mathbf{d}_i^{\boldsymbol{\Theta}}$ unter Verwendung der äußeren Teilvektoren $\widehat{\mathbf{d}}_i^{\boldsymbol{\Theta}}$:

$$\mathcal{D}^{\mathbf{d}} = \left\{ \mathbf{d}^{\boldsymbol{\Theta}} \mid \mathbf{d}^{\boldsymbol{\Theta}} = \mathbf{d} - \boldsymbol{\Theta}\widehat{\mathbf{d}}^{\boldsymbol{\Theta}} \wedge \widehat{\mathbf{d}}^{\boldsymbol{\Theta}} \in \widehat{\mathcal{D}}^{\mathbf{d}} \right\}. \tag{4.37}$$

(4) Bestimmung der Quellenteilräume $\mathcal{K}^{\sigma(\mathbf{d})}$ und Senkenteilräume $\mathcal{K}^{\delta(\mathbf{d})}$ mittels:

$$\mathcal{K}^{\sigma(\mathbf{d})} = \mathcal{K} \cap f(\mathcal{K}) \qquad \text{mit} \qquad f(\boldsymbol{\kappa}) = \mathbf{E}\boldsymbol{\kappa} - \mathbf{d}^{\boldsymbol{\Theta}}, \qquad \mathbf{d}^{\boldsymbol{\Theta}} \in \mathcal{D}^{\mathbf{d}}, \tag{4.38}$$

$$\mathcal{K}^{\delta(\mathbf{d})} = \mathcal{K} \cap f(\mathcal{K}) \qquad \text{mit} \qquad f(\boldsymbol{\kappa}) = \mathbf{E}\boldsymbol{\kappa} + \mathbf{d}^{\boldsymbol{\Theta}}, \qquad \mathbf{d}^{\boldsymbol{\Theta}} \in \mathcal{D}^{\mathbf{d}}, \tag{4.39}$$

wobei $f(\mathcal{K})$ der mittels der Funktion $f(\boldsymbol{\kappa})$, $\boldsymbol{\kappa} \in \mathcal{K}$ transformierte Raum ist.

Da der Raum $\mathcal{K}$ ein Hyperquader mit $\mathbf{a}_0^{\sqcup} = \mathbf{0}$ und $\mathbf{a}_0^{\sqcap} = \boldsymbol{\vartheta} - \mathbf{1}$ ist, können die unteren und oberen Grenzen der Hyperquader $\mathcal{K}^{\sigma(\mathbf{d})}$ und $\mathcal{K}^{\delta(\mathbf{d})}$ direkt bestimmt werden:

$$\mathbf{a}_0^{\sigma(\mathbf{d}),\sqcup} = \max\left(\mathbf{0},\ -\mathbf{d}^{\boldsymbol{\Theta}}\right), \qquad \mathbf{a}_0^{\sigma(\mathbf{d}),\sqcap} = \min\left(\boldsymbol{\vartheta} - \mathbf{1},\ \boldsymbol{\vartheta} - \mathbf{1} - \mathbf{d}^{\boldsymbol{\Theta}}\right), \tag{4.40}$$

$$\mathbf{a}_0^{\delta(\mathbf{d}),\sqcup} = \max\left(\mathbf{0},\ \mathbf{d}^{\boldsymbol{\Theta}}\right), \qquad \mathbf{a}_0^{\delta(\mathbf{d}),\sqcap} = \min\left(\boldsymbol{\vartheta} - \mathbf{1},\ \boldsymbol{\vartheta} - \mathbf{1} + \mathbf{d}^{\boldsymbol{\Theta}}\right). \tag{4.41}$$

Die Elemente eines Quellenteilraums $\mathcal{K}^{\sigma(\mathbf{d})}$ sind die Quellen des Abhängigkeitsvektors $\mathbf{d}$, deren Ziel in der selben Partition liegt. Die Elemente eines Senkenteilraums $\mathcal{K}^{\delta(\mathbf{d})}$ sind die Ziele des Abhängigkeitsvektors $\mathbf{d}$, deren Quelle in der selben Partition liegt. Für die Teilräume $\mathcal{K}_i^{\sigma(\mathbf{d})}$ bzw. $\mathcal{K}_i^{\delta(\mathbf{d})}$, mit $1 \leq i \leq m$ gilt:

$$\mathcal{K} = \bigcup_{1 \leq i \leq m} \mathcal{K}_i^{\sigma(\mathbf{d})} \qquad \emptyset = \mathcal{K}_i^{\sigma(\mathbf{d})} \cap \mathcal{K}_j^{\sigma(\mathbf{d})}, \quad 1 \leq i, j \leq m,\ i \neq j, \tag{4.42}$$

$$\mathcal{K} = \bigcup_{1 \leq i \leq m} \mathcal{K}_i^{\delta(\mathbf{d})} \qquad \emptyset = \mathcal{K}_i^{\delta(\mathbf{d})} \cap \mathcal{K}_j^{\delta(\mathbf{d})}, \quad 1 \leq i, j \leq m,\ i \neq j. \tag{4.43}$$

Alle partitionierten Abhängigkeitsvektoren $\mathbf{d}_i^{\kappa}$, die aus dem Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{d}$ entstehen, werden in der Menge $\mathcal{D}^{\kappa}(e)$ zusammengefasst.

Beispiel 4.5 (Teilräume des einstufig partitionierten Abhängigkeitsvektors) *Für den Abhängigkeitsvektor* $\mathbf{d} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *aus Beispiel 4.4 ergeben sich folgende Teilvektoren* $\mathbf{d}_i^{\boldsymbol{\Theta}}$ *und* $\widehat{\mathbf{d}}_i^{\boldsymbol{\Theta}}$ *sowie Teilräume* $\mathcal{K}_i^{\sigma(\mathbf{d})}$ *und* $\mathcal{K}_i^{\delta(\mathbf{d})}$ *bei Partitionierung mit* $\boldsymbol{\Theta} = \left(\begin{smallmatrix}3&0\\0&2\end{smallmatrix}\right)$*:*

$$\mathbf{d}_1^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right) \quad \widehat{\mathbf{d}}_1^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right) \quad \mathcal{K}_1^{\sigma(\mathbf{d})} = \left\{\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right\} \quad \mathcal{K}_1^{\delta(\mathbf{d})} = \left\{\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)\right\}, \tag{4.44}$$

$$\mathbf{d}_2^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right) \quad \widehat{\mathbf{d}}_2^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right) \quad \mathcal{K}_2^{\sigma(\mathbf{d})} = \left\{\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\right\} \quad \mathcal{K}_2^{\delta(\mathbf{d})} = \left\{\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\right\}, \tag{4.45}$$

$$\mathbf{d}_3^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}2\\-1\end{smallmatrix}\right) \quad \widehat{\mathbf{d}}_3^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right) \quad \mathcal{K}_3^{\sigma(\mathbf{d})} = \left\{\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right\} \quad \mathcal{K}_3^{\delta(\mathbf{d})} = \left\{\left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\right\}, \tag{4.46}$$

$$\mathbf{d}_4^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right) \quad \widehat{\mathbf{d}}_4^{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right) \quad \mathcal{K}_4^{\sigma(\mathbf{d})} = \left\{\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)\right\} \quad \mathcal{K}_4^{\delta(\mathbf{d})} = \left\{\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\right\}. \tag{4.47}$$

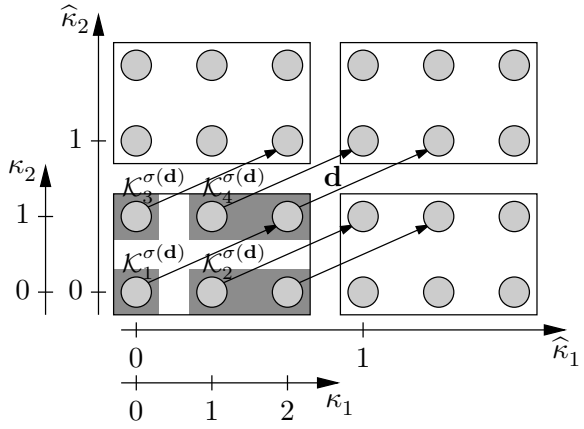
Die inneren Teilvektoren $\mathbf{d}_i^{\boldsymbol{\Theta}}$ *sind in Abbildung 4.5(a) und die äußeren Teilvektoren* $\widehat{\mathbf{d}}_i^{\boldsymbol{\Theta}}$ *in Abbildung 4.5(b) dargestellt. Abbildung 4.6 zeigt die Quellen- und Senkenteilräume für unser Beispiel. Die Teilräume existieren in allen Partitionen. Zur besseren Veranschaulichung wurde jeder Teilraum nur einmal dargestellt.*

Mehrstufig partitionierte Abhängigkeitsvektoren

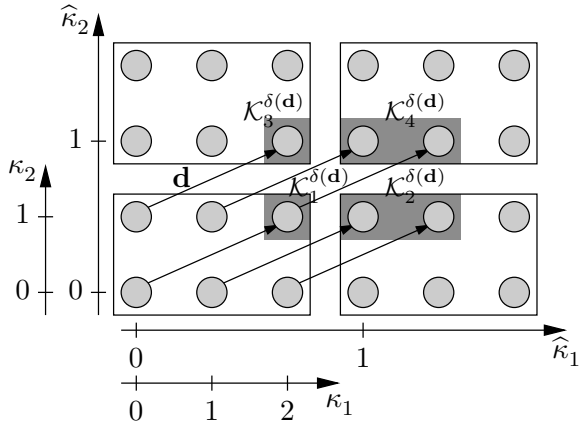
Bei der mehrstufigen Partitionierung müssen die äußeren Teilvektoren $\widehat{\mathbf{d}}_{i_1,\ldots,i_{k-1}}^{\boldsymbol{\Theta}^{k-1}}$ der $(k-1)$-ten Partitionierung in der nachfolgenden k-ten Partitionierungsstufe erneut partitioniert werden. Jeder äußere Teilvektor $\widehat{\mathbf{d}}_{i_1,\ldots,i_{k-1}}^{\boldsymbol{\Theta}^{k-1}}$ zerfällt wieder in m innere Teilvektoren $\mathbf{d}_{i_1,\ldots,i_k}^{\boldsymbol{\Theta}^k}$ und m äußere Teilvektoren $\widehat{\mathbf{d}}_{i_1,\ldots,i_k}^{\boldsymbol{\Theta}^k}$. Die partitionierten Abhängigkeitsvektoren

$$\mathbf{d}_{i_1,i_2,\ldots,i_p}^{\kappa} = \left(\mathbf{d}_{i_1}^{\boldsymbol{\Theta}^1}, \mathbf{d}_{i_1,i_2}^{\boldsymbol{\Theta}^2}, \ldots, \mathbf{d}_{i_1,i_2,\ldots,i_p}^{\boldsymbol{\Theta}^p}, \widehat{\mathbf{d}}_{i_1,i_2,\ldots,i_p}^{\boldsymbol{\Theta}^p}\right)^t \in \mathcal{D}^{\kappa}(e)$$
$$\text{mit } \boldsymbol{\kappa}_{\sigma}^1 \in \mathcal{K}_{i_1}^{1,\sigma(\mathbf{d})}, \boldsymbol{\kappa}_{\sigma}^2 \in \mathcal{K}_{i_1,i_2}^{2,\sigma(\mathbf{d})}, \ldots, \boldsymbol{\kappa}_{\sigma}^p \in \mathcal{K}_{i_1,i_2,\ldots,i_p}^{p,\sigma(\mathbf{d})} \tag{4.48}$$

ergeben sich aus den p inneren Teilvektoren $\mathbf{d}_{i_1}^{\boldsymbol{\Theta}^1}, \mathbf{d}_{i_1,i_2}^{\boldsymbol{\Theta}^2}, \ldots, \mathbf{d}_{i_1,i_2,\ldots,i_p}^{\boldsymbol{\Theta}^p}$ einer jeden Partitionierungsstufe und dem äußeren Teilvektor $\widehat{\mathbf{d}}_{i_1,i_2,\ldots,i_p}^{\boldsymbol{\Theta}^p}$ der p-ten Partitionierungsstufe. Alle partitionieren Abhängigkeitsvektoren $\mathbf{d}_{i_1,\ldots,i_{2p}}^{\kappa}$, welche aus einem Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{d}$ resultieren, sind in der Menge $\mathcal{D}^{\kappa}(e)$ zusammengefasst. Die Anzahl der



(a) Quellenteilräume $\mathcal{K}^{\sigma(\mathbf{d})}$



(b) Senkenteilräume $\mathcal{K}^{\delta(\mathbf{d})}$

Abbildung 4.6: Teilräume in einer Partition

partitionierten Abhängigkeitsvektoren $|\mathcal{D}^{\kappa}(e)|$ ist abhängig von der Anzahl an Partitionierungsstufen p, der Größe der Partitionen und der Dimension n des Abhängigkeitsvektors $\mathbf{d}(e) \in \mathbb{Z}^n$. Im ungünstigsten Fall erhalten wir $|\mathcal{D}^{\kappa}(e)|_{\max} = 2^{p \cdot n}$ partitionierte Abhängigkeitsvektoren aus einem Abhängigkeitsvektor $\mathbf{d}(e)$.

Beispiel 4.6 (Zweistufige Partitionierung eines Abhängigkeitsvektors)
Der Abhängigkeitsvektor $\mathbf{d} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *soll zweifach partitioniert werden. Die Partitionsmatrix der ersten Stufe lautet* $\mathbf{\Theta}^1 = \left(\begin{smallmatrix}3&0\\0&2\end{smallmatrix}\right)$ *und die der zweiten Stufe* $\mathbf{\Theta}^2 = \left(\begin{smallmatrix}4&0\\0&5\end{smallmatrix}\right)$*. Die erste Partitionierung führt zu vier partitionierten Abhängigkeitsvektoren, wie in Beispiel 4.4 gezeigt. Die äußeren Teilvektoren* $\widehat{\mathbf{d}}_1^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$, $\widehat{\mathbf{d}}_2^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$, $\widehat{\mathbf{d}}_3^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *und* $\widehat{\mathbf{d}}_4^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *werden nun unter Verwendung der Partitionsmatrix* $\mathbf{\Theta}^2 = \left(\begin{smallmatrix}4&0\\0&5\end{smallmatrix}\right)$ *erneut partitioniert. Hieraus ergeben sich die folgenden Teilvektoren* $\mathbf{d}_{i_1,i_2}^{\mathbf{\Theta}^2}$ *und* $\widehat{\mathbf{d}}_{i_1,i_2}^{\mathbf{\Theta}^2}$ *für die zweite Partitionierung, abhängig von den äußeren Teilvektoren* $\widehat{\mathbf{d}}_{i_1}^{\mathbf{\Theta}^1}$ *der ersten Partitionierung:*

$$\widehat{\mathbf{d}}_1^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right): \quad \mathbf{d}_{1,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{1,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \tag{4.49}$$

$$\widehat{\mathbf{d}}_2^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right): \quad \mathbf{d}_{2,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{2,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \quad \mathbf{d}_{2,2}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}-3\\0\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{2,2}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \tag{4.50}$$

$$\widehat{\mathbf{d}}_3^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right): \quad \mathbf{d}_{3,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{3,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \quad \mathbf{d}_{3,2}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\-4\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{3,2}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right), \tag{4.51}$$

$$\widehat{\mathbf{d}}_4^{\mathbf{\Theta}^1} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right): \quad \mathbf{d}_{3,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{3,1}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \quad \mathbf{d}_{3,2}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}-3\\1\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{3,2}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right),$$

$$\mathbf{d}_{3,3}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}1\\-4\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{3,3}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right), \quad \mathbf{d}_{3,4}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}-3\\-4\end{smallmatrix}\right), \quad \widehat{\mathbf{d}}_{3,4}^{\mathbf{\Theta}^2} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right). \tag{4.52}$$

Somit erhalten wir neun partitionierte Abhängigkeitsvektoren aus dem Abhängigkeitsvektor $\mathbf{d} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$*. Diese lauten:*

$$\begin{aligned}
\mathbf{d}_{1,1}^{\kappa} &= \left(\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)^t, \\
\mathbf{d}_{2,1}^{\kappa} &= \left(\left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)^t, & \mathbf{d}_{2,2}^{\kappa} &= \left(\left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}-3\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\right)^t, \\
\mathbf{d}_{3,1}^{\kappa} &= \left(\left(\begin{smallmatrix}2\\-1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)^t, & \mathbf{d}_{3,2}^{\kappa} &= \left(\left(\begin{smallmatrix}2\\-1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\-4\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right)^t, \\
\mathbf{d}_{4,1}^{\kappa} &= \left(\left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)^t, & \mathbf{d}_{4,2}^{\kappa} &= \left(\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right), \left(\begin{smallmatrix}-3\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\right)^t, \\
\mathbf{d}_{4,3}^{\kappa} &= \left(\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\-4\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right)^t, & \mathbf{d}_{4,4}^{\kappa} &= \left(\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right), \left(\begin{smallmatrix}-3\\-4\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\right)^t.
\end{aligned}$$

Abbildung 4.7 zeigt die neun unterschiedlichen partitionierten Abhängigkeitsvektoren für die Partition $\widehat{\boldsymbol{\kappa}}^2 = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$*, dabei wurden die partitionierten Abhängigkeitsvektoren in verschiedenen Farben und Strichformen gezeichnet.*

Die Quellenteilräume $\mathcal{K}_{i_1}^{1,\sigma(\mathbf{d})}$ *der ersten Partitionierungstufe sind in Beispiel 4.4 angegeben. Die Quellenteilräume* $\mathcal{K}_{i_1,i_2}^{2,\sigma(\mathbf{d})}$ *der zweiten Partitionierungsstufe betragen:*

$$\begin{aligned}
\mathcal{K}_{1,1}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | 0 \le \kappa_1^2 < 4 \wedge 0 \le \kappa_2^2 < 5\right\}, \\
\mathcal{K}_{2,1}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | 0 \le \kappa_1^2 < 3 \wedge 0 \le \kappa_2^2 < 5\right\}, & \mathcal{K}_{2,2}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | \kappa_1^2 = 3 \wedge 0 \le \kappa_2^2 < 5\right\}, \\
\mathcal{K}_{3,1}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | 0 \le \kappa_1^2 < 4 \wedge 0 \le \kappa_2^2 < 4\right\}, & \mathcal{K}_{3,2}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | 0 \le \kappa_1^2 < 4 \wedge \kappa_2^2 = 4\right\}, \\
\mathcal{K}_{4,1}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | 0 \le \kappa_1^2 < 3 \wedge 0 \le \kappa_2^2 < 4\right\}, & \mathcal{K}_{4,2}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | \kappa_1^2 = 3 \wedge 0 \le \kappa_2^2 < 4\right\}, \\
\mathcal{K}_{4,3}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | 0 \le \kappa_1^2 < 3 \wedge \kappa_2^2 = 4\right\}, & \mathcal{K}_{4,4}^{2,\sigma(\mathbf{d})} &= \left\{\boldsymbol{\kappa}^2 | \kappa_1^2 = 3 \wedge \kappa_2^2 = 4\right\}.
\end{aligned}$$

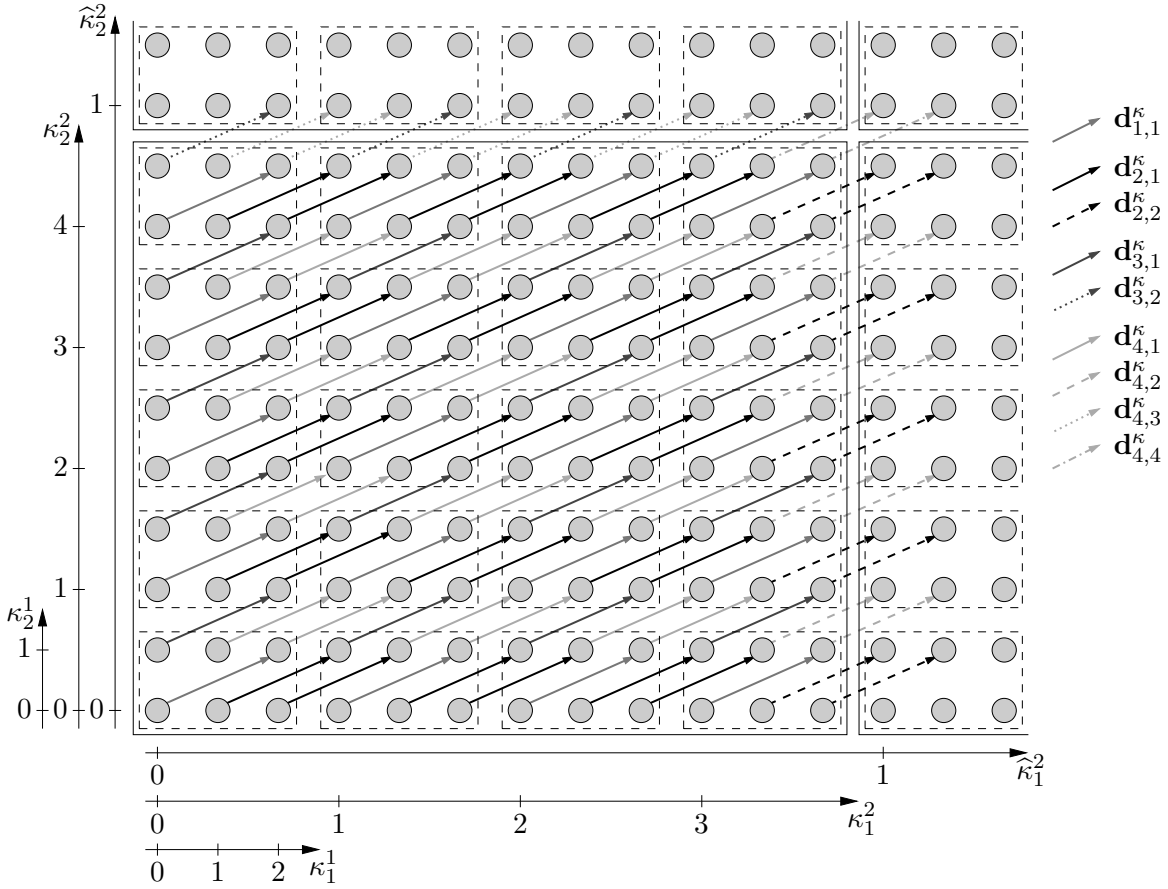


Abbildung 4.7: Abhängigkeitsvektor $\mathbf{d} = (2,\ 1)^t$ bei zweifacher Partitionierung

Die aus einem Abhängigkeitsvektor $\mathbf{d}$ entstehenden partitionierten Abhängigkeitsvektoren können als Baum dargestellt werden. Diesen Baum bezeichnen wir als *Baum der partitionierten Abhängigkeitsvektoren.*

Definition 4.3 (Baum der partitionierten Abhängigkeitsvektoren (BPA))
Der Baum[1] *der partitionierten Abhängigkeitsvektoren (kurz: BPA) ist ein gerichteter Graph* $G = (\mathcal{V}, \mathcal{E})$*, welcher alle aus dem Abhängigkeitsvektor* $\mathbf{d}(e) = \mathbf{d}$ *entstandenen partitionierten Teilvektoren* $\mathbf{d}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}$ *bzw.* $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}_{i_1,i_2,\ldots,i_k}$ *enthält. Die Wurzel des BPA ist der Abhängigkeitsvektor* $\mathbf{d}$*. Die anderen Knoten beinhalten jeweils ein Vektorenpaar* $(\mathbf{d}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k})$*, bestehend aus dem inneren Teilvektor* $\mathbf{d}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}$ *und dem zugehörigen äußeren Teilvektor* $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}$*.*

Die Kindknoten eines Knotens v *geben alle Vektorenpaare* $(\mathbf{d}^{\mathbf{\Theta}^{k+1}}_{i_1,\ldots,i_k,i_{k+1}}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^{k+1}}_{i_1,\ldots,i_k,i_{k+1}})$ *an, welche durch die Partitionierung des Abhängigkeitsvektors* $\mathbf{d}$*, wenn* v *der Wurzelknoten* $(k = 0)$ *ist, oder eines äußeren Teilvektors* $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}$*, wenn* v *ein innerer Knoten* $(k > 0)$ *ist, unter Verwendung der Partitionsmatrix* $\mathbf{\Theta}^{k+1}$ *entstehen.*

Die Kanten $e \in \mathcal{E}$ *des Baums sind gewichtet. Das Gewicht* $n(e)$ *gibt die Anzahl der Elemente der Partition* $\mathbf{\Theta}^k$ *an, dessen Abhängigkeitsvektor* $\mathbf{d}$ $(k = 1)$ *bzw. äußerer Teilvektor* $\widehat{\mathbf{d}}^{\mathbf{\Theta}^{k-1}}_{i_1,\ldots,i_{k-1}}$ $(k > 1)$ *in die Teilvektoren* $\mathbf{d}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_{k-1},i_k}$ *und* $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_{k-1},i_k}$ *zerfällt. Die Quelle* $\sigma(e)$ *der Kante* e *ist* $\sigma(e) = \mathbf{d}$ *für* $k = 1$ *bzw.* $\sigma(e) = \left(\mathbf{d}^{\mathbf{\Theta}^{k-1}}_{i_1,\ldots,i_{k-1}}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^{k-1}}_{i_1,\ldots,i_{k-1}}\right)$ *für* $k > 1$*. Das Ziel* $\delta(e)$ *der Kante* e *ist* $\delta(e) = \left(\mathbf{d}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_{k-1},i_k}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_{k-1},i_k}\right)$*.*

Das Kantengewicht $n(e)$ ist gleich der Anzahl der Elemente des Quellen- bzw. des Senkenteilraums $n(e) = |\mathcal{K}^{k,\sigma(\mathbf{d})}_{i_1,\ldots,i_k}| = |\mathcal{K}^{k,\delta(\mathbf{d})}_{i_1,\ldots,i_k}|$.

Die Summe der Kantengewichte $\sum n(e)$ aller Kanten, welche von einem Knoten $\sigma(e) = v$ ausgehen, ist gleich der Anzahl der Elemente der Partition $\det(\mathbf{\Theta}^{k+1})$, mit der der Abhängigkeitsvektor $\mathbf{d}$ oder der äußere Teilvektor $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}$ partitioniert wurde:

$$\sum_{\substack{\forall e \in \mathcal{E} \\ \sigma(e) = v}} n(e) = \det(\mathbf{\Theta}^{k+1}), \qquad d(\sigma(e)) = d(v) = k. \tag{4.53}$$

Die Tiefe $d(v) = k$ des Knotens $v \in \mathcal{V}$ gibt die Partitionierungsstufe an. Die Tiefe $d(G) = p$ des Baums G ist gleich der Anzahl an Partitionierungsstufen p.

Die partitionierten Abhängigkeitsvektoren

$$\mathbf{d}^{\kappa}_{i_1,\ldots,i_p} = \left(\mathbf{d}^{\mathbf{\Theta}^1}_{i_1}, \mathbf{d}^{\mathbf{\Theta}^2}_{i_1,i_2}, \ldots, \mathbf{d}^{\mathbf{\Theta}^p}_{i_1,i_2,\ldots,i_p}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^p}_{i_1,i_2,\ldots,i_p}\right)^t \in \mathcal{D}^{\kappa}(e)$$

erhält man, indem man alle Wege

$$p_{i_1,\ldots,i_p} = \mathbf{d} \rightarrow \ldots \rightarrow \left(\mathbf{d}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}\right) \rightarrow \ldots \rightarrow \left(\mathbf{d}^{\mathbf{\Theta}^p}_{i_1,\ldots,i_p}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^p}_{i_1,\ldots,i_p}\right) \tag{4.54}$$

[1] Die graphentheoretische Definition eines Baums und der zugehörigen Bezeichnungen sind in Anhang B zu finden.

von der Wurzel des Baums $\sigma(p) = \mathbf{d}$ zu den Blätter $\delta(p) = \left(\mathbf{d}^{\mathbf{\Theta}^p}_{i_1,\ldots,i_p}, \widehat{\mathbf{d}}^{\mathbf{\Theta}^p}_{i_1,\ldots,i_p}\right)$ bestimmt. Die inneren Teilvektoren $\mathbf{d}^{\mathbf{\Theta}^k}_{i_1,\ldots,i_k}$ der Knoten entlang eines jeden Weges und der äußere Teilvektor $\widehat{\mathbf{d}}^{\mathbf{\Theta}^p}_{i_1,\ldots,i_p}$ des Blattes ergeben jeweils einen partitionierten Abhängigkeitsvektor $\mathbf{d}^{\boldsymbol{\kappa}}_{i_1,\ldots,i_p}$.

Es existiert jeweils nur ein Weg von der Wurzel des Baums zu einem Blatt. Somit ist die Anzahl $|\mathcal{D}^{\kappa}(e)|$ der partitionierten Abhängigkeitsvektoren gleich der Anzahl der Blätter des Baums.

Das Produkt der Kantengewichte $\prod_{\forall e \in p_{i_1,\ldots,i_p}} n(e)$ des Weges $p_{i_1,\ldots,i_p}$ gibt die Anzahl der Elemente $\mathbf{i}^{\boldsymbol{\kappa}} = (\boldsymbol{\kappa}^1, \boldsymbol{\kappa}^2, \ldots, \boldsymbol{\kappa}^p, \widehat{\boldsymbol{\kappa}}^p)$ einer p-stufigen Partition $\widehat{\boldsymbol{\kappa}}^P$ an, dessen Abhängigkeitsvektor $\mathbf{d}$ in den partitionierten Abhängigkeitsvektor $\mathbf{d}^{\boldsymbol{\kappa}}_{i_1,\ldots,i_p}$ zerfallen ist.

Beispiel 4.7 (BPA eines zweistufig partitionierten Abhängigkeitsvektors) *Abbildung 4.8 zeigt den BPA für die zweifache Partitionierung des Abhängigkeitsvektors* $\mathbf{d} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *mit den Partitionierungsmatrizen* $\mathbf{\Theta}^1 = \left(\begin{smallmatrix}3 & 0\\0 & 2\end{smallmatrix}\right)$ *und* $\mathbf{\Theta}^2 = \left(\begin{smallmatrix}4 & 0\\0 & 5\end{smallmatrix}\right)$*. Die resultierenden neun partitionierten Abhängigkeitsvektoren* $\mathbf{d}^{\boldsymbol{\kappa}}_{i_1,i_2}$ *wurden bereits in Beispiel 4.6 angegeben.*

Am Beispiel des ersten und letzten Blattes des Baums soll die Herleitung der partitionierten Abhängigkeitsvektoren demonstriert werden. Der Weg p_1 *von der Wurzel zum ersten Blatt lautet* $p_{1,1} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right) \rightarrow \left(\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right) \rightarrow \left(\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)$*, somit ist* $\mathbf{d}^{\boldsymbol{\kappa}}_{1,1} = \left(\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)^t$ *der partitionierte Abhängigkeitsvektor. Für den Weg* p_9 *zum letzten Blatt erhalten wir* $p_{4,4} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right) \rightarrow \left(\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\right) \rightarrow \left(\left(\begin{smallmatrix}-3\\-4\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\right)$*. Der daraus resultierende Abhängigkeitsvektor lautet* $\mathbf{d}^{\boldsymbol{\kappa}}_{4,4} = \left(\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right), \left(\begin{smallmatrix}-3\\-3\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\right)^t$*.*

Aus dem Graphen ist weiterhin ersichtlich, dass der partitionierte Abhängigkeitsvektor $\mathbf{d}^{\boldsymbol{\kappa}}_{1,1}$ *zwanzigmal* $(1 \cdot 20)$ *innerhalb der zweistufigen Partition* $\widehat{\boldsymbol{\kappa}}^2$ *auftritt und der partitionierte Abhängigkeitsvektor* $\mathbf{d}^{\boldsymbol{\kappa}}_{4,4}$ *zweimal* $(2 \cdot 1)$ *innerhalb der zweistufigen Partition* $\widehat{\boldsymbol{\kappa}}^2$ *existiert. In Abbildung 4.7 kann dies überprüft werden. Der partitionierte Abhängigkeitsvektor* $\mathbf{d}^{\boldsymbol{\kappa}}_{1,1}$ *tritt in jeder der 20 Partitionen* $\boldsymbol{\kappa}^2$ *innerhalb der Partition* $\widehat{\boldsymbol{\kappa}}^2$ *einmal auf. Der partitionierte Abhängigkeitsvektor* $\mathbf{d}^{\boldsymbol{\kappa}}_{4,4}$ *existiert hingegen nur in der Partition* $\boldsymbol{\kappa}^2 = \left(\begin{smallmatrix}3\\4\end{smallmatrix}\right)$*. In dieser Partition gibt es zwei solche Vektoren, welche in die Partition* $\boldsymbol{\kappa}^2 = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *der Partition* $\widehat{\boldsymbol{\kappa}}^2 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *weisen.*

4.2.3 Partitionierung des Algorithmus

Nachdem in den vorherigen beiden Abschnitten die mehrstufige Partitionierung eines Iterationsraums und eines Abhängigkeitsvektors vorgestellt wurde, sollen nun diese Methoden zur mehrstufigen Partitionierung eines Algorithmus genutzt werden. Der Algorithmus muss hierfür ausschließlich lokale Berechnungszuweisungen besitzen, um die Komplexität der Transformation so gering als möglich zu halten. Die Transformation einer internen Berechnungszuweisung in eine lokale Berechnungszuweisung und in ein oder mehrere interne Interndatenzuweisungen ist in Abschnitt 3.3.3 vorgestellt worden.

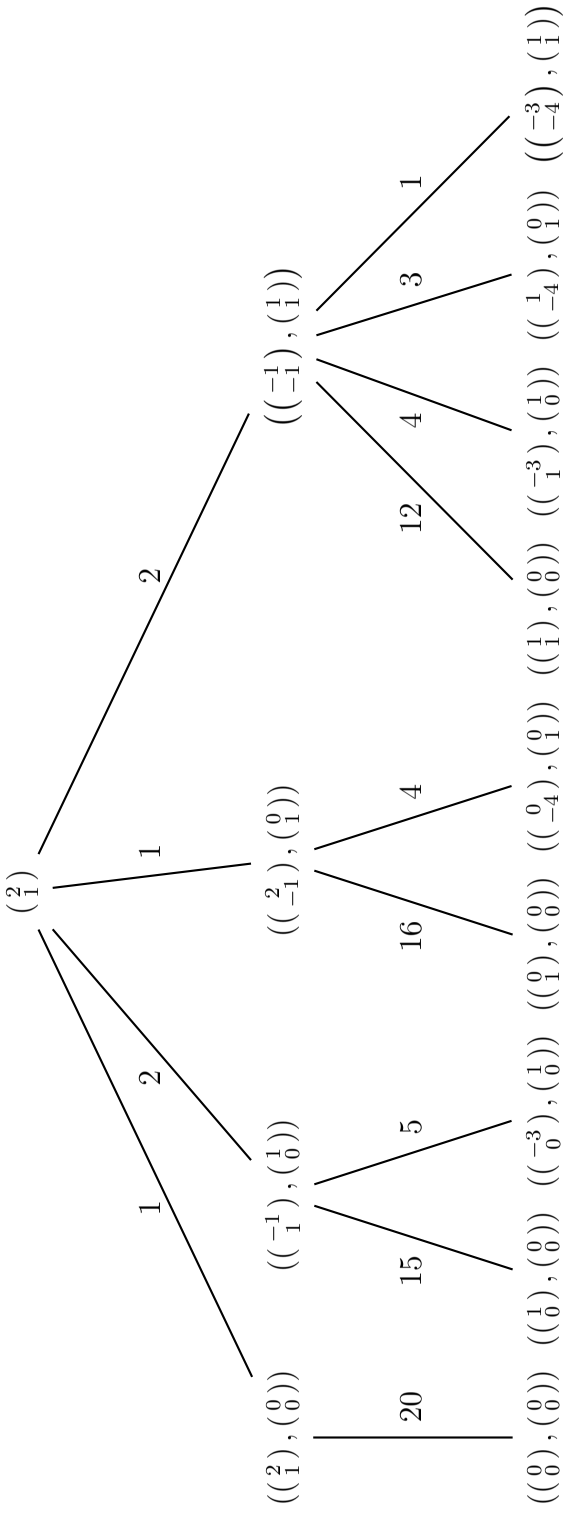


Abbildung 4.8: BPA des Abhängigkeitsvektor $\mathbf{d} = (2,\ 1)^t$ bei zweifacher Partitionierung

Der Algorithmus soll p-fach partitioniert werden. Die Größe der Partition einer jeden Partitionstufe l, $1 \leq l \leq p$, ist gegeben durch die Partitionsmatrix $\mathbf{\Theta}^l$. Bei der Transformation des Algorithmus sollen alle internen Variablen $y_s[.]$ des Ausgangsalgorithmus mit einem Indexraum $\mathcal{L}_s = \mathcal{I}_s \subset \mathbb{Z}^n$ in interne Variablen $y'_s[.]$ mit einem Indexraum $\mathcal{L}_s^\kappa = \mathcal{I}_s^\kappa \subset \mathbb{Z}^{(p+1)n}$ transformiert werden. Der Indexraum von globalen Variablen darf nicht verändert werden.

Im Folgenden soll nun die Transformation der Teilzuweisungen für die verschiedenen Teilzuweisungstypen vorgestellt werden. Wir beginnen mit der *Interndatenteilzuweisung*, siehe (3.28). Diese Teilzuweisung enthält nur lokale Variablen. Auf der rechten Seite der Zuweisung gibt es nur eine Variable $y_{s'}[.]$. Diese Variable besitzt einen Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{d}_{s',s_t}^{k_{s'}} = \mathbf{d}$. Die Partitionierung dieses Abhängigkeitsvektors führt zu einer Menge von partitionierten Abhängigkeitsvektoren $\mathbf{d}_{i_1,i_2,\ldots,i_p}^\kappa \in \mathcal{D}^\kappa(e)$. Für jeden partitionierten Abhängigkeitsvektoren $\mathbf{d}_{i_1,i_2,\ldots,i_p}^\kappa \in \mathcal{D}^\kappa(e)$ muss nun eine Interndatenteilzuweisung im partitionierten Iterationsraum definiert werden:

$$\begin{aligned} s_{t,(i_1,i_2,\ldots,i_p)}: \quad & y'_s[\mathbf{i}^\kappa] = y'_{s'}[\mathbf{i}^\kappa - \mathbf{d}_{i_1,i_2,\ldots,i_p}^\kappa], \\ & \mathbf{i}^\kappa \in \mathcal{I}_{s_{t,(i_1,i_2,\ldots,i_p)}}^\kappa = (\mathcal{K}_{i_1}^{1,\delta(\mathbf{d})} \oplus \mathcal{K}_{i_1,i_2}^{2,\delta(\mathbf{d})} \oplus \cdots \oplus \mathcal{K}_{i_1,i_2,\ldots,i_p}^{p,\delta(\mathbf{d})} \oplus \mathbb{Z}^n) \cap \mathcal{I}_{s_t}^\kappa, \end{aligned} \tag{4.55}$$

wobei $\mathcal{I}_{s_t}^\kappa$ der partitionierte Iterationsraum des Iterationsraums $\mathcal{I}_{s_t}$ ist. Der Iterationsraum $\mathcal{I}_{s_{t,(i_1,i_2,\ldots,i_p)}}^\kappa$ der partitionierten Teilzuweisung $s_{t_{i_1,i_2,\ldots,i_p}}$ ist die Schnittmenge der direkten Summe der Senkenteilräume $\mathcal{K}_{i_1}^{1,\delta(\mathbf{d})}$, $\mathcal{K}_{i_1,i_2}^{2,\delta(\mathbf{d})}$, $\ldots$, $\mathcal{K}_{i_1,i_2,\ldots,i_p}^{p,\delta(\mathbf{d})}$ und $\mathbb{Z}^n$ mit dem partitionierten Iterationsraum $\mathcal{I}_{s_t}^\kappa$.

Betrachtet man die Gleichungen zur Bestimmung des partitionierten Iterationsraums $\mathcal{I}_{s_t}^\kappa$ in (4.21) und der Senkenteilräume $\mathcal{K}_{i_1}^{1,\delta(\mathbf{d})}$, $\mathcal{K}_{i_1,i_2}^{2,\delta(\mathbf{d})}$, $\ldots$, $\mathcal{K}_{i_1,i_2,\ldots,i_p}^{p,\delta(\mathbf{d})}$ in (4.39) bzw. (4.41) etwas genauer, so stellt man fest, dass der Iterationsraum $\mathcal{I}_{s_{t,(i_1,i_2,\ldots,i_p)}}^\kappa = \{\mathbf{i}^\kappa \mid \mathbf{A}^\kappa \mathbf{i}^\kappa \geq \mathbf{a}_{0,(i_1,i_2,\ldots,i_p)}\}$ mit

$$\mathbf{A}^\kappa = \begin{pmatrix} \mathbf{E} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{E} & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & -\mathbf{E} & \mathbf{0} \\ \mathbf{A} & \mathbf{A}\mathbf{\Theta}^1 & \ldots & \mathbf{A}\prod_{k=1}^{p-1}\mathbf{\Theta}^k & \mathbf{A}\prod_{k=1}^{p}\mathbf{\Theta}^k \end{pmatrix} \tag{4.56}$$

und

$$\begin{aligned} \mathbf{a}_{0,(i_1,i_2,\ldots,i_p)} = \Big(& \mathbf{a}_{0,(i_1)}^{1,\delta(\mathbf{d}),\sqcup} \; - \mathbf{a}_{0,(i_1)}^{1,\delta(\mathbf{d}),\sqcap} \; \mathbf{a}_{0,(i_1,i_2)}^{2,\delta(\mathbf{d}),\sqcup} \; - \mathbf{a}_{0,(i_1,i_2)}^{2,\delta(\mathbf{d}),\sqcap} \; \ldots \\ & \ldots \; \mathbf{a}_{0,(i_1,i_2,\ldots,i_p)}^{p,\delta(\mathbf{d}),\sqcup} \; - \mathbf{a}_{0,(i_1,i_2,\ldots,i_p)}^{p,\delta(\mathbf{d}),\sqcap} \; \mathbf{a}_0 \Big)^t \end{aligned} \tag{4.57}$$

bestimmt werden kann.

Einen Sonderfall stellt der Abhängigkeitsvektor $\mathbf{d} = \mathbf{0}$ dar. In diesem Fall erhalten wir einen partitionierten Abhängigkeitsvektor $\mathbf{d}^{\kappa}_{1,1,\ldots,1} = \mathbf{0}$. Folglich ensteht auch nur eine transformierte Teilzuweisung $s_{t,(1,1,\ldots,1)}$.

Da entsprechend unserer Algorithmenkonvention nur bei der Interndatenteilzuweisung Abhängigkeitsvektoren $\mathbf{d} \neq \mathbf{0}$ auftreten können, kann auch nur aus einer Interndatenteilzuweisung im Ausgangsalgorithmus mehrere Interndatenteilzuweisungen im transformierten Algorithmus entstehen. Für die anderen Teilzuweisungstypen erhalten wir somit immer aus einer Teilzuweisung im Ausgangsalgorithmus eine Teilzuweisung im transformierten Algorithmus. Die Transformation dieser Teilzuweisungen sieht wie folgt aus:

- *Eingabeteilzuweisung:* Bei der Eingabeteilzuweisung ist die linksseitige Variable eine interne Variable und die rechtsseitige Variable eine globale Variable. Die Teilzuweisung aus (3.26) wird wie folgt transformiert:

$$s_t : \qquad y'_s[\mathbf{i}^{\kappa}] = y_{s'}[\mathbf{R}^{k_{s'}}_{s's_t}\mathbf{\Theta}^{\kappa}\mathbf{i}^{\kappa} + \mathbf{r}^{k_{s'}}_{s's_t}], \qquad \mathbf{i}^{\kappa} \in \mathcal{I}^{\kappa}_{s_t}, \tag{4.58}$$

  wobei $\mathcal{I}^{\kappa}_{s_t}$ der partitionierte Iterationsraum des Iterationsraums $\mathcal{I}_{s_t}$ ist.

- *Ausgabeteilzuweisung:* Die linksseitige Variable der Ausgabeteilzuweisung ist eine globale Variable und die rechtsseitige Variable eine lokale Variable. Für die Teilzuweisung aus (3.27) erhalten wir:

$$s_t : \qquad y_s[\mathbf{L}_s\mathbf{\Theta}^{\kappa}\mathbf{i}^{\kappa} + \mathbf{l}_s] = y'_{s'}[\mathbf{i}^{\kappa}], \qquad \mathbf{i}^{\kappa} \in \mathcal{I}^{\kappa}_{s_t}. \tag{4.59}$$

- *Lokale Berechnungsteilzuweisung:* Die lokale Bechnungszuweisung besitzt nur lokale Variablen. Die Partitionierung der Teilzuweisung aus (3.33) ergibt

$$s_t : \qquad y'_s[\mathbf{i}^{\kappa}] = F_{s_t}\left(\ldots, y'_{s'}[\mathbf{i}^{\kappa}], \ldots\right), \qquad \mathbf{i}^{\kappa} \in \mathcal{I}^{\kappa}_{s_t}. \tag{4.60}$$

Wie die Transformation des Algorithmus zeigt, ist der mehrstufig partitionierte Algorithmus wieder eine UItA, denn der transformierte Algorithmus erfüllt alle Bedingungen aus Definition 3.1. Der mehrstufig partitionierte Algorithmus hat durch den Zerfall der Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ des Ausgangsalgorithmus im Allgemeinen mehr Datenabhängigkeiten $e^{\kappa} \in \mathcal{E}^{\kappa}$ als der Ausgangsalgorithmus. Die Abhängigkeitsvektoren $\mathbf{d}(e^{\kappa})$ des mehrstufig partitionierten Algorithmus sind die durch den Zerfall der Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ entstandenen partitionierten Abhängigkeitsvektoren $\mathbf{d}^{\kappa} \in \mathcal{D}^{\kappa}(e)$.

Anhand eines Beispiels soll nun die mehrstufige Partitionierung eines Algorithmus illustriert werden.

Beispiel 4.8 (Partitionierung des FIR-Filteralgorithmus)
In Beispiel 3.1 wurde das FIR-Filter als UItA vorgestellt. Dieser Algorithmus soll zweifach partitioniert werden. Hierzu muss der Algorithmus 3.1 in einen Algorithmus mit ausschließlich lokalen Berechnungszuweisungen transformiert werden. In Beispiel 3.8 wurde diese Transformation vorgestellt.

Für die zweistufige Partitionierung nehmen wir an: $\mathbf{\Theta}^1 = \left(\begin{smallmatrix} 5 & 0 \\ 0 & 2 \end{smallmatrix}\right)$ *und* $\mathbf{\Theta}^2 = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 4 \end{smallmatrix}\right)$.

Der Algorithmus besitzt drei Interndatenteilzuweisungen s_{1_2}, s_{2_3} und $s_{3,1_1}$ mit Abhängigkeitsvektoren $\mathbf{d}(e) \neq \mathbf{0}$. Diese Abhängigkeitsvektoren sind $\mathbf{d}^1_{s_1,s_{1_2}} = \mathbf{d}(a) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$, $\mathbf{d}^1_{s_2,s_{1_3}} = \mathbf{d}(x) = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ und $\mathbf{d}^1_{s_{3,2},s_{3,1_1}} = \mathbf{d}(y) = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$. Die BPAs dieser Abhängigkeitsvektoren sind in Abbildung 4.9 gegeben. Die Abhängigkeitsvektoren $\mathbf{d}(a)$ und $\mathbf{d}(y)$ zerfallen jeweils in drei partitionierte Abhängigkeitsvektoren und der Abhängigkeitsvektor $\mathbf{d}(x)$ in neun partitionierte Abhängigkeitsvektoren. Somit steigt die Anzahl der Teilzuweisungen der Zuweisung s_1 von zwei auf vier, der Zuweisung s_2 von drei auf elf und der Zuweisung $s_{3,1}$ von eins auf drei. Abbildung 4.1 zeigt den partitionierten Algorithmus.

Algorithmus 4.1 Partitionierter FIR-Filteralgorithmus

$$
\begin{aligned}
s_1: \quad & y_a[\mathbf{i}^\kappa] = \begin{cases} a[\left(\begin{smallmatrix}0&1\end{smallmatrix}\right)\boldsymbol{\Theta}^\kappa\mathbf{i}^\kappa], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{1_1} \\ y_a[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{1,1}(a)] & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{1_{2,(1,1)}} \\ y_a[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{2,1}(a)] & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{1_{2,(2,1)}} \\ y_a[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{2,2}(a)] & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{1_{2,(2,2)}} \end{cases} \\
s_2: \quad & y_x[\mathbf{i}^\kappa] = \begin{cases} x[\left(\begin{smallmatrix}1&-1\end{smallmatrix}\right)\boldsymbol{\Theta}^\kappa\mathbf{i}^\kappa], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_1} \\ x[\left(\begin{smallmatrix}1&-1\end{smallmatrix}\right)\boldsymbol{\Theta}^\kappa\mathbf{i}^\kappa], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_2} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{1,1}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(1,1)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{2,1}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(2,1)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{2,2}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(2,2)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{3,1}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(3,1)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{3,2}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(3,2)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{4,1}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(4,1)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{4,2}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(4,2)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{4,3}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(4,3)}} \\ y_x[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{4,4}(x)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{2_{3,(4,4)}} \end{cases} \\
s_{3,1}: \quad & y_{y_t}[\mathbf{i}^\kappa] = \begin{cases} y_y[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{1,1}(y)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{3,1_{1,(1,1)}} \\ y_y[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{2,1}(y)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{3,1_{1,(1,1)}} \\ y_y[\mathbf{i}^\kappa - \mathbf{d}^\kappa_{2,2}(y)], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{3,1_{1,(1,1)}} \end{cases} \\
s_{3,2}: \quad & y_y[\mathbf{i}^\kappa] = \begin{cases} y_a[\mathbf{i}^\kappa] \cdot y_x[\mathbf{i}^\kappa], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{3,2_1} \\ y_{y_t}[\mathbf{i}^\kappa] + y_a[\mathbf{i}^\kappa] \cdot y_x[\mathbf{i}^\kappa], & \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{3,2_2} \end{cases} \\
s_4: \quad & y[\left(\begin{smallmatrix}1&0\end{smallmatrix}\right)\boldsymbol{\Theta}^\kappa\mathbf{i}^\kappa] = y_y[\mathbf{i}^\kappa], \qquad \mathbf{i}^\kappa \in \mathcal{I}^\kappa_{4_1}
\end{aligned}
$$

Die Matrix $\boldsymbol{\Theta}^\kappa$ hat die Größe

$$\boldsymbol{\Theta}^\kappa = \begin{pmatrix}\mathbf{E} & \boldsymbol{\Theta}^1 & \boldsymbol{\Theta}^1\boldsymbol{\Theta}^2\end{pmatrix} = \begin{pmatrix}1 & 0 & 5 & 0 & 15 & 0\\ 0 & 1 & 0 & 2 & 0 & 8\end{pmatrix}.$$

(a) Abhängigkeitsvektor $\mathbf{d}(a)$

(b) Abhängigkeitsvektor $\mathbf{d}(y)$

(c) Abhängigkeitsvektor $\mathbf{d}(x)$

Abbildung 4.9: BPA der Abhängigkeitsvektoren des FIR-Filters ($\mathbf{d} \neq \mathbf{0}$)

Die Polyedermatrix $\mathbf{A}^{\kappa}$ *der partitionierten Iterationsräume* $\mathcal{I}^{\kappa}_{s_t(i_1,i_2)}$ *lautet:*

$$\mathbf{A}^{\kappa} = \begin{pmatrix} \mathbf{E} & \mathbf{0} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} & \mathbf{0} \\ \mathbf{0} & -\mathbf{E} & \mathbf{0} \\ \mathbf{A} & \mathbf{A}\boldsymbol{\Theta}^1 & \mathbf{A}\boldsymbol{\Theta}^1\boldsymbol{\Theta}^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 5 & 0 & 15 & 0 \\ 0 & 1 & 0 & 2 & 0 & 8 \\ -1 & 0 & -5 & 0 & -15 & 0 \\ 0 & -1 & 0 & -2 & 0 & -8 \end{pmatrix}.$$

Die Polyedervektoren $\mathbf{a}^{\kappa}_{0,s_{t,(i_1,i_2)}}$ *für die partitionierten Iterationsräume* $\mathcal{I}^{\kappa}_{s_t(i_1,i_2)}$ *lauten:*

$$\begin{aligned}
\mathbf{a}^{\kappa}_{0,1_1} &= \big(0\ \ 0\ \ -4\ \ -1\ \ 0\ \ 0\ \ -2\ \ -3\ \ 0\ \ 0\ \ 0\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,1_{2,(1,1)}} &= \big(0\ \ 0\ \ -3\ \ -1\ \ 0\ \ 0\ \ -2\ \ -3\ \ 1\ \ 0\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,1_{2,(2,1)}} &= \big(4\ \ 0\ \ -4\ \ -1\ \ 0\ \ 0\ \ -1\ \ -3\ \ 1\ \ 0\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,1_{2,(2,2)}} &= \big(4\ \ 0\ \ -4\ \ -1\ \ 2\ \ 0\ \ -2\ \ -3\ \ 1\ \ 0\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\\
\mathbf{a}^{\kappa}_{0,2_1} &= \big(0\ \ 0\ \ -4\ \ -1\ \ 0\ \ 0\ \ -2\ \ -3\ \ 0\ \ 0\ \ 0\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_2} &= \big(0\ \ 0\ \ -4\ \ -1\ \ 0\ \ 0\ \ -2\ \ -3\ \ 1\ \ 0\ \ -(I-1)\ \ 0\ \ \big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(1,1)}} &= \big(0\ \ 0\ \ -3\ \ 0\ \ 0\ \ 0\ \ -2\ \ -3\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(2,1)}} &= \big(0\ \ 1\ \ -3\ \ -1\ \ 0\ \ 0\ \ -2\ \ -2\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(2,2)}} &= \big(0\ \ 1\ \ -3\ \ -1\ \ 0\ \ 3\ \ -2\ \ -3\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(3,1)}} &= \big(4\ \ 0\ \ -4\ \ 0\ \ 0\ \ 0\ \ -1\ \ -3\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(3,2)}} &= \big(4\ \ 0\ \ -4\ \ 0\ \ 2\ \ 0\ \ -2\ \ -3\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(4,1)}} &= \big(4\ \ 1\ \ -4\ \ -1\ \ 0\ \ 0\ \ -1\ \ -2\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(4,2)}} &= \big(4\ \ 1\ \ -4\ \ -1\ \ 0\ \ 3\ \ -1\ \ -3\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(4,3)}} &= \big(4\ \ 1\ \ -4\ \ -1\ \ 2\ \ 0\ \ -2\ \ -2\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,2_{3,(4,4)}} &= \big(4\ \ 1\ \ -4\ \ -1\ \ 2\ \ 3\ \ -2\ \ -3\ \ 1\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\\
\mathbf{a}^{\kappa}_{0,3,1_{1,(1,1)}} &= \big(0\ \ 0\ \ -4\ \ 0\ \ 0\ \ 0\ \ -2\ \ -3\ \ 0\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,3,1_{1,(2,1)}} &= \big(0\ \ 1\ \ -4\ \ -1\ \ 0\ \ 0\ \ -2\ \ -2\ \ 0\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\mathbf{a}^{\kappa}_{0,3,1_{1,(2,2)}} &= \big(0\ \ 1\ \ -4\ \ -1\ \ 0\ \ 3\ \ -2\ \ -3\ \ 0\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\\
\mathbf{a}^{\kappa}_{0,3,2_1} &= \big(0\ \ 0\ \ -4\ \ -1\ \ 0\ \ 0\ \ -2\ \ -3\ \ 0\ \ 0\ \ -(I-1)\ \ 0\ \ \big)^t, \\
\mathbf{a}^{\kappa}_{0,3,2_2} &= \big(0\ \ 0\ \ -4\ \ -1\ \ 0\ \ 0\ \ -2\ \ -3\ \ 0\ \ 1\ \ -(I-1)\ \ -(J-1)\big)^t, \\
\\
\mathbf{a}^{\kappa}_{0,4_1} &= \big(0\ \ 0\ \ -4\ \ -1\ \ 0\ \ 0\ \ -2\ \ -3\ \ 0\ \ (J-1)\ \ -(I-1)\ \ -(J-1)\big)^t.
\end{aligned}$$

Kapitel 5

Mehrstufige Modifizierte Copartitionierung

Im vorherigen Kapitel wurde die mehrstufige Partitionierung vorgestellt. Diese Algorithmentransformation ist die mathematische Grundlage für die mehrstufige modifizierte Copartitionierung (kurz: MMC). Die mehrstufige modifizierte Copartitionierung ist die entscheidende Abbildungmethode zur Abbildung eines Algorithmus auf eine Architektur mit Parallelverarbeitung auf mehreren Ebenen.

Zur Abbildung eines Algorithmus auf ein Rechenfeld wird jeder partitionierten Iteration $\mathbf{i}^{\kappa} \in \mathcal{I}^{\kappa}$ ein Prozessorelement $\mathbf{p} = \gamma(\mathbf{i}^{\kappa}) \in \mathcal{P}$ des Rechenfeldes $\mathcal{P}$ (Allokation) und ein Abarbeitungszeitpunkt $t = t(\mathbf{i}^{\kappa}, s_t)$ (Ablaufplanung) zugeordnet, wobei der Abarbeitungszeitpunkt zusätzlich zur Iteration $\mathbf{i}^{\kappa}$ von der Teilzuweisung s_t des Algorithmus abhängig ist.

Bei dieser Abbildungsmethode werden die Partitionen einer jeden Partitionierungsstufe als paralleles Rechenfeld (lokal parallel) oder als Sequenz von nacheinander abzuarbeitenden Iterationen (lokal sequenziell) interpretiert. Die verschiedenen Interpretationen einer Partitionierung, die darauf aufbauende Variante der Copartitionierung und deren Erweiterung zur mehrstufigen modifizierten Copartitionierung werden im nachfolgenden Abschnitt vorgestellt.

Wenn die Elemente einer Partition als Sequenz von nacheinander abzuarbeitenden Iterationen betrachtet werden, so ist zusätzlich ein exakter Ablaufplan zu erstellen. Die Erstellung dieses Ablaufplans und deren mathematische Beschreibung wird im Abschnitt 5.2 präsentiert. Dieser Ablaufplan beinhaltet ein Modell, welches zu jeder Teilzuweisung des Algorithmus eine Funktionseinheit des Prozessorelements zuordnet und somit die Operation dieser Teilzuweisung ausführt. Diese Zuordnung von Funktionseinheit zu Teilzuweisung bezeichnen wir als Bindung.

5.1 Partitionierungsvarianten

In diesem Abschnitt erfolgt zunächst die Beschreibung der zwei Grundversionen der Partitionierung eines Iterationsraums zur Abbildung eines Algorithmus auf ein Rechenfeld. Diese beiden Grundversionen basieren auf einer einstufigen Partitionierung. Darauf aufbauend wird die modifizierte Copartitionierung erläutert, die die stufenweise Anwendung der zwei Grundversionen der Partitionierung umfasst.

Erstmals wurde das Prinzip der Copartitionierung in [EM99] präsentiert. In dieser Arbeit wurde die Copartitionierung nach der Ort-Zeit-Transformation auf das Rechenfeld voller Größe angewendet. Wie in [Sie03] gezeigt werden konnte, kann die Copartitionierung aber auch direkt auf den Algorithmus angewendet werden, denn durch die (modifizierte) Copartitionierung findet eine Allokation und eine Ablaufplanung (engl.: scheduling) statt.

Nach der Vorstellung der modifizierten Copartitionierung wird der allgemeine Fall der mehrstufigen modifizierten Copartitionierung für die Abbildung von Algorithmen auf Rechenfelder mit Parallelverarbeitung auf mehreren Ebenen eingeführt.

5.1.1 Grundversionen der Partitionierung

Die beiden Grundversionen für die Partitionierung eines Iterationsraums zur Abbildung eines Algorithmus auf ein Rechenfeld sind die lokal parallele, global sequenzielle (LPGS) Partitionierung und die lokal sequenzielle, global parallele (LSGP) Partitionierung. Zur Unterscheidung der beiden Versionen erhalten alle Variablen und Parameter der Partitionierungsgleichung (4.5) ein höhergestelltes P für die LPGS-Partitionierung bzw. ein höhergestelltes S für die LSGP-Partitionierung, wie in (5.1) und (5.2) dargestellt:

$$\text{LPGS:} \qquad \mathbf{i} = \boldsymbol{\kappa}^P + \boldsymbol{\Theta}^P \widehat{\boldsymbol{\kappa}}^P + \boldsymbol{\vartheta}_0, \qquad \boldsymbol{\vartheta}^P = \operatorname{diag}\left(\boldsymbol{\Theta}^P\right), \tag{5.1}$$

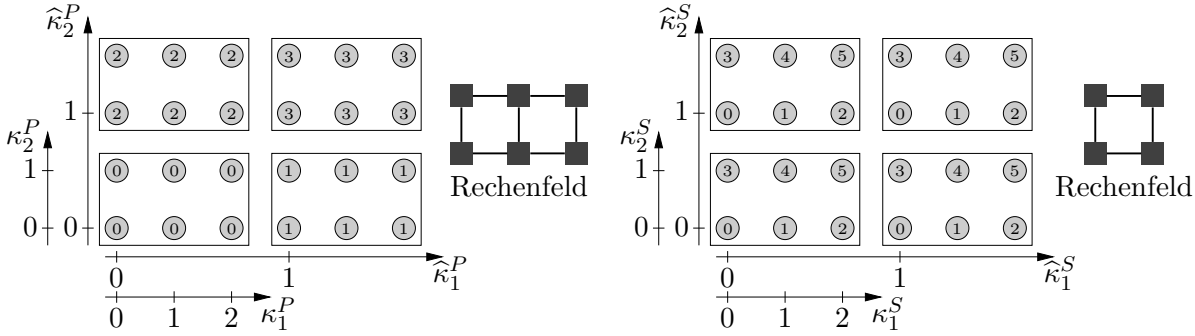
$$\text{LSGP:} \qquad \mathbf{i} = \boldsymbol{\kappa}^S + \boldsymbol{\Theta}^S \widehat{\boldsymbol{\kappa}}^S + \boldsymbol{\vartheta}_0, \qquad \boldsymbol{\vartheta}^S = \operatorname{diag}\left(\boldsymbol{\Theta}^S\right). \tag{5.2}$$

Bei der LPGS-Partitionierung werden alle Iterationen einer Partition gleichzeitig ausgeführt. Somit ergibt sich die Größe ($\det(\boldsymbol{\Theta}^P)$) und die Dimension ($\sum_{j=0}^{n} \operatorname{sign}(\vartheta_j^P - 1)$) des Rechenfeldes aus der Partitionsmatrix $\boldsymbol{\Theta}^P$. Die Partitionen, referenziert durch den Vektor $\widehat{\boldsymbol{\kappa}}^P$, werden nacheinander abgearbeitet.

Bei der LSGP-Partitionierung werden hingegen alle Iterationen einer Partition nacheinander auf einem Prozessorelement abgearbeitet. Die Berechnung der Partitionen findet nun parallel statt. Die Größe ($|\widehat{\mathcal{K}}|$) des Rechenfeldes resultiert somit aus der Anzahl der Partitionen, welche benötigt werden, um den Iterationsraum $\mathcal{I}$ zu überdecken.

Die Partitionen der LSGP-Partitionierung werden im Folgenden LS-Partitionen und die der LPGS-Partitionierung LP-Partitionen genannt.

Beispiel 5.1 (LPGS- und LSGP-Partitionierung eines Iterationsraums)
Der Iterationsraum $\mathcal{I}$ hat die Größe $\mathcal{I} = \left\{\mathbf{i} = \binom{i_1}{i_2} \mid 0 \leq i_1 < 6 \wedge 0 \leq i_2 < 4\right\}$. Abbildung 5.1 zeigt die LSGP- und die LPGS-Partitionierung des Iterationsraums und die



(a) LPGS-Partitionierung (b) LSGP-Partitionierung

Abbildung 5.1: Grundversionen der Partitionierung

zugehörigen Rechenfelder für $\mathbf{\Theta}^P = \mathbf{\Theta}^S = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 2 \end{smallmatrix}\right)$. *Für jede Iteration wird der Zeitpunkt der Abarbeitung im jeweiligen Iterationspunkt angegeben.*

Was sind die wesentlichen Unterschiede der beiden Grundversionen?

- Bei der LPGS-Partitionierung wird die Größe des Rechenfeldes durch die Partitionsmatrix $\mathbf{\Theta}^P$ festgelegt. Die Größe des Rechenfeldes ist damit unabhängig von der Größe des Iterationsraums $\mathcal{I}$. Für die LSGP-Partitionierung gibt es zwei Szenarios. (1) Die Partitionsmatrix $\mathbf{\Theta}^S$ ist konstant, so bestimmt die Größe des Iterationsraums $\mathcal{I}$ die Größe des Rechenfeldes. (2) Die Größe des Rechenfeldes ist vorgegeben, dann ist die Größe der LS-Partitionen abhängig von der Größe des Iterationsraums $\mathcal{I}$.

- Die Größe der LS-Partition beeinflusst das Ein- und Ausgabeverhalten (kurz: E/A-Verhalten) des Rechenfeldes und den Bedarf an lokalen Speicher in den Prozessorelementen. Dabei wird angenommen, dass jede Variable, die in einer Partition benötigt wird, nur einmal gelesen bzw. geschrieben wird. Werden Daten innerhalb einer LS-Partition wiederverwendet, so müssen sie lokal gespeichert werden. Im Allgemeinen führt eine größere LS-Partition zu einem niedrigeren Datentransfer zwischen Rechenfeld und Umgebung. Dadurch steigt aber der Bedarf an lokalen Speicher in den Prozessorelementen. Bei der LPGS-Partitionierung sind lokaler Speicherbedarf als auch E/A-Verhalten je Prozessorelement unabhängig von der Größe der LP-Partition.

5.1.2 Modifizierte Copartitionierung

Die modifizierte Copartitionierung ist eine zweistufige Partitionierung bei der ein LSGP-partitionierter Raum nochmals LPGS-partitioniert wird, wie nachfolgende Gleichung zeigt:

$$\mathbf{i} = \vartheta(\mathbf{i}^\kappa) = \boldsymbol{\kappa}^S + \mathbf{\Theta}^S \left(\boldsymbol{\kappa}^P + \mathbf{\Theta}^P \widehat{\boldsymbol{\kappa}}^P \right) + \boldsymbol{\vartheta}_0. \tag{5.3}$$

Das resultierende Rechenfeld $\mathcal{P}$ wird durch die Partitionsmatrix $\mathbf{\Theta}^P$ bestimmt. Für eine Zuordnung der partitionierten Iterationen $\mathbf{i}^\kappa = \begin{pmatrix}\boldsymbol{\kappa}^S & \boldsymbol{\kappa}^P & \widehat{\boldsymbol{\kappa}}^P\end{pmatrix}^t$ des Iterationsraums $\mathcal{I}^\kappa$ zu den Prozessorelementen $\mathbf{p} \in \mathcal{P}$ des Rechenfeldes $\mathcal{P}$ definieren wir eine Allokationsfunktion.

Definition 5.1 (Allokationsfunktion)
Die Zuordnung der partitionierten Iteration $\mathbf{i}^\kappa \in \mathcal{I}^\kappa$ zu dem Prozessorelement $\mathbf{p} \in \mathcal{P}$ erfolgt durch die Allokationsfunktion

$$\mathbf{p} = \gamma(\mathbf{i}^\kappa) = \mathbf{\Gamma}^\kappa \cdot \mathbf{i}^\kappa = \begin{pmatrix}\mathbf{0} & \mathbf{\Gamma} & \mathbf{0}\end{pmatrix} \begin{pmatrix}\boldsymbol{\kappa}^S \\ \boldsymbol{\kappa}^P \\ \widehat{\boldsymbol{\kappa}}^P\end{pmatrix} = \mathbf{\Gamma} \cdot \boldsymbol{\kappa}^P, \qquad \mathbf{\Gamma} \in \mathbb{N}^{n^\kappa \times n}. \tag{5.4}$$

Die Dimension des Rechenfeldes ist durch den Parameter n^κ gegeben. Dieser ergibt sich aus der Partitionsmatrix $\mathbf{\Theta}^P$:

$$n^\kappa = \begin{cases} \sum_{i=1}^{n} sign(\vartheta_i^P - 1), & wenn \quad \mathbf{\Theta}^P \neq \mathbf{E} \\ 1, & sonst. \end{cases} \tag{5.5}$$

Für die Allokationsmatrix $\mathbf{\Gamma} = (\gamma_{i,j})$ müssen die folgenden Bedingungen erfüllt sein:

(1) $\gamma_{i,j} \in \{0, 1\}$,

(2) $\sum_{i=1}^{n^\kappa} \gamma_{i,j} = \begin{cases} 1, & wenn \quad \vartheta_j^P > 1 \\ 0, & wenn \quad \vartheta_j^P = 1 \end{cases}$, *mit* $1 \leq j \leq n$ *und*

(3) $\sum_{j=1}^{n} \gamma_{i,j} = 1$, *mit* $1 \leq i \leq n^\kappa$ *bzw.* $\operatorname{Rg} \mathbf{\Gamma} = n^\kappa$.

Ist die Partitionsmatrix $\mathbf{\Theta}^P = \mathbf{E}$, so wird die Allokationsmatrix definiert als $\mathbf{\Gamma} = (1\,0\,0\,\ldots 0)$.

Die Dimension des Rechenfeldes ergibt sich im Allgemeinen aus der Anzahl der Elemente der Partitionsmatrix $\mathbf{\Theta}^P$, deren Wert größer '1' ist. Somit ist es möglich, dass ein Algorithmus mit einem Iterationsraum $\mathcal{I} \subset \mathbb{Z}^n$ auf ein Rechenfeld $\mathcal{P} \subset \mathbb{N}^{n^\kappa}$, mit $n^\kappa \leq n$, abgebildet werden kann.

Eine Ausnahme bei der Ermittlung der Dimension des Rechenfeldes und der Allokationsmatrix $\mathbf{\Gamma}$ bildet der Fall $\mathbf{\Theta}^P = \mathbf{E}$. Dieser Fall führt zu einem Rechenfeld mit einem Prozessorelement. Entsprechend der allgemeinen Definition ($n^\kappa = \sum_{i=1}^{n} \operatorname{sign}(\vartheta_i^P - 1)$) resultiert aus dieser Definition ein nulldimensionales Rechenfeld. Dieses Rechenfeld wollen wir künftig entsprechend (5.5) ebenfalls als eindimensionales Rechenfeld mit einem Prozessorelement bezeichnen.

Die Größe $\boldsymbol{\vartheta}^\square$ des Rechenfeldes kann nun mit der Gleichung

$$\boldsymbol{\vartheta}^\square = \mathbf{\Gamma} \cdot \boldsymbol{\vartheta}^P, \qquad \boldsymbol{\vartheta}^\square = \begin{pmatrix}\vartheta_1^\square & \vartheta_2^\square & \ldots & \vartheta_{n^\kappa}^\square\end{pmatrix}^t \in \mathbb{N}^{n^\kappa} \tag{5.6}$$

ermittelt werden.

Beispiel 5.2 (Allokation)
Ein Algorithmus soll auf ein Rechenfeld $\mathcal{P}$ der Größe $\boldsymbol{\vartheta}^{\square} = \left(\begin{smallmatrix}2\\4\end{smallmatrix}\right)$ abgebildet werden. Der Iterationsraum des Algorithmus ist dreidimensional. Er wird mit $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix}4&0&0\\0&1&0\\0&0&2\end{smallmatrix}\right)$ partitioniert. Für die Abbildung $\mathbf{p} = \gamma(\boldsymbol{\kappa}^P)$ gilt:

$$\mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \gamma(\boldsymbol{\kappa}^P) = \boldsymbol{\Gamma} \cdot \boldsymbol{\kappa}^P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \kappa_1^P \\ \kappa_2^P \\ \kappa_3^P \end{pmatrix}, \quad 0 \le p_1 < 2,\ 0 \le p_2 < 4. \tag{5.7}$$

In jedem PE werden alle Iterationen einer LS-Partition sequenziell abgearbeitet. Die Abarbeitung der LP-Partitionen erfolgt ebenfalls sequenziell. Die Festlegung der Abarbeitungszeitpunkte wird in Abschnitt 5.2 vorgestellt. Durch die LSGP-Partitionierung ist es möglich, das E/A-Verhalten bzw. den Bedarf an lokalen Speicher einzustellen.

Die modifizierte Copartitionierung eines Abhängigkeitsvektors $\mathbf{d}(e)$ führt zu den partitionierten Abhängigkeitsvektoren

$$\mathbf{d}^{\kappa}_{i_1,i_2} = \left(\mathbf{d}^{\boldsymbol{\Theta}^S}_{i_1}, \mathbf{d}^{\boldsymbol{\Theta}^P}_{i_1,i_2}, \widehat{\mathbf{d}}^{\boldsymbol{\Theta}^P}_{i_1,i_2}\right)^t = \left(\mathbf{d}^{S}_{i_1}\ \mathbf{d}^{P}_{i_1,i_2}\ \widehat{\mathbf{d}}^{P}_{i_1,i_2}\right)^t \in \mathcal{D}^{\kappa}(e). \tag{5.8}$$

Die Teilvektoren $\mathbf{d}^{\boldsymbol{\Theta}^P}_{i_1,i_2}$ repräsentieren hierbei Datenkanäle im Rechenfeld. Die Teilvektoren $\mathbf{d}^{\boldsymbol{\Theta}^S}_{i_1}$ und $\widehat{\mathbf{d}}^{\boldsymbol{\Theta}^P}_{i_1,i_2}$ geben zusammen mit dem Ablaufplan, der in Abschnitt 5.2 vorgestellt wird, Aufschluss über die Dauer der Speicherung einer Variable bis zu deren Verwendung.

Beispiel 5.3 (Modifizierte Copartitionierung eines Iterationsraums)
Abbildung 5.2 zeigt eine modifizierte Copartitionierung des Iterationsraums $\mathcal{I} = \{\mathbf{i} = \left(\begin{smallmatrix}i_1\\i_2\end{smallmatrix}\right) \mid 0 \le i_1 < 8 \wedge 0 \le i_2 < 6\}$ mit LS-Partitionen der Größe $\boldsymbol{\Theta}^S = \left(\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right)$ und LP-Partitionen der Größe $\boldsymbol{\Theta}^S = \left(\begin{smallmatrix}2&0\\0&1\end{smallmatrix}\right)$. Das resultierende Rechenfeld $\mathcal{P}$ ist linear und hat 2 PEs ($n^{\kappa} = 1$, $\mathbf{p} = p = \boldsymbol{\Gamma}\boldsymbol{\kappa}^P = \begin{pmatrix}1 & 0\end{pmatrix}\boldsymbol{\kappa}^P$). Die Zeitpunkte der Abarbeitung der Zuweisungen der Iterationen auf dem Rechenfeld sind in den Iterationspunkten der Abbildung 5.2 angegeben.

In dem Iterationsraum $\mathcal{I}$ soll es einen Abhängigkeitsvektor $\mathbf{d} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ geben. Mit den oben genannten Partitionsgrößen erhalten wir sechs verschiedene partitionierte Abhängigkeitsvektoren:

$$\mathbf{d}^{\kappa}_{1,1} = \left(\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)^t, \quad \mathbf{d}^{\kappa}_{2,1} = \left(\left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right)^t, \quad \mathbf{d}^{\kappa}_{2,2} = \left(\left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)\ \left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\right)^t,$$
$$\mathbf{d}^{\kappa}_{3,1} = \left(\left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right)\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right)^t, \quad \mathbf{d}^{\kappa}_{4,1} = \left(\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right)\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right)^t, \quad \mathbf{d}^{\kappa}_{4,2} = \left(\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right)\ \left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\right)^t.$$

Aus den Teilvektoren $\mathbf{d}^{\boldsymbol{\Theta}^P}_{i_1,i_2}$ der partitionierten Abhängigkeitsvektoren $\mathbf{d}^{\kappa}_{i_1,i_2}$ wird ersichtlich, dass es sowohl einen Datenkanal $\mathbf{q}_1 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ als auch $\mathbf{q}_2 = \left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)$ geben muss oder es gibt einen bidirektionalen Kanal $\mathbf{q} \in \mathcal{Q}^{bi}$ zwischen den Prozessorelementen, wie in Abbildung 5.2 dargestellt.

Zwei Sonderfälle der modifizierten Copartitionierung ergeben sich durch $\boldsymbol{\Theta}^S = \mathbf{E}$ bzw. $\boldsymbol{\Theta}^P = \mathbf{E}$. Diese sollen im Folgenden diskutiert werden.

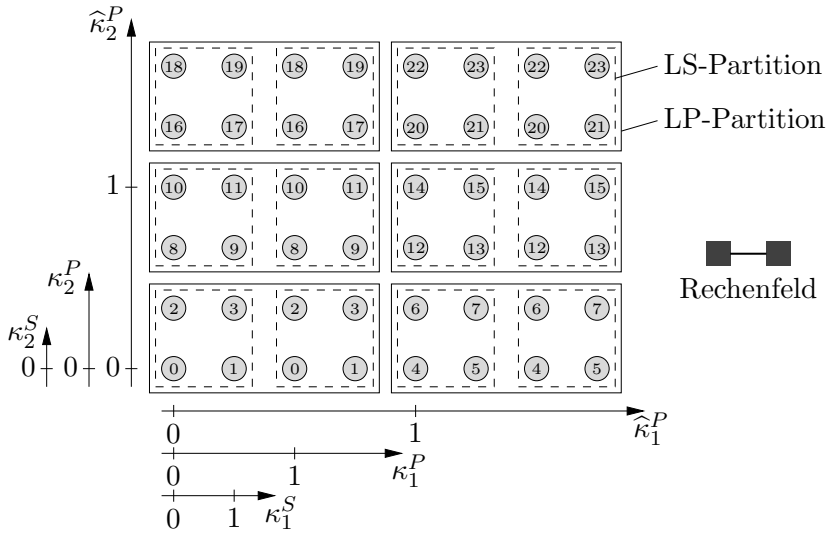


Abbildung 5.2: Modifizierte Copartitionierung mit $\mathbf{\Theta}^S = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 2 \end{smallmatrix}\right)$ und $\mathbf{\Theta}^P = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 1 \end{smallmatrix}\right)$

Modifizierte Copartitionierung mit $\mathbf{\Theta}^S = \mathbf{E}$: In diesem Fall entspricht die modifizierte Copartitionierung der LPGS-Partitionierung, wie (5.9) bzw. (5.1) zeigt:

$$\mathbf{i} = \boldsymbol{\kappa}^S + \mathbf{E}\left(\boldsymbol{\kappa}^P + \mathbf{\Theta}^P \widehat{\boldsymbol{\kappa}}^P\right) = \boldsymbol{\kappa}^P + \mathbf{\Theta}^P \widehat{\boldsymbol{\kappa}}^P, \qquad \boldsymbol{\kappa}^S = \mathbf{0}. \tag{5.9}$$

Diese Partitionierung führt zu einem Rechenfeld mit maximalen Lese- und Schreibezugriffen je Prozessorelement und minimalen lokalen Speicherbedarf, denn die Größe der LS-Partition ist minimal.

Modifizierte Copartitionierung mit $\mathbf{\Theta}^P = \mathbf{E}$: Mit einer solchen Partitionierung erhalten wir ein 'Rechenfeld' mit nur einem Prozessorelement ($\det(\mathbf{\Theta}^P) = 1$):

$$\mathbf{i} = \boldsymbol{\kappa}^S + \mathbf{\Theta}^S\left(\boldsymbol{\kappa}^P + \mathbf{E}\,\widehat{\boldsymbol{\kappa}}^P\right) = \boldsymbol{\kappa}^S + \mathbf{\Theta}^S \widehat{\boldsymbol{\kappa}}^P, \qquad \boldsymbol{\kappa}^P = \mathbf{0}. \tag{5.10}$$

Durch die sequenzielle Abarbeitung der Iterationen in der LS-Partition als auch der Partitionen selbst, ist es möglich, verschiedene Abarbeitungsreihenfolgen zu wählen. Wie sich später zeigen wird, kann mit einer solchen modifizierten Copartitionierung eine Anpassung an den lokalen Speicher bzw. an die externe Speicherhierarchie vorgenommen werden.

5.1.3 Mehrstufige modifizierte Copartitionierung

Die mehrstufige modifizierte Copartitionierung (kurz: MMC) dient der Abbildung von Algorithmen auf Rechenfelder, die auf mehreren Ebenen Parallelverarbeitung gestatten. In Abbildung 5.3 ist ein Beispiel eines Rechenfeldes dargestellt, das in seinen Prozessorelementen wieder ein Rechenfeld enthält und damit Parallelverarbeitung auf zwei Ebenen ermöglicht.

Die mehrstufige modifizierte Copartitionierung basiert darauf, dass ein copartitionierter Iterationsraum stets wieder copartitioniert werden kann. Für eine p-fache Copartitionierung gilt:

$$\begin{aligned}\mathbf{i} = \vartheta(\mathbf{i}^{\kappa}) = \boldsymbol{\kappa}^{S_1} + \boldsymbol{\Theta}^{S_1}\Big(\boldsymbol{\kappa}^{P_1} + \boldsymbol{\Theta}^{P_1}\Big(\boldsymbol{\kappa}^{S_2} + \boldsymbol{\Theta}^{S_2}\Big(\boldsymbol{\kappa}^{P_2} + \ldots \\ \ldots + \boldsymbol{\Theta}^{P_{p-1}}\Big(\boldsymbol{\kappa}^{S_p} + \boldsymbol{\Theta}^{S_p}\Big(\boldsymbol{\kappa}^{P_p} + \boldsymbol{\Theta}^{P_p}\widehat{\boldsymbol{\kappa}}^{P_p}\Big)\Big)\Big)\Big)\Big) + \boldsymbol{\vartheta}_0, \end{aligned} \tag{5.11}$$

$$\mathbf{i} = \vartheta(\mathbf{i}^{\kappa}) = \sum_{k=1}^{p}\left(\prod_{l=1}^{k-1}\left(\boldsymbol{\Theta}^{S_l}\boldsymbol{\Theta}^{P_l}\right)\left(\boldsymbol{\kappa}^{S_k} + \boldsymbol{\Theta}^{S_k}\boldsymbol{\kappa}^{P_k}\right)\right) + \prod_{l=1}^{p}\left(\boldsymbol{\Theta}^{S_l}\boldsymbol{\Theta}^{P_l}\right)\widehat{\boldsymbol{\kappa}}^{P_p} + \boldsymbol{\vartheta}_0, \tag{5.12}$$

wobei $\mathbf{i}^{\kappa} = \begin{pmatrix}\boldsymbol{\kappa}^{S_1} & \boldsymbol{\kappa}^{P_1} & \boldsymbol{\kappa}^{S_2} & \boldsymbol{\kappa}^{P_2} & \ldots & \boldsymbol{\kappa}^{S_p} & \boldsymbol{\kappa}^{P_p} & \widehat{\boldsymbol{\kappa}}^{P_p}\end{pmatrix}^t$ ist.

Mit jeder Copartitionierungsstufe kann mittels der LS-Partition die sequenzielle Abarbeitung und mittels der LP-Partition der Grad der Parallelverarbeitung der Copartitionen der vorherigen Copartitionierungsstufe verändert werden.

Andere Entwurfssysteme, wie zum Beispiel PARO [HDT09] partitionieren den Iterationsraum erst mehrfach und führen anschließend bei der Ort-Zeit-Abbildung (engl.: space-time mapping) eine Allokation und Ablaufplanung durch. Hierbei wird streng zwischen Algorithmentransformation und Abbildung auf die Zielarchitektur bzw. Entwurf der Prozessorarchitektur unterschieden. Da die Algorithmentransformation mit dem Ziel der Abbildung auf die Zielarchitektur bzw. des Entwurfs der Prozessorarchitektur stattfindet, ist es konsequent, diese beiden Schritte mit der mehrstufigen modifizierten Copartitionierung zusammenzufassen.

Es wurde aus folgenden Gründen eine mehrstufige modifizierten Copartitionierung definiert, statt einer mehrstufigen Partitionierung mit freiwählbarer LSGP- oder LPGS-Partitionierung je Partitionierungsstufe:

(1) Es wird durch die wechselnde Anwendung der Partitionierungsgrundversionen immer der äußere parallele bzw. sequenzielle Teil einer LSGP- bzw. LPGS-Partitionierung durch die nachfolgende Partitionierung LPGS- bzw. LSGP-partitioniert, deren innerer Teil gleichfalls parallel bzw. sequenziell abgearbeitet wird.

(2) Die mehrstufige modifizierte Copartitionierung gestattet eine systematische Beschreibung.

(3) Durch entsprechende Wahl der Partitionsmatrix $\boldsymbol{\Theta}^{P_k} = \mathbf{E}$ bzw. $\boldsymbol{\Theta}^{S_k} = \mathbf{E}$ kann eine lokal parallele bzw. eine lokal sequenzielle Abarbeitung in jeder Copartitionierungsstufe k vermieden werden. Die systematische Beschreibung bleibt dabei erhalten.

(4) Der durch die mehrstufige modifizierte Copartitionierung entstandene Zusatzaufwand in der Beschreibung (z. B. $\boldsymbol{\kappa}^{P_k} = \mathbf{0}, \forall \boldsymbol{\kappa}^{P_k} \in \mathcal{K}^{P_k}$ bei $\boldsymbol{\Theta}^{P_k} = \mathbf{E}$) kann, wenn notwendig, durch nachfolgende Transformationen wieder leicht beseitigt werden.

Die Allokationsfunktion für die mehrstufige modifizierte Copartitionierung lautet nun:

$$\mathbf{p} = \begin{pmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \vdots \\ \mathbf{p}^p \end{pmatrix} = \gamma(\mathbf{i}^\kappa) = \mathbf{\Gamma}^\kappa \cdot \mathbf{i}^\kappa = \begin{pmatrix} \mathbf{0} & \mathbf{\Gamma}^1 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{\Gamma}^2 & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{\Gamma}^p & \mathbf{0} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{\kappa}^{S_1} \\ \boldsymbol{\kappa}^{P_1} \\ \boldsymbol{\kappa}^{S_2} \\ \boldsymbol{\kappa}^{P_2} \\ \vdots \\ \boldsymbol{\kappa}^{S_p} \\ \boldsymbol{\kappa}^{P_p} \\ \widehat{\boldsymbol{\kappa}}^{P_p} \end{pmatrix}, \tag{5.13}$$

wobei $\mathbf{p}^k$, $1 \leq k \leq p$, das Rechenfeld auf der k-ten Copartitionierungsebene beschreibt. Für die Allokationsmatrizen $\mathbf{\Gamma}^k$, $1 \leq k \leq p$, müssen die Bedingungen aus Definition 5.1 gelten.

Ein Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{d}$ aus dem Iterationsraum $\mathcal{I}$ zerfällt in einem p-fach copartitionierten Iterationsraum in partitionierte Abhängigkeitsvektoren $\mathbf{d}^\kappa_{i_1,\ldots,i_{2p}} \in \mathcal{D}^\kappa(e)$ der Form:

$$\mathbf{d}^\kappa_{i_1,\ldots,i_{2p}} = \left(\mathbf{d}^{S_1}_{i_1}, \mathbf{d}^{P_1}_{i_1,i_2}, \mathbf{d}^{S_2}_{i_1,i_2,i_3}, \ldots, \mathbf{d}^{P_{p-1}}_{i_1,i_2,\ldots,i_{2(p-1)}}, \mathbf{d}^{S_p}_{i_1,i_2,\ldots,i_{2p-1}}, \mathbf{d}^{P_p}_{i_1,\ldots,i_{2p}}, \widehat{\mathbf{d}}^{P_p}_{i_1,\ldots,i_{2p}}\right)^t.$$

Die Teilvektoren $\mathbf{d}^{P_k}_{i_1,i_2,\ldots,i_{2k}}$ der k-ten Copartitionierungsstufe stellen einen Datenkanal innerhalb dieser Copartitionierungsstufe dar. Die anderen Teilvektoren geben mit Hilfe des Ablaufplans aus Abschnitt 5.2 Aufschluss über die Dauer der Speicherung einer Variable bis zu deren Verwendung.

Beispiel 5.4 (Zweifach modifizierte Copartitionierung eines Iterationsraums) *Abbildung 5.3 zeigt eine zweistufige modifizierte Copartitionierung des Iterationsraums* $\mathcal{I} = \{\mathbf{i} = \binom{i_1}{i_2} \mid 0 \leq i_1 < 8 \wedge 0 \leq i_2 < 6\}$ *mit*

$$\mathbf{\Theta}^{S_1} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{\Theta}^{P_1} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathbf{\Theta}^{S_2} = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \quad \textit{und} \quad \mathbf{\Theta}^{P_2} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}. \tag{5.14}$$

Das resultierende lineare Rechenfeld $\mathcal{P}_2 \subset \mathbb{N}^1$ *hat zwei PEs* ($\mathbf{\Theta}^{P_2}$), *wobei in jedem PE sich wieder ein lineares Rechenfeld* $\mathcal{P}_1 \subset \mathbb{N}^1$ *mit zwei PEs* ($\mathbf{\Theta}^{P_1}$) *befindet. Für die Allokationsmatrizen* $\mathbf{\Gamma}_1$ *und* $\mathbf{\Gamma}_2$ *gilt:* $\mathbf{\Gamma}_1 = \mathbf{\Gamma}_2 = \begin{pmatrix} 1 & 0 \end{pmatrix}$. *In Abbildung 5.3 sind in den Iterationspunkten des Iterationsraums die Zeiten zur Abarbeitung der Zuweisungen der Iteration angegeben.*

Existiert in dem Iterationsraum $\mathcal{I}$ *wieder ein Abhängigkeitsvektor* $\mathbf{d} = \binom{1}{1}$, *so erhalten wir für die zweistufige Copartitionierung mit den oben angegebenen Parametern acht partitionierte Abhängigkeitsvektoren* $\mathbf{d}^\kappa_{i_1,i_2,i_3,i_4}$:

$$\mathbf{d}^\kappa_{1,1,1,1} = \left(\binom{1,0}{} \ \binom{0}{0} \ \binom{0}{1} \ \binom{0}{0} \ \binom{0}{0}\right)^t, \qquad \mathbf{d}^\kappa_{1,1,2,1} = \left(\binom{1}{0} \ \binom{0}{0} \ \binom{0}{-2} \ \binom{0}{0} \ \binom{0}{1}\right)^t,$$
$$\mathbf{d}^\kappa_{2,1,1,1} = \left(\binom{-1}{0} \ \binom{1}{0}, \ \binom{0}{1} \ \binom{0}{0} \ \binom{0}{0}\right)^t, \qquad \mathbf{d}^\kappa_{2,1,2,1} = \left(\binom{-1}{0} \ \binom{1}{0}, \ \binom{0}{-2} \ \binom{0}{0} \ \binom{0}{1}\right)^t,$$
$$\mathbf{d}^\kappa_{2,2,1,1} = \left(\binom{-1}{0} \ \binom{-1}{0} \ \binom{0}{1} \ \binom{1}{0} \ \binom{0}{0}\right)^t, \qquad \mathbf{d}^\kappa_{2,2,1,2} = \left(\binom{-1}{0} \ \binom{-1}{0} \ \binom{0}{1} \ \binom{-1}{0} \ \binom{1}{0}\right)^t,$$
$$\mathbf{d}^\kappa_{2,2,2,1} = \left(\binom{-1}{0} \ \binom{-1}{0} \ \binom{0}{-2} \ \binom{1}{0} \ \binom{0}{1}\right)^t, \qquad \mathbf{d}^\kappa_{2,2,2,2} = \left(\binom{-1}{0} \ \binom{-1}{0} \ \binom{0}{-2} \ \binom{-1}{0} \ \binom{1}{1}\right)^t.$$

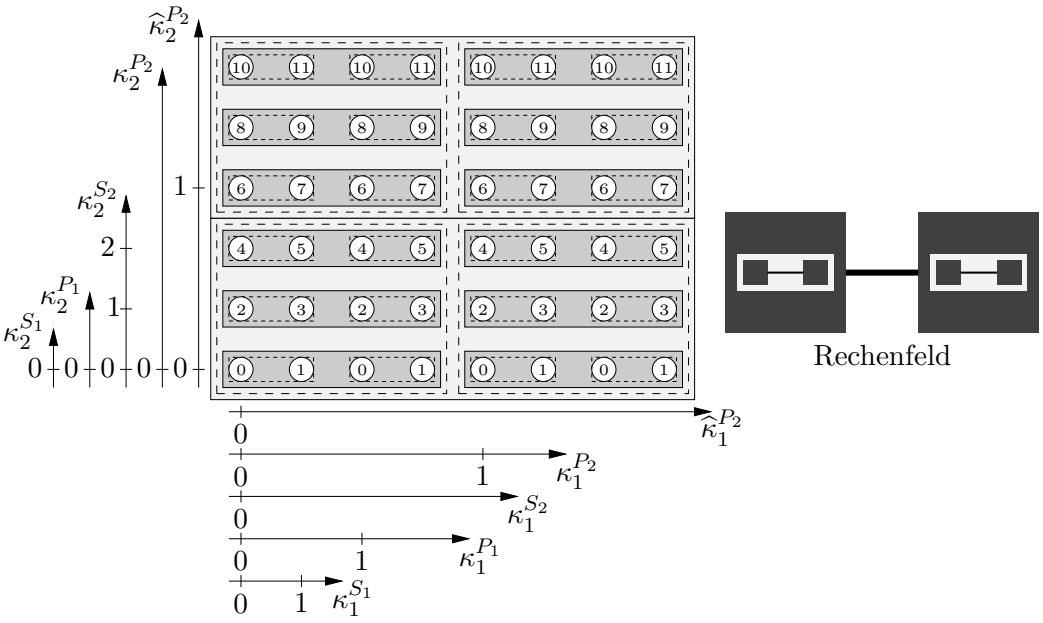


Abbildung 5.3: Zweistufige modifizierte Copartitionierung

Diese führen zu jeweils einen bidirektionalen Datenkanal $\mathbf{q} = \binom{1}{0} \in \mathcal{Q}^{bi}$ *in jeder Copartitionierungsstufe. Das Rechenfeld in Abbildung 5.3 zeigt diese Kanäle.*

Nachfolgendes Beispiel zeigt einen zweistufig copartitionierten Iterationsraum, bei dem in der ersten Copartitinierungsstufe die Partitionsmatrix $\mathbf{\Theta}^{S_1} = \mathbf{E}$ und in der zweiten Copartitinierungsstufe die Partitionsmatrix $\mathbf{\Theta}^{P_2} = \mathbf{E}$ gesetzt wurde. Somit haben wir in der ersten Copartitinierungsstufe keine lokal sequenzielle Abarbeitung und in der zweiten Copartitinierungsstufe keine lokal parallele Verarbeitung.

Beispiel 5.5
Der Iterationsraum $\mathcal{I} = \{\mathbf{i} = \binom{i_1}{i_2} \mid 0 \leq i_1 < 8 \wedge 0 \leq i_2 < 6\}$ *aus Beispiel 5.4 wird in diesem Beispiel ebenfalls zweistufig copartitioniert, wobei jeweils eine Partitionsmatrix je Copartitionierungsstufe eine Einheitsmatrix ist:*

$$\mathbf{\Theta}^{S_1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{\Theta}^{P_1} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathbf{\Theta}^{S_2} = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \quad \textit{und} \quad \mathbf{\Theta}^{P_2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{5.15}$$

Abbildung 5.4 zeigt den in dieser Form partitionierten Iterationsraum $\mathcal{I}$ *und das zugehörige Rechenfeld. In den Iterationen sind die Abarbeitungzeitpunkte für die Zuweisungen der jeweiligen Iteration angegeben.*

Durch die Partitionsmatrix $\mathbf{\Theta}^{S_1} = \mathbf{E}$ *erhalten wir in der ersten Copartitionierungsstufe eine Parallelverarbeitung auf einem linearen Rechenfeld* $\mathcal{P}_1 = \{0, 1\} \subset \mathbb{N}^1$ *mit zwei PEs (*$\mathbf{\Theta}^{P_1}$*). Eine lokal sequenzielle Abarbeitung findet nicht statt. Durch die Copartitionierung mit* $\mathbf{\Theta}^{P_2} = \mathbf{E}$ *findet in der zweiten Copartitionierungstufe keine Parallelverarbeitung* $\mathcal{P}_2 = \{0\} \subset \mathbb{N}^1$ *statt. Mittels der Partitionsmatrix* $\mathbf{\Theta}^{S_2} \neq \mathbf{E}$ *wird aber die*

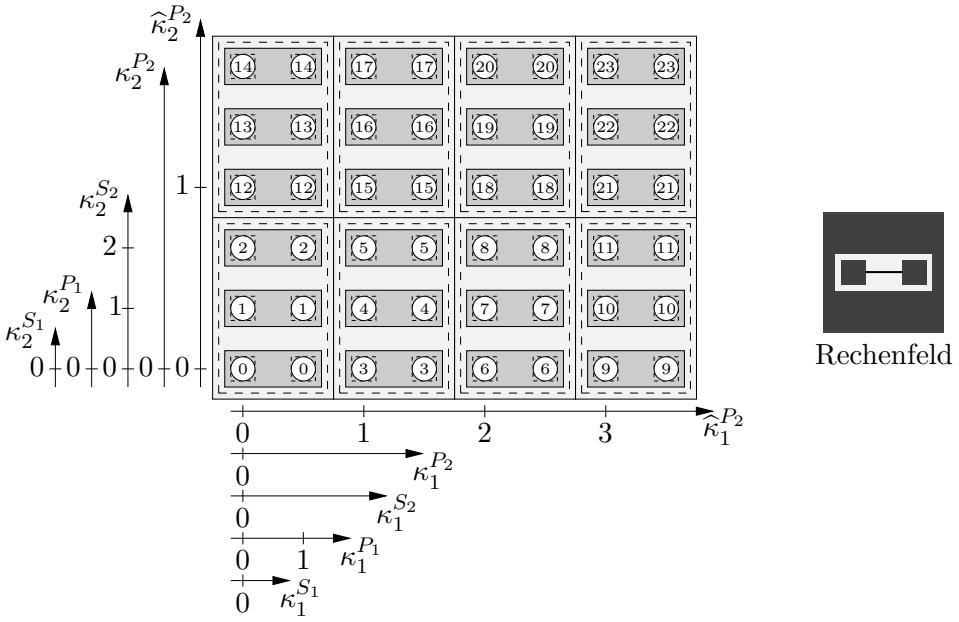


Abbildung 5.4: Zweistufige modifizierte Copartitionierung mit $\mathbf{\Theta}^{S_1} = \mathbf{E}$ und $\mathbf{\Theta}^{P_2} = \mathbf{E}$

Abarbeitungsreihenfolge verändert, wie in Abbildung 5.4 zu erkennen ist. Die Partitionierungsgleichung lautet:

$$\mathbf{i} = \vartheta(\mathbf{i}^{\kappa}) = \underbrace{\boldsymbol{\kappa}^{S_1}}_{\mathbf{0}} + \underbrace{\mathbf{\Theta}^{S_1}}_{\mathbf{E}}\Big(\boldsymbol{\kappa}^{P_1} + \mathbf{\Theta}^{P_1}\Big(\boldsymbol{\kappa}^{S_2} + \mathbf{\Theta}^{S_2}\Big(\underbrace{\boldsymbol{\kappa}^{P_2}}_{\mathbf{0}} + \underbrace{\mathbf{\Theta}^{P_2}}_{\mathbf{E}} \cdot \widehat{\boldsymbol{\kappa}}^{P_2}\Big)\Big)\Big) \tag{5.16}$$

Diese Gleichung kann aufgrund $\mathbf{\Theta}^{S_1} = \mathbf{\Theta}^{P_2} = \mathbf{E}$ und damit $\boldsymbol{\kappa}^{S_1} = \boldsymbol{\kappa}^{P_2} = \mathbf{0}$ zu

$$\mathbf{i} = \vartheta(\mathbf{i}^{\kappa}) = \boldsymbol{\kappa}^{P_1} + \mathbf{\Theta}^{P_1}\big(\boldsymbol{\kappa}^{S_2} + \mathbf{\Theta}^{S_2} \cdot \widehat{\boldsymbol{\kappa}}^{P_2}\big) \tag{5.17}$$

vereinfacht werden.

5.2 Ablaufplanung

Mit der modifizierte Copartitionierung wurde jeder Iteration $\mathbf{i} \in \mathcal{I}$ ein Prozessorelement $\mathbf{p} \in \mathcal{P}$ zugeordnet. Bezüglich des Abarbeitungsverhalten wurde bisher nur festgelegt, welche Iterationen parallel und welche sequenziell verarbeitet werden sollen. Die Ablaufplanung, d. h. die Zuordnung der partitionierten Iterationen zu Abarbeitungszeitpunkten, ist Gegenstand dieses Abschnitts.

In dieser Arbeit betrachten wir nur lineare Ablaufpläne (engl.: linear schedules). Diese Pläne können bei Bedarf zu stückweise linearen Ablaufplänen (engl.: piecewise linear schedules) erweitert werden. Dabei werden unterschiedliche lineare Ablaufpläne für Teilräume des Iterationsraums $\mathcal{I}$ bzw. des partitionierten Iterationsraums $\mathcal{I}^{\kappa}$ definiert [DKR92][RMK92].

In diesem Abschnitt werden zuerst die Methoden zur Beschreibung der Ablaufplans vorgestellt. Anschließend werden diese an der einfachen und mehrstufigen modifizierten Copartitionierung demonstriert.

5.2.1 Abfolgevektor α, Ablaufvektor τ und Leerlaufverzögerung ι

Die sequenzielle Abarbeitung der Elemente einer LS-Partition mit der Größe $\mathbf{\Theta}^S$ soll mittels linearer Ablaufplanung (engl.: linear scheduling) erfolgen. Das heißt, die einer Iteration $\boldsymbol{\kappa}^S$ zugeordnete Zeit t wird bestimmt durch:

$$t = \boldsymbol{\tau} \cdot \boldsymbol{\kappa}^S, \qquad \boldsymbol{\tau} \in \mathbb{Z}^{1\times n} \tag{5.18}$$

Der Ablaufvektor $\boldsymbol{\tau}$ (engl.: scheduling vector) ist ein Zeilenvektor. Die Elemente des Vektors sollen so gewählt werden, dass eine lückenlose Abarbeitung der Elemente der Partition gewährleistet ist. Die Anzahl der Zeitpunkte T zur Abarbeitung einer LS-Partition ergibt sich aus der Größe $\mathbf{\Theta}^S$ der Partition mit $T = \det(\mathbf{\Theta}^S)$.

Die Reihenfolge der Abarbeitung der Elemente der Partition wird durch den Ablaufvektor $\boldsymbol{\tau}$ festgelegt. Die lineare Ablaufplanung gestattet es nur, die zeitliche Abfolge der Abarbeitung der Raumrichtungen innerhalb einer Partition festzulegen und die aufsteigende oder absteigene Abarbeitung der Elemente jeder Raumrichtung vorzugeben. Hierfür wurde in [Sie03] der Abfolgevektor $\boldsymbol{\alpha}$ (engl.: succession vector) eingeführt.

Definition 5.2 (Abfolgevektor $\boldsymbol{\alpha}$)
Der Vektor $\boldsymbol{\alpha} \in \mathbb{Z}^{1\times n}$ legt die zeitliche Abfolge der n Raumrichtungen innerhalb einer Partition ($\mathbf{\Theta} \in \mathbb{Z}^{n\times n}$) fest, wobei α_k, $(1 \leq k \leq n)$ die k-te Raumrichtung spezifiziert. Die erste Richtung ist gegeben durch α_1 und die letzte Richtung durch α_n. Die Richtung der Abarbeitung der Partitionselemente einer Raumrichtung wird durch das Vorzeichen von α_k angegeben. Bei positivem Vorzeichen erfolgt eine aufsteigende Abarbeitung und bei negativen Vorzeichen ein absteigende Abarbeitung.

Beispiel 5.6 (Verschiedene Abfolgevektoren einer LS-Partition)
Die Iterationen $\boldsymbol{\kappa}^S = \begin{pmatrix} \kappa_1^S \\ \kappa_2^S \end{pmatrix}$ der Partition $\mathbf{\Theta}^S = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 7 \end{smallmatrix}\right)$ sollen sequenzialisiert werden. Es gibt 8 Möglichkeiten der Abarbeitung:

$$\boldsymbol{\alpha} \in \left\{ \begin{pmatrix} 1 & 2 \end{pmatrix}, \begin{pmatrix} -1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & -2 \end{pmatrix}, \begin{pmatrix} -1 & -2 \end{pmatrix} \begin{pmatrix} 2 & 1 \end{pmatrix}, \begin{pmatrix} -2 & 1 \end{pmatrix}, \begin{pmatrix} 2 & -1 \end{pmatrix}, \begin{pmatrix} -2 & -1 \end{pmatrix} \right\}.$$

Im Allgemeinen ergibt sich die Anzahl der möglichen Abfolgevektoren für eine Partition $\mathbf{\Theta}^S \in \mathbb{Z}^{n\times n}$ durch $N = n! \cdot 2^n$. Wenn eine Partition aber in einer Raumrichtung nur ein Element besitzt ($\vartheta_j^S = 1$), so gibt es für diese Richtung keine Auswahl der Abarbeitung. Somit ergibt sich die effektive Anzahl an Abfolgevektoren durch: $N = m! \cdot 2^m$, $m = \sum_{j=1}^{n} \operatorname{sign}(\vartheta_j^S - 1)$.

Der Ablaufvektor $\boldsymbol{\tau} \in \mathbb{Z}^{1\times n}$ für einen lückenlosen Ablauf lässt sich bei gegebener Partitionsgröße $\boldsymbol{\Theta}^S$ aus dem Abfolgevektor wie folgt bestimmen:

$$\boldsymbol{\tau} = \begin{cases} \tau_{|\alpha_j|} = \text{sign}(\alpha_j) & \vartheta^S_{|\alpha_j|} > 1 \wedge j = 1 \\ \tau_{|\alpha_j|} = \text{sign}(\alpha_j) \cdot \prod_{k=1}^{j-1} \vartheta^S_{|\alpha_k|} & \vartheta^S_{|\alpha_j|} > 1 \wedge 1 < j \leq n \\ \tau_{|\alpha_j|} = 0 & \text{sonst.} \end{cases} \tag{5.19}$$

Beispiel 5.7 (Bestimmung der Ablaufvektoren aus Abfolgevektoren)
Für die Abfolgevektoren $\boldsymbol{\alpha}$ aus Beispiel 5.6 ergeben sich die folgenden Ablaufvektoren:

$$\boldsymbol{\tau} \in \left\{ \begin{pmatrix} 1 & 4 \end{pmatrix}, \begin{pmatrix} -1 & 4 \end{pmatrix}, \begin{pmatrix} 1 & -4 \end{pmatrix}, \begin{pmatrix} -1 & -4 \end{pmatrix} \begin{pmatrix} 7 & 1 \end{pmatrix}, \begin{pmatrix} -7 & 1 \end{pmatrix}, \begin{pmatrix} 7 & -1 \end{pmatrix}, \begin{pmatrix} -7 & -1 \end{pmatrix} \right\}.$$

Nicht immer können die Iterationen einer Partitionen lückenlos abgearbeitet werden. Aus diesem Grund führen wir die Leerlaufverzögerung $\boldsymbol{\iota} \in \mathbb{Z}^{1\times n}$ (engl.: idle offset) ein.

Definition 5.3 (Leerlaufverzögerung $\boldsymbol{\iota}$)
Die Leerlaufverzögerung ι_j gibt für jede Abarbeitungsrichtung α_j die Anzahl der Leertakte an, die nach einer Iteration bei Veränderung des Iterators $\kappa^S_{|\alpha_j|}$ eingefügt werden.

Der Ablaufvektor $\boldsymbol{\tau} \in \mathbb{Z}^{1\times n}$ für einen lückenhaften Ablauf wird nun wie folgt bestimmt:

$$\boldsymbol{\tau} = \begin{cases} \tau_{|\alpha_j|} = \text{sign}(\alpha_j)(1 + \frac{\iota_j}{q}) & \vartheta^S_{|\alpha_j|} > 1 \wedge j = 1, \\ \tau_{|\alpha_j|} = \text{sign}(\alpha_j) \left(\prod_{k=1}^{j-1} (1 + \frac{\iota_k}{q}) \cdot \vartheta^S_{|\alpha_k|} + \frac{\iota_j}{q} \right) & \vartheta^S_{|\alpha_j|} > 1 \wedge 1 < j \leq n, \\ \tau_{|\alpha_j|} = 0 & \text{sonst.} \end{cases} \tag{5.20}$$

Der Nenner q des Bruchs $\frac{\iota_j}{q}$ dient der taktgenauen Verzögerung bei einem Iterationsintervall $\lambda > 1$. Das Iterationsintervall λ wird im nächsten Abschnitt vorgestellt. In diesem Abschnitt setzen wir den Nenner q vorerst $q = 1$.

Die Partitionsperiode T, welche die Abarbeitungsdauer aller Iterationen der Partition beschreibt, lässt sich durch

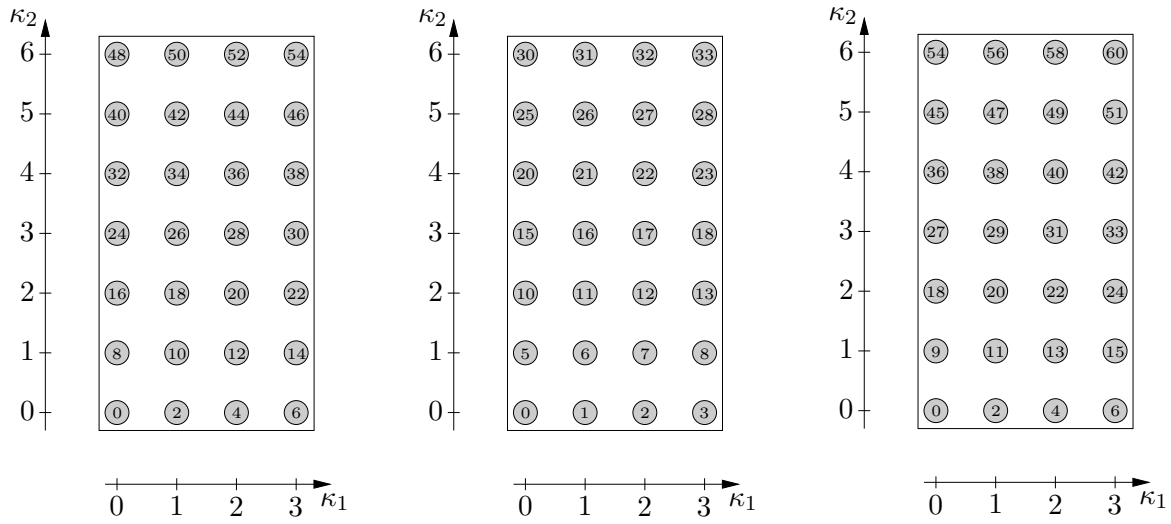
$$T = \begin{cases} \max_{j=1}^{n} |\tau_j \cdot \vartheta^S_j|, & \text{wenn} \quad \boldsymbol{\tau} \neq \mathbf{0}, \\ 1, & \text{sonst,} \end{cases} \tag{5.21}$$

bestimmen.

Beispiel 5.8 (Verschiedene Leerlaufverzögerungen in einer Partition)
Für den Abfolgevektor $\boldsymbol{\alpha} = \begin{pmatrix} 1 & 2 \end{pmatrix}$ aus Beispiel 5.6 sollen vier Leerlaufverzögerungen $\boldsymbol{\iota}$ betrachtet werden. Der Wert für Nenner q beträgt $q = 1$.

$$\begin{aligned} \boldsymbol{\iota}_1 &= \begin{pmatrix} 0 & 0 \end{pmatrix} & \Rightarrow \quad \boldsymbol{\tau}_1 &= \begin{pmatrix} 1 & 4 \end{pmatrix} & \Rightarrow \quad T_1 &= 4 \cdot 7 = 28 \\ \boldsymbol{\iota}_2 &= \begin{pmatrix} 1 & 0 \end{pmatrix} & \Rightarrow \quad \boldsymbol{\tau}_2 &= \begin{pmatrix} 2 & 8 \end{pmatrix} & \Rightarrow \quad T_2 &= 8 \cdot 7 = 56 \\ \boldsymbol{\iota}_3 &= \begin{pmatrix} 0 & 1 \end{pmatrix} & \Rightarrow \quad \boldsymbol{\tau}_3 &= \begin{pmatrix} 1 & 5 \end{pmatrix} & \Rightarrow \quad T_3 &= 5 \cdot 7 = 35 \\ \boldsymbol{\iota}_4 &= \begin{pmatrix} 1 & 1 \end{pmatrix} & \Rightarrow \quad \boldsymbol{\tau}_4 &= \begin{pmatrix} 2 & 9 \end{pmatrix} & \Rightarrow \quad T_4 &= 9 \cdot 7 = 63 \end{aligned}$$

Mit Leerlaufverzögerung $\boldsymbol{\iota}_1$ erhalten wir die lückenlose Abarbeitung, wie in Beispiel 5.7 bereits vorgestellt. Abbildung 5.5 zeigt die Abarbeitungszeiten für die Iterationen einer Partition der Größe $\boldsymbol{\Theta}^S = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 7 \end{smallmatrix}\right)$ mit den Leerlaufverzögerungen $\boldsymbol{\iota}_2$, $\boldsymbol{\iota}_3$ und $\boldsymbol{\iota}_4$.



(a) $\boldsymbol{\iota}_2 = (1\ 0) \Rightarrow \boldsymbol{\tau}_2 = (2\ 8)$ (b) $\boldsymbol{\iota}_3 = (0\ 1) \Rightarrow \boldsymbol{\tau}_3 = (1\ 5)$ (c) $\boldsymbol{\iota}_4 = (1\ 1) \Rightarrow \boldsymbol{\tau}_4 = (2\ 9)$

Abbildung 5.5: zeitlicher Ablauf bei verschiedenen Leerlaufverzögerungen

5.2.2 Verzögerungsvektor $\boldsymbol{\tau}^{\text{offs}}$

Bei der gleichzeitigen Abarbeitung der Elemente der Partition $\boldsymbol{\Theta}^P$ kann es zu zeitlichen Konflikten aufgrund von Datenabhängigkeiten zwischen den Elementen der Partition kommen[1]. Um diese zu vermeiden, muss der Verzögerungsvektor $\boldsymbol{\tau}^{\text{offs}}$ eingeführt werden:

$$t = \boldsymbol{\tau}^{\text{offs}} \cdot \boldsymbol{\kappa}^P, \qquad \boldsymbol{\tau}^{\text{offs}} = \mathbb{Z}^{1\times n}. \tag{5.22}$$

Dieser Verzögerungsvektor verursacht eine konstante zeitliche Verzögerung für die Abarbeitung der Elemente der Partition.

Beispiel 5.9 (Verzögerungsvektor)
Im Beispiel 5.3 wurde eine modifizierte Copartitionierung mit $\boldsymbol{\Theta}^S = \left(\begin{smallmatrix}2&0\\0&1\end{smallmatrix}\right)$ *und* $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix}2&0\\0&1\end{smallmatrix}\right)$ *vorgestellt. In Abbildung 5.2 wurden in den Iterationen die Zeitpunkte der Abarbeitung angegeben. Nun sein diese Abarbeitung aufgrund einer Datenabhängigkeit mit Abhängigkeitsvektor* $\mathbf{d} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *nicht möglich. Mit dem Verzögerungsvektor* $\boldsymbol{\tau}^{\text{offs}} = \begin{pmatrix}2 & 0\end{pmatrix}$ *kann aber der Beginn der Abarbeitung von* $\mathbf{i} = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$ *so verzögert werden, dass die Abarbeitung erst nach* $\mathbf{i} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *beginnt. Abbildung 5.6 zeigt den copartitionierten Iterationsraum mit dem veränderten Ablaufplan.*

5.2.3 Iterationsintervall und teilzuweisungsabhängige Verzögerung

Bisher wurde davon ausgegangen, dass alle Teilzuweisungen s_t eines Algorithmus, die in einer Iteration $\mathbf{i} \in \mathcal{I}$ existieren, gleichzeitig innerhalb einer Zeiteinheit ausgeführt werden. Es soll nun für jede Teilzuweisung s_t ein individueller Startzeitpunkt $t_{0,s_t} \in \mathbb{N}$ festgelegt

[1] Eine Variable muss erst bestimmt sein, bevor sie verwendet werden kann.

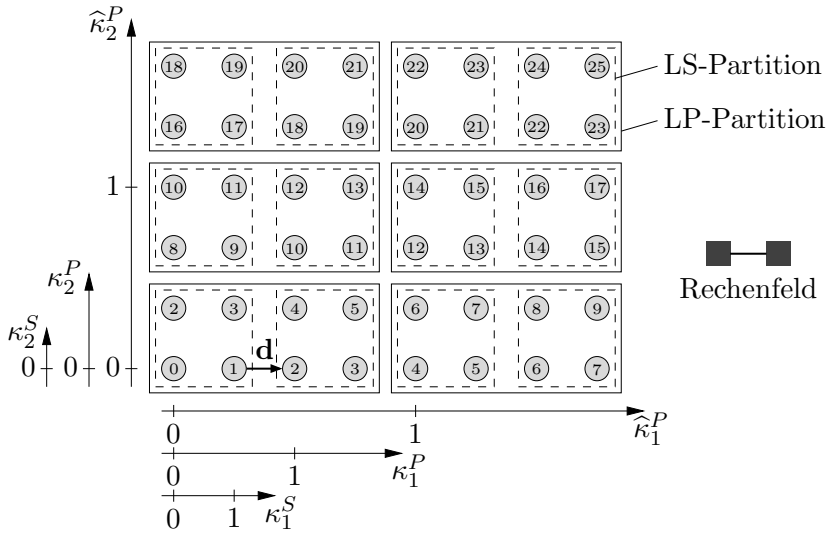


Abbildung 5.6: Modifizierte Copartitionierung mit Verzögerungsvektor $\boldsymbol{\tau}^{\text{offs}} = \begin{pmatrix} 2 & 0 \end{pmatrix}$

werden, zu dem bezüglich des Iterationsintervalls λ die Berechnung der Teilzuweisung s_t beginnt. Das teilzweisungsunabhängige Iterationsintervall $\lambda \in N^+$ gibt den zeitlichen Abstand der Berechnung der Teilzuweisung s_t zweier zeitlich nacheinander folgender Iterationen an:

$$t = \lambda \cdot \boldsymbol{\tau}^S \cdot \boldsymbol{\kappa}^S + t_{0,s_t}, \qquad t_{0,s_t} \in \mathbb{N}. \tag{5.23}$$

Durch diese Beschreibung ist es möglich, individuelle Latenzen $l_{m(s_t)}$ und Verzögerungen $o_{m(s_t)}$ für die Operationen F_{s_t} der Teilzuweisungen s_t, verursacht durch die Funktionseinheiten y$m \in \mathcal{M}$, zu berücksichtigen. Auch Funktionspipelining ($t_{0,s_t} > \lambda$) kann damit realisiert werden. Das heißt, die Abarbeitung der Zuweisungen einer Iteration kann mehrere Iterationsintervalle lang dauern, wobei in einem Iterationsintervall Zuweisungen von verschiedenen Iterationen in einem Prozessorelement verarbeitet werden.

Beispiel 5.10 (Lokaler Ablaufplan)
Ein Algorithmus hat vier Zuweisungen $s_1, \ldots, s_4$ mit jeweils einer Teilzuweisung. Die vier Operationen $F_{1_1}, \ldots, F_{4_1}$ der Teilzuweisungen $s_{1_1}, \ldots, s_{4_1}$ sollen in zwei Funktionseinheiten m_1 und m_2 abgearbeitet werden, wobei s_{1_1} und s_{3_1} in der Funktionseinheit m_1 und s_{2_1} und s_{4_1} in der Funktionseinheit m_2 abgebildet werden. Die normierten Zeiten für die Abarbeitung der Teilzuweisungen in den Funktionseinheiten sind:

$$\begin{aligned} l_{m_1(s_{1_1})} &= l_1 = 4, & o_{m_1(s_{1_1})} &= o_1 = 4, & l_{m_2(s_{2_1})} &= l_2 = 3, & o_{m_2(s_{2_1})} &= o_2 = 3, \\ l_{m_1(s_{3_1})} &= l_3 = 5, & o_{m_1(s_{3_1})} &= o_3 = 4, & l_{m_2(s_{4_1})} &= l_4 = 4, & o_{m_2(s_{4_1})} &= o_4 = 4. \end{aligned}$$

Abbildung 5.7 zeigt eine mögliche Realisierung zur Abarbeitung der Teilzuweisungen auf den beiden Funktionseinheiten. Die im gleichen Grauton gezeichneten Felder stellen die Latenzen der Teilzuweisungen einer Iteration dar. Die zwischen den hellgrau gezeichneten (Teil-)Zuweisungen bestehenden Datenabhängigkeiten wurden eingezeichnet. Das

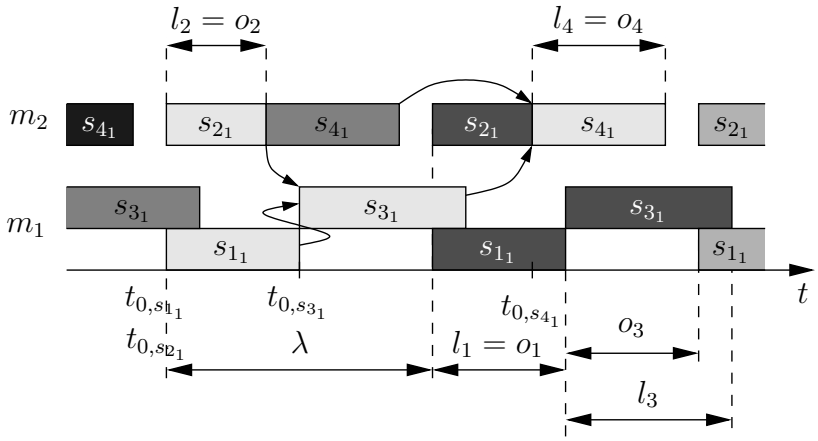


Abbildung 5.7: Iterationsintervall λ und teilzuweisungsabhängige Verzögerung t_{0,s_t}

Iterationsintervall hat die Länge $\lambda = 8$. *Die teilzuweisungsabhängigen Verzögerungen lauten:*

$$t_{0,s_{1_1}} = 0, \qquad t_{0,s_{2_1}} = 0, \qquad t_{0,s_{3_1}} = 4, \qquad t_{0,s_{4_1}} = 11.$$

Die Berechnung der Teilzuweisung s_{4_1} *erfolgt nicht im aktuellen Iterationsintervall, sondern erst im nachfolgenden* ($t_{0,s_{4_1}} > \lambda$). *Der Startzeitpunkt für Teilzuweisung* s_{3_1} *ist zwar im aktuellen Iterationsintervall* ($t_{0,s_{3_1}} < \lambda$), *doch die Berechnung ist erst im nachfolgenden Iterationsintervall beendet* ($t_{0,s_{3_1}} + l_3 > \lambda$). *Wäre kein Funktionspipelining möglich, so beträge das minimale Iterationsintervall* $\lambda_{\min} = l_1 + l_3 + l_4 = 13$.

Besitzt eine Zuweisung s mehrere Teilzuweisungen s_t, so ist es grundsätzlich möglich, dass die Bestimmung der Funktionen F_{s_t} einer Zuweisung s in unterschiedlichen Modulen $m \in \mathcal{M}$ erfolgt oder zu verschiedenen Zeitpunkten $t_{0,s_t} + l_{m(s_t)}$ abgeschlossen ist. Deshalb führen wir die Zeit

$$t_{v,s} = \max_{1 \le t \le T_s} t_{0,s_t} + l_{m(s_t)}, \tag{5.24}$$

ein, welche den frühesten Zeitpunkt der Verfügbarkeit der Variable $y_s[\mathbf{i}]$ bezüglich des Iterationsintervalls λ angibt. Eine Abarbeitung der Teilzuweisung s'_t, welche die Variable $y_s[\mathbf{i}]$ in der gleichen Iteration benötigt, kann also frühestens $t_{0,s'_t} = t_{v,s}$ beginnen. Abbildung 5.8 illustriert das.

In Abbildung 5.8 ist die Latenz $l_{m(s_t)}$ immer kleiner als die Verzögerung $o_{m(s_t)}$. Man beachte, dass die Teilzuweisungen s_1 und s_2 scheinbar das gleiche Modul m_1 zur gleichen Zeit benutzen. Da die Iterationsräume der beiden Teilzuweisungen disjunkt sind, wird aber abhängig von der Iteration nur eine der beiden Teilzuweisungen abgearbeitet.

Bei der Bestimmung des Iterationsintervall λ, der teilzuweisungsabhängigen Verzögerungen t_{0,s_t} und der Zuordnung der Teilzuweisungen ist darauf zu achten, dass die Kausalitätsbedingung und die Ressourcenbeschränkung eingehalten wird. Mit der Kausalitätsbedingung wird sichergestellt, dass keine Variable verwendet wird bevor sie berechnet

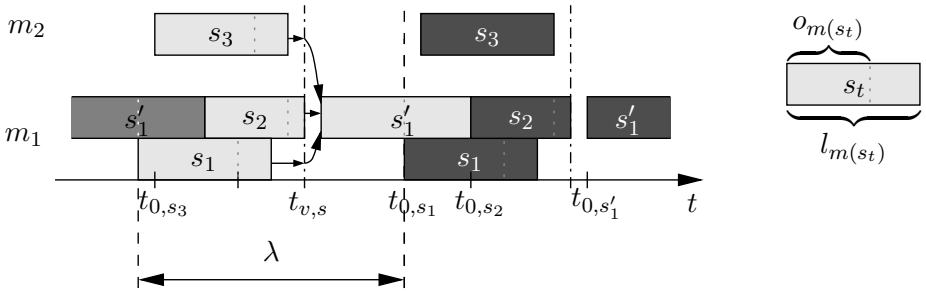


Abbildung 5.8: Iterationsintervall λ mit Zuweisung s mit mehreren Teilzuweisungen s_t

wurde und die Ressourcenbeschränkung gewährleistet, dass nicht mehrere Teilzuweisungen gleichzeitig in einem Modul abgearbeitet werden. Die Kausalitätsbeschränkung wird noch einmal ausführlich in Abschnitt 5.2.6 vorgestellt.

5.2.4 Ablaufplanung für modifizierte Copartitionierung

Für einen copartitionierten Iterationsraum $\mathcal{I}^\kappa$ lautet die Ablaufplanung wie folgt:

$$t = t(\mathbf{i}^\kappa, s_t) = \boldsymbol{\tau}^\kappa \mathbf{i}^\kappa + t_{0,s_t}, \tag{5.25}$$

$$= \begin{pmatrix} \lambda \cdot \boldsymbol{\tau}^S & \boldsymbol{\tau}^{\text{offs}} & \lambda \cdot T^S \cdot \widehat{\boldsymbol{\tau}}^P \end{pmatrix} \begin{pmatrix} \boldsymbol{\kappa}^S \\ \boldsymbol{\kappa}^P \\ \widehat{\boldsymbol{\kappa}}^P \end{pmatrix} + t_{0,s_t}, \tag{5.26}$$

$$= \lambda(\boldsymbol{\tau}^S \cdot \boldsymbol{\kappa}^S + T^S \cdot \widehat{\boldsymbol{\tau}}^P \cdot \widehat{\boldsymbol{\kappa}}^P) + \boldsymbol{\tau}^{\text{offs}} \cdot \boldsymbol{\kappa}^P + t_{0,s_t}, \tag{5.27}$$

wobei T^S die Partitionsperiode der LS-Partition ist.

Es erfolgt eine seqenzielle Abarbeitung sowohl innerhalb der LS-Partition, gegeben durch $\boldsymbol{\Theta}^S$, als auch der LP-Partitionen selber. Für die Bestimmung des Ablaufvektors $\boldsymbol{\tau}^S$ kann die in (5.20) präsentierte Methode genutzt werden, wobei für eine taktgenaue Leerlaufverzögerung $\boldsymbol{\iota}^S$ der Nenner $q^S = \lambda$ sein muss. Für die Ermittlung von $\widehat{\boldsymbol{\tau}}^P$ muss erst die Hilfsmatrix $\widehat{\boldsymbol{\Theta}}^P = \text{diag}(\widehat{\boldsymbol{\vartheta}}^P) \in \mathbb{N}^{n \times n}$ bestimmt werden:

$$\widehat{\boldsymbol{\Theta}}^P = \text{diag}(\widehat{\boldsymbol{\vartheta}}^P), \qquad \widehat{\vartheta}_j^P = \max_{\forall \widehat{\boldsymbol{\kappa}}_1^P, \widehat{\boldsymbol{\kappa}}_2 \in \widehat{\mathcal{K}}} \widehat{\kappa}_{1,j}^P - \widehat{\kappa}_{2,j}^P + 1, \qquad \widehat{\vartheta}_j^P \in \mathbb{N}^+, \quad 1 \leq j \leq n. \tag{5.28}$$

Diese Matrix beschreibt die Größe eines Hyperquaders, der den Raum $\widehat{\mathcal{K}}$ einschließt. Mit der Matrix $\widehat{\boldsymbol{\Theta}}^P$ lässt sich nun der Ablaufvektor $\widehat{\boldsymbol{\tau}}^P$ entsprechend (5.20) ermitteln. Für eine taktgenaue Leerlaufverzögerung $\widehat{\boldsymbol{\iota}}^P$ muss der Devisor $\widehat{q}^P = \lambda \cdot T^S$ sein. Die äußere sequenzielle Abarbeitung $\widehat{\boldsymbol{\tau}}^P \cdot \widehat{\boldsymbol{\kappa}}^P$ wird um die Abarbeitungsdauer T^S aller Iterationen der LS-Partition gestreckt, um die gewünschte sequenzielle Abarbeitung zu erhalten. Die Ermittlung der Partitionsperiode T^S erfolgt mit (5.21).

Alle seqenziellen Abarbeitungszeitpunkte werden um das Iterationsintervall λ gedehnt, um zusammen mit der zuweisungsabhängigen Verzögerung $t_{0,s}$ einen zuweisungs- und

ressourcenabhängigen Ablaufplan innerhalb des Iterationsintervalls beschreiben zu können.

Abschließend enthält (5.25) noch die Verzögerung $\boldsymbol{\tau}^{\text{offs}} \cdot \boldsymbol{\kappa}^P$, mit der Datenabhängigkeitskonflikte innerhalb der LP-Partition vermieden werden können.

Beispiel 5.11 (Ablaufplan für copartitionierten Iterationsraum)
Für die im Beispiel 5.9 angegebene modifizierte Copartitionierung ($\boldsymbol{\Theta}^S = \left(\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right)$ und $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix}2&0\\0&1\end{smallmatrix}\right)$) des Iterationsraums $\mathcal{I} = \{\mathbf{i} = \left(\begin{smallmatrix}i_1\\i_2\end{smallmatrix}\right) \mid 0 \leq i_1 < 8 \wedge 0 \leq i_2 < 6\}$ mit dem Verzögerungsvektor $\boldsymbol{\kappa}^{\text{offs}} = \begin{pmatrix}2 & 0\end{pmatrix}$ und der in Abbildung 5.6 angegebenen Abarbeitungsreihenfolge soll der Ablaufplan angegeben werden.

Die Abfolgevektoren und die Leerlaufverzögerungen für die Elemente der LS-Partition und die LP-Partitionen lauten:

$$\boldsymbol{\alpha}^S = \begin{pmatrix}1 & 2\end{pmatrix}, \qquad \boldsymbol{\iota}^S = \mathbf{0}, \qquad \widehat{\boldsymbol{\alpha}}^P = \begin{pmatrix}1 & 2\end{pmatrix}, \qquad \widehat{\boldsymbol{\iota}}^P = \mathbf{0}.$$

Mit der Partitionsgröße $\boldsymbol{\Theta}^S = \left(\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right)$ ergibt sich der Ablaufvektor $\boldsymbol{\tau}^S = \begin{pmatrix}1 & 2\end{pmatrix}$. Der Ablaufvektor $\widehat{\boldsymbol{\tau}}^P$ hat die Größe $\widehat{\boldsymbol{\tau}}^P = \begin{pmatrix}1 & 2\end{pmatrix}$ unter Verwendung der Hilfsmatrix $\widehat{\boldsymbol{\Theta}} = \left(\begin{smallmatrix}2&0\\0&3\end{smallmatrix}\right)$.

Werden in jeder Iteration $\mathbf{i} \in \mathcal{I}$ die vier Zuweisungen $s_1, \ldots, s_4$ entsprechend der in Beispiel 5.10 präsentierten lokalen Abarbeitung berechnet, so ergibt sich der folgende Ablaufplan:

$$t(\mathbf{i}^\kappa, s_t) = 8\left(\begin{pmatrix}1 & 2\end{pmatrix}\boldsymbol{\kappa}^S + 4\begin{pmatrix}1 & 2\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P\right) + 8 \cdot \begin{pmatrix}2 & 0\end{pmatrix}\boldsymbol{\kappa}^P + t_{0,s_t}, \tag{5.29}$$

$$t(\mathbf{i}^\kappa, s_t) = \begin{pmatrix}8 & 16 & 16 & 0 & 32 & 64\end{pmatrix} \cdot \mathbf{i}^\kappa + t_{0,s_t}, \tag{5.30}$$

wobei das Iterationsintervall die Länge $\lambda = 8$ hat und die teilzuweisungsabhängigen Verzögerungen t_{0,s_t} die in Beispiel 5.10 angegebenen Werte haben.

5.2.5 Ablaufplanung für mehrstufige modifizierte Copartitionierung

Für den allgemeinen Fall eines p-fach mehrstufig copartitionierten Iterationsraums $\mathcal{I}^\kappa$ gilt:

$$t = t(\mathbf{i}^\kappa, s_t) = \boldsymbol{\tau}^\kappa \cdot \mathbf{i}^\kappa + t_{0,s_t} \tag{5.31}$$

$$= \Big(\lambda \cdot \boldsymbol{\tau}^{S_1}, \quad \boldsymbol{\tau}^{\text{offs},1}, \quad \lambda \cdot \textstyle\prod_{l=1}^{1}(T^{S_l})\boldsymbol{\tau}^{S_2}, \quad \boldsymbol{\tau}^{\text{offs},2}, \quad \ldots$$

$$\ldots \quad \lambda \cdot \textstyle\prod_{l=1}^{p-1}(T^{S_l})\boldsymbol{\tau}^{S_p}, \quad \boldsymbol{\tau}^{\text{offs},p}, \quad \lambda \cdot \textstyle\prod_{l=1}^{p}(T^{S_l})\widehat{\boldsymbol{\tau}}^{P_p}\Big) \begin{pmatrix} \boldsymbol{\kappa}^{S_1} \\ \boldsymbol{\kappa}^{P_1} \\ \boldsymbol{\kappa}^{S_2} \\ \boldsymbol{\kappa}^{P_2} \\ \vdots \\ \boldsymbol{\kappa}^{S_p} \\ \boldsymbol{\kappa}^{P_p} \\ \widehat{\boldsymbol{\kappa}}^{P_p} \end{pmatrix} + t_{0,s_t}, \tag{5.32}$$

$$t = \lambda \left(\sum_{k=1}^{p} \left(\prod_{l=1}^{k-1} T^{S_l} \right) \boldsymbol{\tau}^{S_k} \cdot \boldsymbol{\kappa}^{S_k} + \left(\prod_{l=1}^{p} T^{S_l} \right) \cdot \widehat{\boldsymbol{\tau}}^{P_p} \cdot \widehat{\boldsymbol{\kappa}}^{P_p} \right) + \sum_{k=1}^{p} \boldsymbol{\tau}^{\mathrm{offs},k} \cdot \boldsymbol{\kappa}^{P_k} + t_{0,s_t}. \quad (5.33)$$

Die Ablaufvektoren $\boldsymbol{\tau}^{S_k}$ der LS-Partitionen $\boldsymbol{\Theta}^{S_k}$, $1 \leq k \leq p$ lassen sich mit (5.20) ermitteln, wobei für die Devisoren $q^{S_k} = \lambda \cdot \prod_{l=1}^{k-1} T^{S_l}$ gilt. Für eine hierarchisch sequenzielle Abarbeitung müssen die Zeitpunkte der k-ten LS-Partition mit der Abarbeitungsdauer der 1-ten bis $(k-1)$-ten LS-Partition ($\prod_{l=1}^{k-1} T^{S_l}$) multipliziert werden.

Der Ablaufvektor $\widehat{\boldsymbol{\tau}}^{P_p}$ kann mit Hilfe der Matrix $\widehat{\boldsymbol{\Theta}}^{P_p}$, welche entsprechend (5.28) bestimmt wird, berechnet werden. Für den Devisor $\widehat{q}^{P_p}$ gilt $\widehat{q}^{P_p} = \lambda \cdot \prod_{l=1}^{p} T^{S_l}$. Die Zeitpunkte der äußeren sequenziellen Abarbeitung müssen mit den Abarbeitungsdauern T^{S_l} aller p LS-Partitionen multiplizert werden.

Für einen zuweisungs- und ressourcenabhängigen Ablaufplan werden, wie bei der einfachen modifizierten Copartitionierung, alle sequenziellen Abarbeitungszeitpunkte um das Iterationsintervall λ gestreckt und die zuweisungsabhängigen Verzögerungen $t_{0,s}$ hinzugefügt.

Abschließend existiert für jede LP-Partition eine Verzögerung $\boldsymbol{\tau}^{\mathrm{offs},k} \cdot \boldsymbol{\kappa}^{P_k}$.

Beispiel 5.12 (Ablaufplan für mehrstufig copartitionierten Iterationsraum)
Für den in Beispiel 5.4 vorgestellten zweistufig copartitionierten Iterationsraum $\mathcal{I}^{\kappa}$ soll ein Ablaufplan erstellt werden. In dem Iterationsraum soll zusätzlich ein Abhängigkeitsvektor $\mathbf{d} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ existieren, welcher die gleichzeitige Abarbeitung der Iterationen einer Zeile des Iterationsraums verbietet.

Für die LS-Partitionen wurden die Abfolgevektoren $\boldsymbol{\alpha}^{S_1} = \begin{pmatrix}1 & 2\end{pmatrix}$ und $\boldsymbol{\alpha}^{S_2} = \begin{pmatrix}2 & 1\end{pmatrix}$ festgelegt. Mit den Partitionierungsmatrizen $\boldsymbol{\Theta}^{S_1}$ und $\boldsymbol{\Theta}^{S_2}$ aus Beispiel 5.4 und den Leerlaufverzögerungen $\boldsymbol{\iota}^{S_1} = \boldsymbol{\iota}^{S_2} = \widehat{\boldsymbol{\iota}}^{P_2} = \mathbf{0}$ erhalten wir die Ablaufvektoren $\boldsymbol{\tau}^{S_1} = \begin{pmatrix}1 & 0\end{pmatrix}$ und $\boldsymbol{\tau}^{S_2} = \begin{pmatrix}0 & 1\end{pmatrix}$, sowie die Partitionsperioden $T^{S_1} = 2$ und $T^{S_2} = 3$. Der Abfolgevektor für die Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ beträgt $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix}1 & 2\end{pmatrix}$. Unter Zuhilfenahme der Hilfsmatrix $\widehat{\boldsymbol{\Theta}}^{P_2} = \left(\begin{smallmatrix}1 & 0\\ 0 & 2\end{smallmatrix}\right)$ erhalten wir den Ablaufvektor $\widehat{\boldsymbol{\tau}}^{P_2} = \begin{pmatrix}0 & 1\end{pmatrix}$.

Da in dem Iterationsraum eine Datenabhängigkeit $\mathbf{d} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ vorhanden ist, müssen entgegen dem Beispiel 5.4 die Iterationen mit $\boldsymbol{\kappa}^{P_1} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ jeweils um zwei Zeiteinheiten und die Iterationen mit $\boldsymbol{\kappa}^{P_2} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ jeweils um vier Zeiteinheiten verzögert werden. Die dafür notwendigen Verzögerungsvektoren lauten $\boldsymbol{\tau}^{\mathrm{offs},1} = \begin{pmatrix}2 & 0\end{pmatrix}$ bzw. $\boldsymbol{\tau}^{\mathrm{offs},2} = \begin{pmatrix}4 & 0\end{pmatrix}$.

Abbildung 5.9 zeigt den bisher beschriebenen Ablaufplan. Die in den Iterationen angegebenen Werte sind die Startzeitpunkte für die Abarbeitung dieser Iteration. Zwei Abhängigkeitsvektoren $\mathbf{d}$ wurden examplarisch in die Abbildung eingezeichntet, welche die oben angegebenen Verzögerungsvektoren notwendig machen.

Soll in jeder Iteration der lokale Ablaufplan entsprechend Beispiel 5.10 realisiert werden,

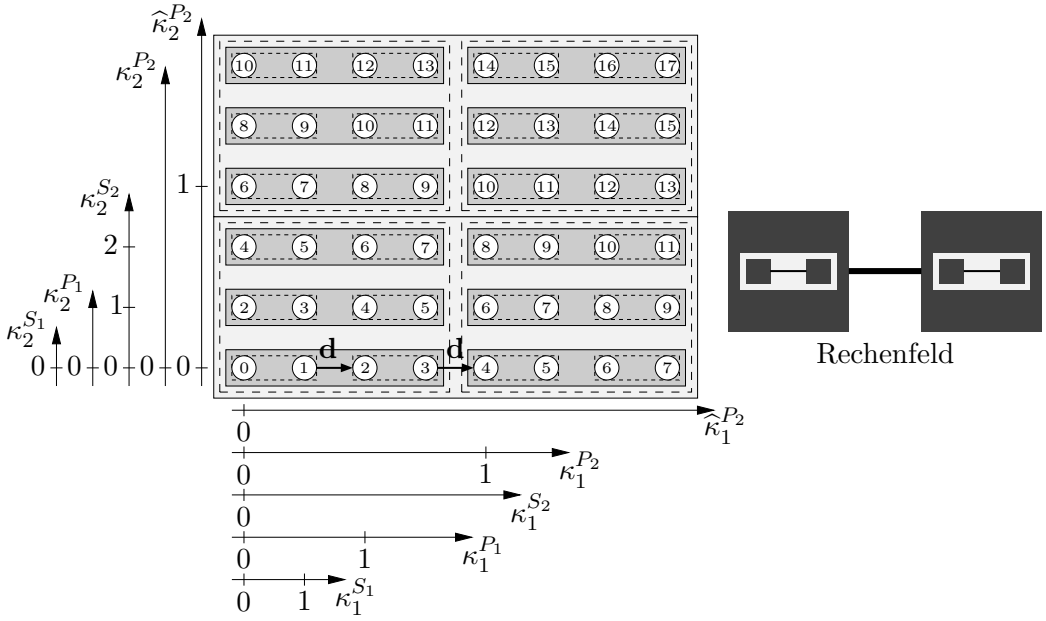


Abbildung 5.9: Ablaufplan für zweistufig modifizierte Copartitionierung

so lautet der gesamte Ablaufplan:

$$t(\mathbf{i}^{\kappa}, s_t) = 8\left(\begin{pmatrix}1 & 0\end{pmatrix}\boldsymbol{\kappa}^{S_1} + 2\begin{pmatrix}0 & 1\end{pmatrix}\boldsymbol{\kappa}^{S_2} + 2\cdot 3\begin{pmatrix}0 & 1\end{pmatrix}\widehat{\boldsymbol{\kappa}}^{P_2}\right) + 8\cdot\begin{pmatrix}2 & 0\end{pmatrix}\boldsymbol{\kappa}^{P_1} + 8\cdot\begin{pmatrix}4 & 0\end{pmatrix}\boldsymbol{\kappa}^{P_2} + t_{0,s_t}, \quad (5.34)$$

$$= \begin{pmatrix}8 & 0 & 16 & 0 & 0 & 16 & 32 & 0 & 0 & 48\end{pmatrix}\cdot\mathbf{i}^{\kappa} + t_{0,s_t}, \quad (5.35)$$

wobei das Iterationsintervall $\lambda = 8$ ist und die teilzuweisungsabhängigen Verzögerungen t_{0,s_t} die in diesem Beispiel angegebenen Werte besitzen.

5.2.6 Kausalitätsbedingung

Bereits bei der Vorstellung des Verzögerungsvektors $\boldsymbol{\tau}^{\text{offs}}$ wurde erwähnt, dass die Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ eines Algorithmus die Ablaufplanung für diesen Algorithmus beschränken können. Diese Einschränkung ist unter der Bezeichnung *Kausalitätsbedingung* bekannt. Sie besagt im Allgemeinen, dass eine Variable $y_s[.]$ erst verwendet werden kann, wenn sie vorher berechnet worden ist.

Definition 5.4 (Kausalitätsbedingung)
Alle Teilzuweisungen s'_t der Zuweisung s' der Iteration $\mathbf{i}_\sigma$ müssen zeitlich vor der Teilzuweisung s_t in Iteration $\mathbf{i}_\delta$ ausgeführt werden, wenn ein Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{i}_\delta - \mathbf{i}_\sigma$ für die Datenabhängigkeit e, mit $\sigma(e) = s'$ und $\delta(e) = s_t$ existiert.

Bei einem mehrstufig copartitionierten Algorithmus können wir für jede Teilzuweisung s_t den Startzeitpunkt der Abarbeitung dieser Teilzuweisung mittels des Ablaufplans $t(\mathbf{i}^{\kappa}, s_t)$

bestimmen. Da der Ablaufplan $t(\mathbf{i}^\kappa, s_t)$ eine lineare Funktion ist und die Datenabhängigkeiten $e^\kappa \in \mathcal{E}^\kappa$ des mehrstufig copartitionierten Algorithmus konstante Abhängigkeitsvektoren $\mathbf{d}(e^\kappa)$ besitzen, muss zur Erfüllung der Kausalitätsbedingung für alle Abhängigkeitsvektor $\mathbf{d}(e^\kappa)$, $e^\kappa \in \mathcal{E}^\kappa$ des mehrstufig copartitionierten Algorithmus gelten:

$$\boldsymbol{\tau}^\kappa \mathbf{d}^\kappa(e^\kappa) + t_{0,\delta(e^\kappa)} - t_{v,\sigma(e^\kappa)} \geq 0, \tag{5.36}$$

$$\boldsymbol{\tau}^\kappa \mathbf{d}^\kappa(e^\kappa) + t_{0,\delta(e^\kappa)} - \max_{1 \leq t \leq T_{\sigma(e^\kappa)}} \left(t_{0,\sigma(e^\kappa)_t} + l_{m(\sigma(e^\kappa)_t)} \right) \geq 0, \tag{5.37}$$

$$\boldsymbol{\tau}^\kappa \left(\mathbf{i}^\kappa_\delta - \mathbf{i}^\kappa_\sigma \right) + t_{0,\delta(e^\kappa)} - \max_{1 \leq t \leq T_{\sigma(e^\kappa)}} \left(t_{0,\sigma(e^\kappa)_t} + l_{m(\sigma(e^\kappa)_t)} \right) \geq 0, \tag{5.38}$$

$$\boldsymbol{\tau}^\kappa \mathbf{i}^\kappa_\delta + t_{0,\delta(e^\kappa)} - \max_{1 \leq t \leq T_{\sigma(e^\kappa)}} \left(\boldsymbol{\tau}^\kappa \mathbf{i}^\kappa_\sigma + t_{0,\sigma(e^\kappa)_t} + l_{m(\sigma(e^\kappa)_t)} \right) \geq 0, \tag{5.39}$$

$$t(\mathbf{i}^\kappa_\delta, \delta(e^\kappa)) - \max_{1 \leq t \leq T_{\sigma(e^\kappa)}} \left(t(\mathbf{i}^\kappa_\sigma, \sigma(e^\kappa)_t) + l_{m(\sigma(e^\kappa)_t)} \right) \geq 0, \tag{5.40}$$

wobei $l_{m(\sigma(e)_t)}$ die Latenz für die Ausführung der Funktion $F_{\sigma(e)_t}$ in Funktionseinheit m ist.

Damit kann der minimale zeitliche Abstand zwischen der Erzeugung einer Variable und der Nutzung dieser Variable bei einem Datenabhängigkeit e^κ und dem daraus resultierenden Abhängigkeitsvektors $\mathbf{d}^\kappa(e^\kappa)$ wie folgt bestimmt werden:

$$t^\triangle \left(\mathbf{d}^\kappa(e^\kappa), e^\kappa \right) = \boldsymbol{\tau}^\kappa \mathbf{d}^\kappa(e^\kappa) + t_{0,\delta(e^\kappa)} - t_{v,\sigma(e^\kappa)}, \tag{5.41}$$

wobei $t_{v,\sigma(e^\kappa)} = \max_{1 \leq t \leq T_{\sigma(e^\kappa)}} \left(t_{0,\sigma(e^\kappa)_t} + l_{m(\sigma(e^\kappa)_t)} \right)$ der früheste Zeitpunkt der Verfügbarkeit der Variable $y_{\sigma(e^\kappa)}[.]$ bezüglich des Iterationsintervalls λ ist.

Kapitel 6

Teilwortparallelisierung

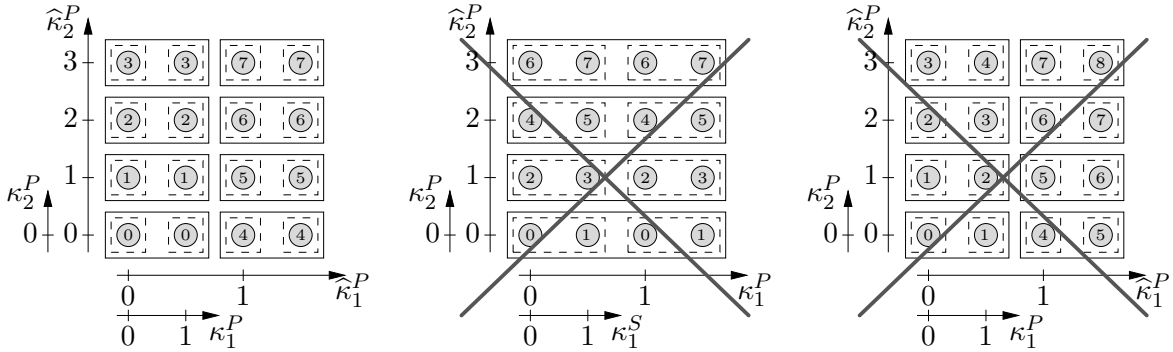
In diesem Kapitel wird die Transformation eines Algorithmus mit Einzeldatenoperationen zu einem ein-/ausgabeäquivalenten Algorithmus mit teilwortparallelen Operationen (kurz: TWP-Operationen) beschrieben. Dabei leiten wir die TWP-Operationen für die Berechnung der Daten und für das notwendige Umsortieren der Daten her. Diese Transformation bezeichnen wir als *Teilwortparallelisierung*. Abbildung 6.1 illustriert diesen Prozess.

Abbildung 6.1: Teilwortparallelisierung

Ausgangsalgorithmus der Teilwortparallelisierung ist ein UItA, wie er im Kapitel 3 beschrieben wurde. Hierbei sind die Variablen $y_s[.]$ Einzeldaten und die Funktionen F_s sind Einzeldatenoperationen. Für eine Abbildung des Algorithmus auf einem Rechenfeld mit Teilwortparallelität in den Funktionseinheiten, siehe Kapitel 2, muss der Ausgangsalgorithmus in einen UItA umgewandelt werden, dessen Variablen $\mathbf{y}_s[.]$ Datenwörter voller Breite (kurz: DvB) sind und dessen Funktionen dieses DvBs als Ein- und Ausgabevariablen haben. Die Teilwörter der DvBs sind hierbei die Einzeldaten $y_s[.]$ des Ausgangsalgorithmus. Die Funktionen auf Basis von DvBs bezeichnen wir als teilwortparallele Operationen (kurz: TWP-Operationen).

Die Teilwörter der DvBs müssen teilweise vor TWP-Operationen umsortiert werden, um die gewünschte TWP-Operation ausführen zu können. Die für das Umsortieren notwendigen Operationen werden als Packoperationen bezeichnet. Die Entstehung der Packoperationen bei der Teilwortparallelisierung und die Funktionen dieser Operationen werden in diesem Kapitel ebenfalls hergeleitet.

Für unsere Transformation betrachten wir die Berechnungen der Teilwörter in den Funktionseinheiten, siehe Abbildung 2.3 auf Seite 10, als Berechnungen von Einzeldaten in Prozessorelementen eines Rechenfeldes. Diese Herangehensweise erlaubt es uns, einen neuen Ansatz für die Ausnutzung von Teilwortparallelität zu verwenden, indem wir die



(a) $\boldsymbol{\Theta}^S = \mathbf{E}$, $\boldsymbol{\tau}^{\text{offs}} = \mathbf{0}$ (b) $\boldsymbol{\Theta}^S \neq \mathbf{E}$, $\boldsymbol{\tau}^{\text{offs}} = \mathbf{0}$ (c) $\boldsymbol{\Theta}^S = \mathbf{E}$, $\boldsymbol{\tau}^{\text{offs}} \neq \mathbf{0}$

Abbildung 6.2: Packen von Teilwörtern in einem Datenwort voller Breite

Methoden des Rechenfeldentwurfs, siehe Kapitel 4 und 5, nutzen. Diese Methoden müssen aber an die Beschränkungen der Teilwortparallelität angepasst werden.

Ausgangspunkt der Teilwortparallelisierung ist eine einstufige modifizierte Copartitionierung, wie in Abschnitt 5.1.2 beschrieben. Mittels der Partitionierungsmatrizen $\boldsymbol{\Theta}^S$ und $\boldsymbol{\Theta}^P$ sollen den B Teilwörtern der Funktionseinheit B Einzeldaten $y_s[\mathbf{i}]$ des Algorithmus zugeordnet werden, welche parallel verarbeitet werden können. Dazu interpretieren wir die parallele Berechnung der B Teilwörter in der Funktionseinheit als parallele Berechnung von Einzeldaten in einem **linearen** Rechenfeld mit B Prozessorelementen. Somit muss für die LP-Partition $\boldsymbol{\Theta}^P = \text{diag}(\boldsymbol{\vartheta}^P)$ gelten:

$$\exists! \, \vartheta_j^P > 1 \qquad \text{und} \qquad \det(\boldsymbol{\Theta}^P) = B. \tag{6.1}$$

Folglich ist die Allokationsmatrix $\boldsymbol{\Gamma}$ entsprechend der Definition 5.1 eine einzeilige Matrix mit $\gamma_{1,j} = 1$.

Damit die Einzeldaten $y_s[\mathbf{i}]$ des Ausgangsalgorithmus, welche in der Raumrichtung j benachbart sind, auch direkt nebeneinander in der Funktionseinheit verarbeitet werden, hat die LS-Partition der modifizierten Copartition immer die Größe $\det(\boldsymbol{\Theta}^S) = 1 \Rightarrow \boldsymbol{\Theta}^S = \mathbf{E}$. Diese Vorgabe gewährleistet auch ein effizientes Zusammenfassen der Teilwörter in DvBs, wie sich später zeigen wird.

Weiterhin muss bei der Ablaufplanung für den Verzögerungsvektor $\boldsymbol{\tau}^{\text{offs}}$, vorgestellt in Abschnitt 5.2.2, gelten:

$$\boldsymbol{\tau}^{\text{offs}} = \mathbf{0}. \tag{6.2}$$

Diese Bedingung stellt sicher, dass die in einer LP-Partition $\widehat{\boldsymbol{\kappa}}^P$ enthaltenen Operationen gleichzeitig ausgeführt werden und somit durch TWP-Operationen ersetzt werden können.

Abbildung 6.2 illustriert die Notwendigkeit der beiden Bedingungen $\boldsymbol{\Theta}^S = \mathbf{E}$ und $\boldsymbol{\tau}^{\text{offs}} = \mathbf{0}$. Der Algorithmus wurde jeweils mit $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ partitioniert, was zu einer Parallelverarbeitung mit zwei Teilwörtern ($B = 2$) führt. In Bild 6.2(a) sind die Beschränkungen für

die LS-Partition und den Offsetvektor eingehalten, während in 6.2(b) die LS-Partition die Größe $\boldsymbol{\Theta}^S = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ hat und in 6.2(c) der Offsetvektor $\boldsymbol{\tau}^{\text{offs}} = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ gesetzt wurde. Die Abarbeitungszeitpunkte der Iterationen, welche in den jeweiligen Iterationspunkten angegeben sind, zeigen nun, dass nur bei Abbildung 6.2(a) die horizontal paarweise benachbarten Iterationen gleichzeitig ausgeführt werden.

Wenn die Zuweisungen der Iterationen einer Partition gleichzeitig abgearbeitet werden, so gilt $\boldsymbol{\tau}^{\kappa} \cdot \mathbf{i}^{\kappa} = 0$. Existiert zwischen zwei Iterationen einer Partition eine Datenabhängigkeit e, so kann nur mittels der teilzuweisungsabhängigen Verzögerungen t_{0,s_t} die Kausalitätsbedingung erfüllt werden. Ist nun aber die Quelle $\sigma(e) = s$ und die Senke $\delta(e) = s_t$ der Datenabhängigkeit die gleiche Zuweisung s, so kann die Kausalitätsbedingung im Allgemeinen nicht mehr erfüllt werden. Eine Ausnahme bilden die Propagierungszuweisungen, deren Transformation in den Abschnitten 6.4.2 (interne Propagierungszuweisungen) und 6.5.4 (Eingabe- und Propagierungszuweisungen) vorgestellt wird.

Die Partitionsmatrix $\boldsymbol{\Theta}^P$ ist somit so zu wählen, dass für alle Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$, mit $\sigma(e) = s$ und $\delta(e) = s_t$, wobei die Zuweisung s eine Berechnungszuweisung ist, gilt: $|\mathbf{d}(e)| \neq k \cdot \boldsymbol{\Gamma}^t$, $1 \leq k < B$.

Basierend auf den vorgestellten Beschränkungen für die Copartitionierung wird im nachfolgenden Abschnitt ein Datenwort voller Breite definiert, welches die Daten $y_s[\mathbf{i}]$ einer Copartition enthält. In den Abschnitten 6.2 und 6.3 wird die Entstehung von Packoperationen durch den Zerfall der Abhängigkeitsvektoren des Algorithmus und aufgrund der Partitionierung von Zuweisungen mit mehreren Teilzuweisungen vorgestellt. Anschließend erfolgt in den Abschnitten 6.4 und 6.5 die Transformation der internen Zuweisungen und der Ein- und Ausgabezuweisungen des Algorithmus. Bei der Transformation der Ein- und Ausgabezuweisungen können durch das Lesen der Daten aus dem Speicher oder durch das Schreiben der Daten in den Speicher wiederum zusätzliche Packoperationen entstehen.

Ausgehend von den Packoperationen, welche bei die Teilwortparallelisierung eines UItA entstehen, werden in Abschnitt 6.6 Packbefehle hergeleitet, welche die benötigten Packoperationen realisieren können.

Abschließend findet eine Einordnung der vorgestellten Algorithmentransformation bezüglich bekannter Methoden der Teilwortparallelisierung statt.

6.1 Datenwort voller Breite (DvB)

Ein Datenwort voller Breite soll als Vektor der Länge B definiert werden. Die Beschreibung eines DvBs erfolgt unter Verwendung einer einstufigen Copartitionierung mit Verschiebungsvektor $\boldsymbol{\vartheta}_0$, siehe Abschnitt 5.1.2.

Definition 6.1 (Datenwort voller Breite (DvB))
Ein Datenwort voller Breite (DvB) $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ *ist ein Vektor der Länge* B*. Die Elemente (Teilwörter)* $y[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ *des DvBs* $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ *sind Instanzen der Variablen* $y[\mathbf{i}]$*. Die*

Zuordnung der Teilwörter $y[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0] = y[\mathbf{i}]$ *zu einem DvB erfolgt über eine Copartitionierung mit den Bedingungen:* $\det(\boldsymbol{\Theta}^P) = B$, $\exists!\, \vartheta^P_{j,j} > 1$ *und* $0 \leq \vartheta_0 < B$. *Die Teilwörter* $y[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ *sind wie folgt im DvB angeordnet:*

$$\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0] = \left(y[0, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0],\, y[1, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0],\, \ldots,\, y[B-1, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]\right)^t, \tag{6.3}$$

wobei gilt: $y[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0] = y[b \cdot \boldsymbol{\Gamma}^t + \boldsymbol{\Theta}^P \widehat{\boldsymbol{\kappa}} + \vartheta_0 \cdot \boldsymbol{\Gamma}^t] = y[\mathbf{i}]$.

Die Allokationsmatrix $\boldsymbol{\Gamma}$ *hat entsprechend der Vorgaben für die Partitionsmatrix* $\boldsymbol{\Theta}^P$ *die Größe* $1 \times n$. *Sie besitzt* ***ein*** *Nicht-Null-Element* $\gamma_{1,j} = 1$, *welches sich an der Position* j *befindet, an der* $\vartheta^P_{j,j} = B > 1$ *ist.*

Beispiel 6.1 (Datenwort voller Breite)
Die Variablen $y[\mathbf{i}]$ *mit* $\mathbf{i} = \left(\begin{smallmatrix} i_1 \\ i_2 \end{smallmatrix}\right) \in \mathcal{I}$ *sollen zu DvBs* $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ *mit* $B = 4$ *Teilwörtern gepackt werden. Für die Partitionsmatrix* $\boldsymbol{\Theta}^P$ *kann* $\boldsymbol{\Theta}^P_1 = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ *oder* $\boldsymbol{\Theta}^P_2 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ *gewählt werden. Die zugehörigen Allokationsmatrizen* $\boldsymbol{\Gamma}$ *lauten* $\boldsymbol{\Gamma}_1 = \begin{pmatrix} 1 & 0 \end{pmatrix}$ *und* $\boldsymbol{\Gamma}_2 = \begin{pmatrix} 0 & 1 \end{pmatrix}$. *Die Verschiebung* ϑ_0 *kann die Werte 0, 1, 2 oder 3 annehmen.*

Wird die Partitionsmatrix $\boldsymbol{\Theta}^P_1 = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ *und die Verschiebung* $\vartheta_0 = 1$ *gewählt, so erhalten wir die DvBs* $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, 1] = \left(y[0, \widehat{\boldsymbol{\kappa}}^P, 1], y[1, \widehat{\boldsymbol{\kappa}}^P, 1], y[2, \widehat{\boldsymbol{\kappa}}^P, 1], y[3, \widehat{\boldsymbol{\kappa}}^P, 1]\right)^t$. *Die Teilwörter* $y[b, \widehat{\boldsymbol{\kappa}}^P, 1]$ *können mittels* $\mathbf{i} = \left(\begin{smallmatrix} b \\ 0 \end{smallmatrix}\right) + \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right) \widehat{\boldsymbol{\kappa}} + \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ *eineindeutig den Variablen* $y[\mathbf{i}]$ *zugeordnet werden. So erhalten wir für die Iteration* $\widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right)$ *das DvB* $\mathbf{y}[\left(\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right), 1] = \left(y\left[\begin{smallmatrix} 9 \\ 1 \end{smallmatrix}\right], y\left[\begin{smallmatrix} 10 \\ 1 \end{smallmatrix}\right], y\left[\begin{smallmatrix} 11 \\ 1 \end{smallmatrix}\right], y\left[\begin{smallmatrix} 12 \\ 1 \end{smallmatrix}\right]\right)^t$.

Bei Wahl der Partitionsmatrix $\boldsymbol{\Theta}^P_2 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ *und der Verschiebung* $\vartheta_0 = 1$ *ergibt sich wieder das DvBs* $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, 1] = \left(y[0, \widehat{\boldsymbol{\kappa}}^P, 1], y[1, \widehat{\boldsymbol{\kappa}}^P, 1], y[2, \widehat{\boldsymbol{\kappa}}^P, 1], y[3, \widehat{\boldsymbol{\kappa}}^P, 1]\right)^t$. *Die Zuordnung der Teilwörter* $y[b, \widehat{\boldsymbol{\kappa}}^P, 1]$ *zu den Variablen* $y[\mathbf{i}]$ *lautet nun aber* $\mathbf{i} = \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right) + \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right) \widehat{\boldsymbol{\kappa}} + \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$. *Für die Iteration* $\widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right)$ *erhalten wir nun das DvB* $\mathbf{y}[\left(\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right), 1] = \left(y\left[\begin{smallmatrix} 2 \\ 5 \end{smallmatrix}\right], y\left[\begin{smallmatrix} 2 \\ 6 \end{smallmatrix}\right], y\left[\begin{smallmatrix} 2 \\ 7 \end{smallmatrix}\right], y\left[\begin{smallmatrix} 2 \\ 8 \end{smallmatrix}\right]\right)^t$.

Durch die beschriebene Copartitionierung entsteht aus dem Iterationsraum $\mathcal{I} = \{\mathbf{i} \mid \mathbf{A} \cdot \mathbf{i} \geq \mathbf{a}_0\}$ der Variablen $y[\mathbf{i}]$ der partitionierte Iterationsraum

$$\mathcal{I}^\kappa = \left(\mathbf{i}^\kappa = \left(\begin{smallmatrix} b \\ \widehat{\boldsymbol{\kappa}}^P \end{smallmatrix}\right) \,\middle|\, \left(\begin{smallmatrix} 1 & \mathbf{0} \\ -1 & \mathbf{0} \\ \mathbf{A}\boldsymbol{\Gamma}^t & \mathbf{A}\boldsymbol{\Theta}^P \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} b \\ \widehat{\boldsymbol{\kappa}}^P \end{smallmatrix}\right) \geq \left(\begin{smallmatrix} 0 \\ -B+1 \\ \mathbf{a}_0 - \mathbf{A}\boldsymbol{\vartheta}_0 \end{smallmatrix}\right) \right) = \mathcal{K}^b \oplus \widehat{\mathcal{K}} \tag{6.4}$$

der Teilwörter $y[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$. Für die Partitionsteilräume $\mathcal{K}^b$ und $\widehat{\mathcal{K}}$ gilt:

$$\mathcal{K}^b = \{b \mid 0 \leq b < B\} \qquad \text{und} \qquad \widehat{\mathcal{K}} = \underset{b}{\mathrm{FME}}\left(\mathcal{I}^\kappa\right).$$

Durch Fourier-Motzkin-Elimination der Variabe b aus dem Ungleichungssystem 6.4 erhalten wir den Partitionsraum $\widehat{\mathcal{K}} = \mathrm{FME}_b(\mathcal{I}^\kappa)$. Dieser Raum ist der Definitionsbereich bzw. der Iterationsraum für die DvBs $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$.

Beispiel 6.2 (Iterationsraum der DvBs)
Der Iterationsraum $\mathcal{I}$ *der Iterationen* $\mathbf{i} \in \mathcal{I}$ *der Variable* $y[\mathbf{i}]$ *aus Beispiel 6.1 betrage*

$$\mathcal{I} = \left\{ \mathbf{i} = \left(\begin{smallmatrix} i_1 \\ i_2 \end{smallmatrix}\right) \;\middle|\; \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \left(\begin{smallmatrix} i_1 \\ i_2 \end{smallmatrix}\right) \geq \begin{pmatrix} 1 \\ 0 \\ -14 \\ -15 \end{pmatrix} \right\}. \tag{6.5}$$

Bei der Wahl der Partitionsmatrix $\mathbf{\Theta}_1^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ *und der Verschiebung* $\vartheta_0 = 1$ *erhalten wir den folgenden partitionierten Iterationsraum* $\mathcal{I}^\kappa$*:*

$$\mathcal{I}^\kappa = \left\{ \mathbf{i}^\kappa = \begin{pmatrix} b \\ \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 4 & 0 \\ 0 & 0 & 1 \\ -1 & -4 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} b \\ \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \geq \begin{pmatrix} 0 \\ -3 \\ 0 \\ 0 \\ -13 \\ -15 \end{pmatrix} \right\}. \tag{6.6}$$

Somit ergibt sich durch die Fourier-Motzkin-Elimination der Iterationsraum $\widehat{\mathcal{K}}$ *für die DvBs* $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, 1]$*:*

$$\widehat{\mathcal{K}} = \underset{b}{\mathrm{FME}}\left(\mathcal{I}^\kappa\right) = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 4 & 0 \\ 0 & 1 \\ -4 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \geq \begin{pmatrix} -3 \\ 0 \\ -13 \\ -15 \end{pmatrix} \right\}. \tag{6.7}$$

Für die weitere Arbeit ist es notwendig, verschiedene Arten von DvBs zu unterscheiden. Es gibt *Berechnungs-DvBs*, welche für die teilwortparallele Verarbeitung benötigt werden und es gibt *Speicher-DvBs*, welche zur Beschreibung der Ein- und Ausgabe-DvBs im Speicher benötigt werden. Diese beiden Arten mit deren Eigenschaften werden im Abschitt 6.1.1 vorgestellt.

Ob ein Teilwort $y[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ in einem DvB $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ definiert ist, ist im Allgemeinen abhängig vom DvB $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$. Im Abschnitt 6.1.2 werden die Iterationen $\widehat{\boldsymbol{\kappa}}^P$ ermittelt, in welchen die DvBs vollständig sind oder in denen nur die unteren, mittleren oder oberen Teilwörter gültig sind, d. h. in dem die zugehörigen Iterationen $\mathbf{i}$ im Iterationsraum $\mathcal{I}$ definiert sind.

6.1.1 Berechnungs- und Speicher-DvBs

Bei der Teilwortparallelisierung muss zwischen Berechnungs- und Speicher-DvBs unterschieden werden. Berechnungs-DvBs dienen der Beschreibung von internen Variablen, welche vorrangig für die Datenverarbeitung genutzt werden. Die Speicher-DvBs sind Ein- und Ausgabedaten, die vor oder nach der Verarbeitung des Algorithmus im Speicher liegen.

Berechnungs-DvBs

Berechnungs-DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P, 0] = \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ erhalten wir durch die Copartitionierung des Iterationsraums $\mathcal{I}_s$ der internen Variablen mit $\mathbf{\Theta}^S = \mathbf{E}$, $\det(\mathbf{\Theta}^P) = B$ und $\exists!\, \vartheta_{j,j}^P > 1$. Die Verschiebung ϑ_0 beträgt bei Berechnungs-DvBs immer $\vartheta_0 = 0$.

Durch die Partitionierung eines Iterationsraums $\mathcal{I}_s$ einer Zuweisung s oder eines Iterationsraums $\mathcal{I}_{s_t}$ einer Teilzuweisung s_t können DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$, $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s$ oder $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t}$ entstehen, in denen einige Teilwörter $y_s[b, \widehat{\boldsymbol{\kappa}}^P, 0] = y_s[b, \widehat{\boldsymbol{\kappa}}^P]$ undefiniert sind, da die zugehörige Iterationen $\mathbf{i}$ außerhalb des Iterationsraums $\mathcal{I}_s$ bzw. $\mathcal{I}_{s_t}$ liegen. Ein solches DvB bezeichnen wir als unvollständiges DvB.

Besteht das unvollständige DvB aus internen und undefinierten Daten, so kann mit diesem DvB so gerechnet werden, als ob das DvB vollständig wäre. Die undefinierten Teilwörter in den unvollständigen DvBs haben keinen Einfluss auf das Ergebnis des Algorithmus, wenn der Ausgangsalgorithmus exakt formuliert wurde. Es ist aber zu beachten, dass der Speicherbedarf $B \cdot |\widehat{\mathcal{K}}_s|$ für die Daten in Form von DvBs größer sein kann als der Speicherbedarf $|\mathcal{I}_s| \leq B \cdot |\widehat{\mathcal{K}}_s|$ in Form von Einzeldaten.

Speicher-DvBs

Die Speicher-DvBs $\mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ werden durch die Partitionierung des Indexraums $\mathcal{I}^m$ der globalen Variablen $y_m[\mathbf{i}^M]$ bestimmt. Hierzu verwenden wir die Datenausrichtungsmatrix $\mathbf{\Gamma}^M$, siehe Definition 2.1. Mit dieser Matrix bestimmen wir die Partitionsmatrix $\mathbf{\Theta}^P = \mathbf{\Theta}^M$ mit $\det(\mathbf{\Theta}^M) = B$ unter der Annahme, dass die Allokationsmatrix $\mathbf{\Gamma} = \mathbf{\Gamma}^M$ ist. Die LS-Partition hat die Größe $\mathbf{\Theta}^S = \mathbf{E}$.

Mit Hilfe der Verschiebung $\vartheta_0 = \vartheta_0^M$ kann festgelegt werden, ob nur ausgerichtete Speicherzugriffe[1] ($\vartheta_0^M = 0$) oder auch unausgerichtete Speicherzugriffe ($1 \leq \vartheta_0^M < B$) möglich sind. Das heißt, abhängig von der Speicherarchitektur muss der Wertebereich für die Verschiebung ϑ_0^M festgelegt werden, um für einen Algorithmus vorzugeben, ob dessen Daten als DvBs gepackt ausgerichtet oder unausgerichtet gelesen oder geschrieben werden können.

Beispiel 6.3 (Ein- und Ausgabe-DvBs bei der Matrixmultiplikation)
Bei der Matrixmultiplikation $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ *gibt es zwei Eingabevariablen* $a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right]$ *und* $b\left[\begin{smallmatrix} j \\ k \end{smallmatrix}\right]$, *sowie eine Ausgabevariable* $c\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right]$. *Die Indexräume der Variablen sind zweidimensional. Die Variablen sind immer zeilenweise* $\mathbf{\Gamma}_a^M = \mathbf{\Gamma}_b^M = \mathbf{\Gamma}_c^M = \mathbf{\Gamma}^M = \begin{pmatrix} 1 & 0 \end{pmatrix}$ *im Speicher hinterlegt. Somit gilt für alle Variablen* $\mathbf{\Theta}^M = \left(\begin{smallmatrix} B & 0 \\ 0 & 1 \end{smallmatrix}\right)$, *wobei wir für* $B = 4$ *annehmen.*

Erlaubt die Speicherarchitektur nun nur ausgerichtete Speicherzugriffe, so erhalten wir für die Variable $a[.]$ *die folgenden Speicher-DvBs der Form:*

$$\mathbf{a}[\widehat{\boldsymbol{\kappa}}^M, 0] = \left(a\left[\begin{smallmatrix} 0+B\cdot\widehat{\kappa}_1^M \\ \widehat{\kappa}_2^M \end{smallmatrix}\right], a\left[\begin{smallmatrix} 1+B\cdot\widehat{\kappa}_1^M \\ \widehat{\kappa}_2^M \end{smallmatrix}\right], \ldots, a\left[\begin{smallmatrix} B-1+B\cdot\widehat{\kappa}_1^M \\ \widehat{\kappa}_2^M \end{smallmatrix}\right] \right). \tag{6.8}$$

Ist zum Beispiel $\widehat{\boldsymbol{\kappa}}^M = \left(\begin{smallmatrix} 3 \\ 5 \end{smallmatrix}\right)$, *so folgt:* $\mathbf{a}\left[\begin{smallmatrix} 3 \\ 5 \end{smallmatrix}\right] = \left(a\left[\begin{smallmatrix} 12 \\ 5 \end{smallmatrix}\right], a\left[\begin{smallmatrix} 13 \\ 5 \end{smallmatrix}\right], a\left[\begin{smallmatrix} 14 \\ 5 \end{smallmatrix}\right], a\left[\begin{smallmatrix} 15 \\ 5 \end{smallmatrix}\right]\right)$.

Bei unausgerichteten Speicherzugriffen kann zusätzlich noch eine Verschiebung ϑ_0^M *realisiert werden. Somit sind Speicher-DvBs der Form*

$$\mathbf{a}[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M] = \left(a\left[\begin{smallmatrix} 0+B\cdot\widehat{\kappa}_1^M+\vartheta_0^M \\ \widehat{\kappa}_2^M \end{smallmatrix}\right], a\left[\begin{smallmatrix} 1+B\cdot\widehat{\kappa}_1^M+\vartheta_0^M \\ \widehat{\kappa}_2^M \end{smallmatrix}\right], \ldots, a\left[\begin{smallmatrix} B-1+B\cdot\widehat{\kappa}_1^M+\vartheta_0^M \\ \widehat{\kappa}_2^M \end{smallmatrix}\right] \right). \tag{6.9}$$

möglich. Ist zum Beispiel $\vartheta_0^M = 1$ *und* $\widehat{\boldsymbol{\kappa}}^M = \left(\begin{smallmatrix} 3 \\ 5 \end{smallmatrix}\right)$, *so erhalten wir das DvB:* $\mathbf{a}[\left(\begin{smallmatrix} 3 \\ 5 \end{smallmatrix}\right), 1] = \left(a\left[\begin{smallmatrix} 13 \\ 5 \end{smallmatrix}\right], a\left[\begin{smallmatrix} 14 \\ 5 \end{smallmatrix}\right], a\left[\begin{smallmatrix} 15 \\ 5 \end{smallmatrix}\right], a\left[\begin{smallmatrix} 16 \\ 5 \end{smallmatrix}\right]\right)$.

In der gleichen Form wie für die Variablen $a[.]$ *können auch für die Variablen* $b[.]$ *und* $c[.]$ *die Speicher-DvBs definiert werden.*

[1] siehe Abschnitt 2.4.

Durch die Partitionierung des Indexraums der globalen Variablen können unvollständige DvBs entstehen. Hierbei muss zwischen Eingabe- und Ausgabe-DvBs unterschieden werden. Ist das unvollständige DvB ein Eingabe-DvB, so muss sichergestellt sein, dass das unvollständige DvB gelesen werden kann, ohne einen Speicherzugriffsfehler zu verursachen. Das heißt, das DvB muss eine gültige Speicheradresse besitzen. Im Algorithmus kann mit den unvollständigen Eingabe-DvBs genauso umgegangen werden, wie mit den unvollständigen Berechnungs-DvBs.

Wie beim unvollständigen Eingabe-DvB muss auch beim unvollständigen Ausgabe-DvB die Gültigkeit der Speicheradresse gewährleistet sein. Doch weiterhin muss beim Schreiben des Ausgabe-DvBs sichergestellt werden, dass die im Speicher vorhandenen gültigen Daten nicht durch die undefinierten Teilwörter des unvollständigen Ausgabe-DvBs überschrieben werden.

Der Umgang mit Speicher-DvBs, die bei der Transformation der Ein- und Ausgabezuweisungen eines Algorithmus entstehen, wird in Abschnitt 6.5 ausführlich vorgestellt.

6.1.2 Vollständigkeit von Datenwörtern voller Breite

Durch die Partitionierung des Iterationsraums $\mathcal{I}_s$ oder des Indexraums $\mathcal{I}^m$ in DvBs mit B Teilwörtern können DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ entstehen, in denen nicht alle Teilwörter $y_s[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ gültig sind. Anhand des Iterationsraums $\mathcal{I}_s$ soll die Bestimmung der vollständigen und unvollständigen DvBs vorgestellt werden. In der gleichen Weise kann die Bestimmung der vollständigen und unvollständigen DvBs des Indexraums $\mathcal{I}^m$ erfolgen.

Abbildung 6.3 zeigt die Problematik am Beispiel des Iterationsraums

$$\mathcal{I}_s = \left\{ \mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \\ 0 & -1 \end{pmatrix} \mathbf{i} \geq \begin{pmatrix} 0 \\ 0 \\ -7 \\ -5 \end{pmatrix} \right\}, \tag{6.10}$$

welcher mit $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ und $\vartheta_0 = 0$ partitioniert wurde.

Nur das DvB $\mathbf{y}_s[\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right), 0]$ ist ein vollständiges DvB, alle anderen DvBs des Partitionsraums $\widehat{\mathcal{K}}_s$ sind unvollständig.

Nachfolgend soll ein Methode zur Bestimmung der Iterationen $\widehat{\boldsymbol{\kappa}}^P$ in denen die DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ vollständig sind oder in denen mindestens das unterste und/oder das oberste Teilwort **nicht** definiert ist, vorgestellt werden.

Für die unvollständigen DvBs ist es notwendig, das erste gültige Teilwort $y_s[b_{\min}, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ und das letzte gültige Teilwort $y_s[b_{\max}, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ zu bestimmen. Im Allgemeinen sind diese Positionen von der Iteration $\widehat{\boldsymbol{\kappa}}^P$ des DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ abhängig.

Ein Sonderfall der unvollständigen DvBs sind Partitionsräume, bei denen alle DvBs nur **ein** gültiges Teilwort besitzen.

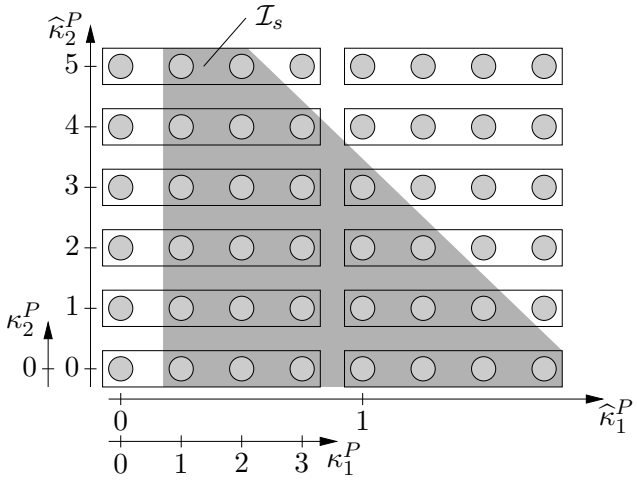


Abbildung 6.3: Vollständige und unvollständige Datenwörter voller Breite

In welchen Iterationen sind die DvBs vollständig und in welchen nicht?

Im Folgenden werden die Bereiche $\widehat{\mathcal{K}}_s^v$, $\widehat{\mathcal{K}}_s^a$, $\widehat{\mathcal{K}}_s^e$, $\widehat{\mathcal{K}}_s^i$ bestimmt, in denen die DvBs vollständig ($\widehat{\mathcal{K}}_s^v$) sind, beziehungsweise nur für die oberen ($\widehat{\mathcal{K}}_s^a$), die unteren ($\widehat{\mathcal{K}}_s^e$) oder inneren Teilwörter ($\widehat{\mathcal{K}}_s^i$), das heißt $b_{\min} > 0$ und $b_{\max} < B-1$, definiert sind. Hierzu wenden wir auf alle Elemente des Iterationsraums $\mathcal{I}_s$ der Ausgangszuweisung s die Funktionen $f^l(\mathbf{i}) = \mathbf{E}\cdot\mathbf{i} - (B-1)\cdot\mathbf{\Gamma}^t$ und $f^r(\mathbf{i}) = \mathbf{E}\cdot\mathbf{i} + (B-1)\cdot\mathbf{\Gamma}^t$ an, welche ein Verschieben der Iterationen in der Raumrichtung, in der die Daten zu DvBs gepackt werden, um $(B-1)$ Iterationen nach rechts und links realisieren. Dadurch erhalten wir die verschobenen Iterationsräume $\mathcal{I}_s^l = f^l(\mathcal{I}_s)$ und $\mathcal{I}_s^r = f^r(\mathcal{I}_s)$. Diese Iterationsräume $\mathcal{I}_s$, $\mathcal{I}_s^l$ und $\mathcal{I}_s^r$ partitionieren wir mit $\mathbf{\Theta}^P$ und bestimmen mittels der Fourier-Motzkin-Elimination die Partitionsräume $\widehat{\mathcal{K}}_s$, $\widehat{\mathcal{K}}_s^l$ und $\widehat{\mathcal{K}}_s^r$.

Das Verschieben der Iterationen $\mathbf{i}$ im Iterationsraum $\mathcal{I}_s$ um $(B-1)\cdot\mathbf{\Gamma}^t$ Iterationen nach rechts oder links soll anhand von B Teilwörtern $y_s[\mathbf{i}], y_s[\mathbf{i}+\mathbf{\Gamma}], \ldots y_s[\mathbf{i}+(B-1)\cdot\mathbf{\Gamma}]$, welche durch die Teilwortparallelisierung des Iterationsraums $\mathcal{I}_s$ in dem DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \big(y_s[\mathbf{i}], y_s[\mathbf{i}+\mathbf{\Gamma}], \ldots, y_s[\mathbf{i}+(B-1)\cdot\mathbf{\Gamma}]\big)$ zusammengefasst sind, diskutiert werden. Ohne Beschränkung der Allgemeinheit wurde $\vartheta_o = 0$ gesetzt.

Bei einer Linksverschiebung befinden sich nach der Teilwortparallelisierung $B-1$ Teilwörter in dem DvB $\mathbf{y}_s^l[\widehat{\boldsymbol{\kappa}}^P - \mathbf{\Gamma}^t]$ und ein Teilwort $(y_s[\mathbf{i}+(B-1)\cdot\mathbf{\Gamma}])$ in dem DvB $\mathbf{y}_s^l[\widehat{\boldsymbol{\kappa}}^P]$. Nach der Rechtsverschiebung und der Teilwortparallelisierung erhalten wir $B-1$ Teilwörter in dem DvB $\mathbf{y}_s^r[\widehat{\boldsymbol{\kappa}}^P + \mathbf{\Gamma}^t]$ und ein Teilwort $(y_s[\mathbf{i}])$ in dem DvB $\mathbf{y}_s^r[\widehat{\boldsymbol{\kappa}}^P]$.

Fehlt dagegen das Teilwort $y_s[\mathbf{i}]$ in dem DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ und existiert damit diese Iteration nicht, so erhalten wir durch die Rechtsverschiebung der Iterationen nur ein DvB $\mathbf{y}_s^r[\widehat{\boldsymbol{\kappa}}^P - \mathbf{\Gamma}^t]$. Entsprechendes gilt bei fehlenden Teilwort $y_s[\mathbf{i}+(B-1)\cdot\mathbf{\Gamma}]$ für Linksverschiebung.

Die Iterationen des Orginal-DvBs $\mathbf{y}_s[.]$ und der verschobenen DvBs $\mathbf{y}_s^l[.]$ und $\mathbf{y}_s^r[.]$ sind in den Mengen $\widehat{\mathcal{K}}_s$, $\widehat{\mathcal{K}}_s^l$ und $\widehat{\mathcal{K}}_s^r$ zusammengefasst. Durch Mengenoperationen lassen sich nun die Iterationen $\widehat{\boldsymbol{\kappa}}^P$ bestimmen,

- bei denen alle Teilwörter definiert sind:

$$\widehat{\mathcal{K}}_s^v = \widehat{\mathcal{K}}_s^l \cap \widehat{\mathcal{K}}_s^r, \tag{6.11}$$

- bei denen gilt, $b_{\min} > 0$ und $b_{\max} < B - 1$:

$$\widehat{\mathcal{K}}_s^i = \widehat{\mathcal{K}}_s \backslash (\widehat{\mathcal{K}}_s^r \cup \widehat{\mathcal{K}}_s^l), \tag{6.12}$$

- bei denen gilt, $b_{\min} > 0$ und $b_{\max} = B - 1$:

$$\widehat{\mathcal{K}}_s^a = (\widehat{\mathcal{K}}_s \backslash \widehat{\mathcal{K}}_s^r) \backslash \widehat{\mathcal{K}}_s^i \text{ und} \tag{6.13}$$

- bei denen gilt, $b_{\min} = 0$ und $b_{\max} < B - 1$:

$$\widehat{\mathcal{K}}_s^e = (\widehat{\mathcal{K}}_s \backslash \widehat{\mathcal{K}}_s^l) \backslash \widehat{\mathcal{K}}_s^i. \tag{6.14}$$

Beispiel 6.4 (Vollständigkeit von DvBs)
Der Iterationsraum $\mathcal{I}_s$, siehe (6.10) auf Seite 85, wurde mittels Partitionsmatrix $\mathbf{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ copartitioniert. Abbildung 6.3 auf Seite 86 illustriert diese Partitionierung.

Es sollen nun die Iterationen $\widehat{\boldsymbol{\kappa}}^P$ ermittelt werden, bei denen die DvBs vollständig sind oder bei denen mindestens das niederwertigste und/oder das höchstwertigste Teilwort fehlt.

Durch die Copartitionierung des Iterationsraums $\mathcal{I}_s$ erhalten wir den Partitionsraum

$$\widehat{\mathcal{K}}_s = \{\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} 0 \\ 5 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} 1 \\ 3 \end{smallmatrix}\right)\}. \tag{6.15}$$

Die Verschiebung des Iterationsraums $\mathcal{I}_s$ um $B - 1 = 3$ nach rechts und links und die Copartitionierung dieser Räume ergibt:

$$\widehat{\mathcal{K}}_s^l = \{\left(\begin{smallmatrix} -1 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} -1 \\ 1 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} -1 \\ 5 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} 0 \\ 4 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2 \\ 0 \end{smallmatrix}\right)\} \quad \text{und} \tag{6.16}$$

$$\widehat{\mathcal{K}}_s^r = \{\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} 1 \\ 5 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 2 \\ 2 \end{smallmatrix}\right), \ldots, \left(\begin{smallmatrix} 2 \\ 3 \end{smallmatrix}\right), \left(\begin{smallmatrix} 3 \\ 0 \end{smallmatrix}\right)\}. \tag{6.17}$$

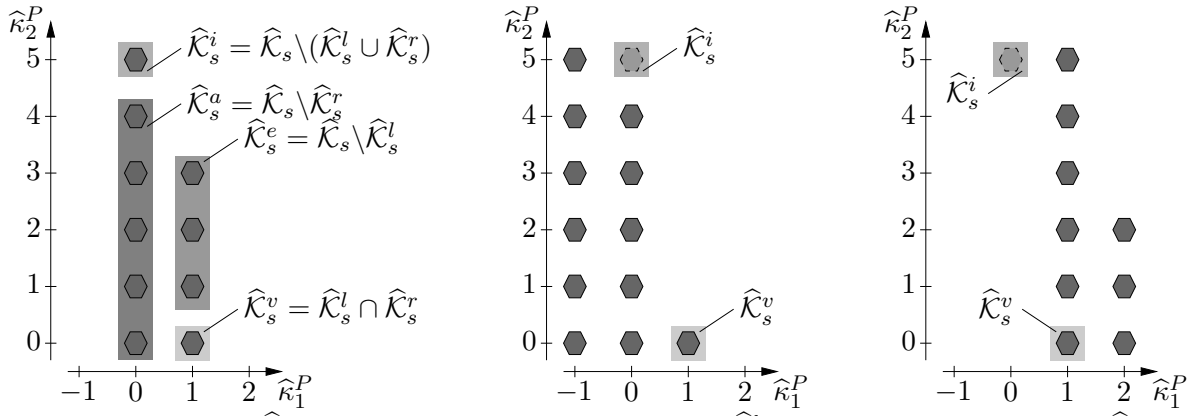
In Abbildung 6.4 sind die Iterationen $\widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{smallmatrix}\right)$ der drei Partitionsräume $\widehat{\mathcal{K}}_s$, $\widehat{\mathcal{K}}_s^l$ und $\widehat{\mathcal{K}}_s^r$ durch dunkelgraue Sechsecke dargestellt.

Mittels der Mengenoperationen aus (6.11) bis (6.14) von Seite 87 erhalten wir nun für unser Beispiel die Teilräume:

$$\widehat{\mathcal{K}}_s^v = \{\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)\}, \qquad \widehat{\mathcal{K}}_s^a = \{\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 2 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 3 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 4 \end{smallmatrix}\right)\}, \tag{6.18}$$

$$\widehat{\mathcal{K}}_s^i = \{\left(\begin{smallmatrix} 0 \\ 5 \end{smallmatrix}\right)\}, \qquad \widehat{\mathcal{K}}_s^e = \{\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ 3 \end{smallmatrix}\right)\}. \tag{6.19}$$

In Abbildung 6.4(a) sind diese Räume dargestellt. Die Partitionsräume $\widehat{\mathcal{K}}_s^v$ und $\widehat{\mathcal{K}}_s^i$ wurden zusätzlich noch in Abbildung 6.4(b) und 6.4(c) eingezeichnet, wobei $\widehat{\mathcal{K}}_s^i$ nicht Teilmenge von $\widehat{\mathcal{K}}_s^u$ oder $\widehat{\mathcal{K}}_s^o$ ist.



(a) Partitionsraum $\widehat{\mathcal{K}}_s$ (b) Partitionsraum $\widehat{\mathcal{K}}_s^l$ (links verschoben) (c) Partitionsraum $\widehat{\mathcal{K}}_s^r$ (rechts verschoben)

Abbildung 6.4: Bestimmung der Vollständigkeit von DvBs mittels verschobener Iterationsräume

Welches ist das unterste und oberste gültige Teilwort eines unvollständigen DvBs?

Für die unvollständigen DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ der Teilräume $\widehat{\mathcal{K}}_s^i$, $\widehat{\mathcal{K}}_s^a$ und $\widehat{\mathcal{K}}_s^e$ muss die Position $b_{\min}$ des untersten und/oder die Position $b_{\max}$ des obersten gültigen Teilworts ermittelt werden.

Aufgrund der Berechnungsvorschrift für die Teilräume $\widehat{\mathcal{K}}_s^a$ und $\widehat{\mathcal{K}}_s^e$ gilt: $b_{\max} = B - 1$ für DvBs des Teilraums $\widehat{\mathcal{K}}_s^a$ und $b_{\min} = 0$ für DvBs des Teilraums $\widehat{\mathcal{K}}_s^e$.

Im Allgemeinen sind die Positionen $b_{\min}$ und $b_{\max}$ von der Iteration $\widehat{\boldsymbol{\kappa}}^P$ des Datenworts voller Breite $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ abhängig. Sie lassen sich aus dem Ungleichungssystem

$$\begin{pmatrix} 1 \\ -1 \\ \mathbf{A}_s \cdot \boldsymbol{\Gamma}^t \end{pmatrix} b \geq \begin{pmatrix} 0 \\ -B+1 \\ \mathbf{a}_{0,s} - \mathbf{A}_s \cdot \vartheta_0 \cdot \boldsymbol{\Gamma}^t \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{A}_s \cdot \boldsymbol{\Theta}^P \end{pmatrix} \widehat{\boldsymbol{\kappa}}^P, \tag{6.20}$$

$$\mathbf{B} \cdot b \geq \mathbf{b}_0 \tag{6.21}$$

bestimmen. Programm 6.1 zeigt einen Algorithmus zur Ermittlung der Positionen mit Hilfe der Matrix $\mathbf{B}$ und des Vektors $\mathbf{b}_0$ aus (6.21). Die Variablen „b_min“ und „b_max“ sind Rückgabewerte der Funktion „begrenzung_DvB()“.

Haben die DvBs jeweils nur ein gültiges Teilwort?

Wird der Iterationsraum $\mathcal{I}_s = \{\mathbf{i} \mid \mathbf{A}_s \cdot \mathbf{i} \geq \mathbf{a}_{0,s}\}$ mit $\boldsymbol{\Theta}^P$ und ϑ_0 partitioniert, so erhalten wir den partitionierten Iterationsraum

$$\mathcal{I}_s^\kappa = \left\{ \mathbf{i}^\kappa = \begin{pmatrix} b \\ \widehat{\boldsymbol{\kappa}}^P \end{pmatrix} \;\middle|\; \begin{pmatrix} 1 & \mathbf{0} \\ -1 & \mathbf{0} \\ \mathbf{A}_s \cdot \boldsymbol{\Gamma}^t & \mathbf{A}_s \cdot \boldsymbol{\Theta}^P \end{pmatrix} \cdot \begin{pmatrix} b \\ \widehat{\boldsymbol{\kappa}}^P \end{pmatrix} \geq \begin{pmatrix} 0 \\ -B+1 \\ \mathbf{a}_{0,s} - \mathbf{A}_s \cdot \vartheta_0 \cdot \boldsymbol{\Gamma}^t \end{pmatrix} \right\}. \tag{6.22}$$

Programm 6.1 Bestimme obere und untere Grenze eines DvBs

```
void begrenzung_DvB( Matrix B, Vektor b0, int &b_min, int &b_max ){
  b_min = b0[0];
  b_max = -b1[1];
  for( unsigned i = 2; i < B.rows(); i++ ){
    if( B[i][0] < 0 ){
      if( b_max > b0[i]/B[i][0] )
        b_max = b0[i]/B[i][0];
    }
    if( B[i][0] > 0 ){
      if( b_min < b0[i]/B[i][0] )
        b_min = b0[i]/B[i][0];
    }
  }
  return;
}
```

Gilt für das Ungleichungssystem $\mathbf{A}^{\kappa} \cdot \mathbf{i}^{\kappa} \geq \mathbf{a}_0^{\kappa}$ allgemein die Bedingung $b_{\min} = b_{\max}$ für $b_{\min} \leq b \leq b_{\max}$, so besitzt jedes DvB immer genau ein Teilwort.

Programm 6.2 zeigt eine Methode zur Überprüfung dieser Bedingung. Die Eingabematrix „A“ und der Eingabevektor „a0“ der Methode sind die Polyedermatrix $\mathbf{A}^{\kappa}$ und der Polyedervektor $\mathbf{a}_0^{\kappa}$ des partitionierten Iterationsraums $\mathcal{I}_s^{\kappa}$. Mit dem Programm wird untersucht, ob es zwei Zeilenvektoren $\mathbf{a}_{i_1}^{\kappa}$ und $\mathbf{a}_{i_2}^{\kappa}$ $(i_1 \neq i_2)$ der Matrix $\mathbf{A}^{\kappa}$ gibt, für die gilt: $\mathbf{a}_{i_1}^{\kappa} = -\mathbf{a}_{i_2}^{\kappa}$, wobei das erste Element der Vektoren ungleich Null sein muss. Gilt nun für die zugehörigen Elemente $a_{0,i_1}^{\kappa} = -a_{0,i_2}^{\kappa}$ des Polyedervektors $\mathbf{a}_0^{\kappa}$, so ist $b = b_{\min} = b_{\max}$. Die Variable b, die Position des Teilworts im DvB, kann hierbei von der Iteration $\widehat{\boldsymbol{\kappa}}^P$ abhängig sein.

Besitzt jedes DvBs nur ein gültiges Teilwort, so ist der Rückgabewert der Funktion „besitzt_ein_Teilwort_je_DvB()“ gleich „true“. In diesem Fall kann mit den Rückgabewerten $\mathbf{a}^{\kappa}$ („con_A“) und a_0 („con_a0“) die Position des gültigen Teilworts b abhängig von der Iteration $\widehat{\boldsymbol{\kappa}}^P$ bestimmt werden:

$$b = \frac{a_0}{a_1^{\kappa}} - \frac{1}{a_1^{\kappa}} \cdot \mathbf{a}_{2\ldots n+1}^{\kappa} \cdot \widehat{\boldsymbol{\kappa}}^P \tag{6.23}$$

Der Vektor $\mathbf{a}_{2\ldots n+1}^{\kappa} = \left(a_2^{\kappa}, a_3^{\kappa}, \ldots, a_{n+1}^{\kappa}\right)$ ist der Vektor $\mathbf{a}^{\kappa}$ ohne dessen erstes Element a_1^{κ}.

Beispiel 6.5 (Ein Teilwort je DvB)
Der Iterationsraum $\mathcal{I} = \left\{\mathbf{i} = \binom{i_1}{i_2} \mid i_1 = i_2 \wedge 0 \leq i_1 \leq 7\right\} = \left\{\binom{0}{0}, \binom{1}{1}, \ldots, \binom{7}{7}\right\}$ *soll für die Teilwortparallelisierung partitioniert werden mit* $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ *und* $\vartheta_0 = 0$. *Die Polyedermatrix* $\mathbf{A}$ *und der Polyedervektor* $\mathbf{a}_0$ *des Iterationsraums* $\mathcal{I}$ *lauten*

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ -1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \textit{und} \qquad \mathbf{a}_0 = \begin{pmatrix} 0 \\ -7 \\ 0 \\ 0 \end{pmatrix}. \tag{6.24}$$

Programm 6.2 Test auf ein Teilworte je DvB

```
bool besitzt_ein_Teilwort_je_DvB( Matrix A, Vektor a0,
                                  Vektor& con_A, int& con_a0 ){
  for( unsigned i = 0; i < A.rows(); i++ ){
    for( unsigned j = i+1; j < A.rows(); j++ ){
      if( A[i][0] != 0 && A[i] == -A[j] && a0[i] == -a0[j] ){
        con_A = A[i];
        con_a0 = a0[i];
        return true;
      }
    }
  }
  return false;
}
```

Für den partitionierten Iterationsraum $\mathcal{I}^\kappa$ gilt nun:

$$\mathcal{I}^\kappa = \left\{ \mathbf{i}^\kappa = \begin{pmatrix} b \\ \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \;\middle|\; \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 4 & 0 \\ -1 & -4 & 0 \\ -1 & -4 & 1 \\ 1 & 4 & -1 \end{pmatrix} \begin{pmatrix} b \\ \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \geq \begin{pmatrix} 0 \\ -3 \\ 0 \\ -7 \\ 0 \\ 0 \end{pmatrix} \right\}. \tag{6.25}$$

Für die letzten beiden Zeilen der Polyedermatrix $\mathbf{A}^\kappa$ gilt: $\mathbf{a}_5^\kappa = -\mathbf{a}_6^\kappa$ mit $a_{5,1}^\kappa \neq 0$. Für die zugehörigen Elemente $a_{0,5}^\kappa$ und $a_{0,6}^\kappa$ des Polyedervektors $\mathbf{a}_0^\kappa$ erhalten wir: $a_{0,5}^\kappa = -a_{0,6}^\kappa$.

Die Position b des gültigen Teilworts kann nun mittels der Gleichung 6.24 ermittelt werden:

$$b = -1 \cdot 0 + \begin{pmatrix} -4 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} -4 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P. \tag{6.26}$$

6.1.3 Zusammenfassung

In diesem Abschnitt wurde das Datenwort voller Breite $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$, welches durch die Coartitionierung des Iterationsraums $\mathcal{I}_s$ oder des Indexraums $\mathcal{I}^m$ entsteht vorgestellt. Ein DvB besteht aus B Teilwörtern $y[b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$.

Es wird zwischen zwei Arten von DvBs unterschieden: Berechnungs-DvB, welche durch die Copartitionierung des Iterationsraums $\mathcal{I}_s$ entstehen und Speicher-DvB, welche wir durch die Copartitionierung des Indexraums $\mathcal{I}^m$ erhalten. Die Verschiebung ϑ_0 beträgt für Berechnungs-DvBs immer Null. Bei Speicher-DvBs kann die Verschiebung ϑ_0, abhängig von der Art des Speicherzugriffs, Null bis $B-1$ betragen. Erlaubt die Architektur nur ausgerichtete Speicherzugriffe so gilt: $\vartheta_0 = 0$. Bei unausgerichteten Speicherzugriffen kann $0 \leq \vartheta_0 < B$ sein.

Abhängig von der Größe und der Form des Iterationsraums $\mathcal{I}_s$ oder des Indexraums $\mathcal{I}^m$ können bei der Copartitionierung der Räume vollständige und unvollständige DvBs entstehen. Für die Bestimmung der Iterationen $\widehat{\boldsymbol{\kappa}}^P$, in der die DvBs $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P, \vartheta_0]$ vollständig

oder unvollständig sind, wurde eine Methode vorgestellt. Hierbei wurden die unvollständigen DvBs in weitere Untergruppen unterteilt. Für die spätere Verwendung der unvollständigen DvBs wurde ein Programm zur Bestimmung des untersten und des obersten gültigen Teilworts eines DvBs präsentiert. Weiterhin kann mit dem Programm 6.2 ermittelt werden, ob die DvBs eines Partitionsraums $\widehat{\mathcal{K}}$ jeweils nur in Teilwort besitzen.

6.2 Packoperationen durch den Zerfall der Abhängigkeitsvektoren

Wie in Abschnitt 4.2.2 gezeigt wurde, kann durch eine Copartitionierung für jeden Abhängigkeitsvektor $\mathbf{d} = \mathbf{d}(e) \in \mathcal{D}$ eine Menge $|\mathcal{D}^\kappa(e)|$ von partitionierten Abhängigkeitsvektoren $\mathbf{d}^\kappa = \mathbf{d}^\kappa(e) = \left(\mathbf{d}_l^S, \mathbf{d}_l^P, \widehat{\mathbf{d}}_l^P\right)^t \in \mathcal{D}^\kappa(e)$ mit drei Teilvektoren bestimmt werden. Unter den obigen Beschränkungen für die Copartitionierung sind die Teilvektoren $\mathbf{d}_l^S$ immer Nullvektoren, denn die LS-Partition hat die Größe $\mathbf{\Theta}^S = \mathbf{E}$. Weiterhin können durch die Vorgabe, dass die Partitionsmatrix $\mathbf{\Theta}^P$ nur ein Element größer eins ($\exists!\, \vartheta_j^P > 1$) besitzen darf, nur ein oder zwei partitionierte Abhängigkeitsvektoren $\mathbf{d}^\kappa(e) \in \mathcal{D}^\kappa(e)$ entstehen, denn die Menge $\mathcal{X}^2(\mathbf{d})$ (siehe (4.31) auf Seite 46) kann maximal ein Element enthalten.

Die Anzahl der Packoperationen, die für die teilwortparallele Berechnung des Algorithmus benötigt werden, ergibt sich aus der Anzahl der Abhängigkeitsvektoren $\mathbf{d}(e) \in \mathcal{D}$, welche in zwei partitionierte Abhängigkeitsvektoren ($|\mathcal{D}^\kappa(e)| = 2$) zerfallen.

Im Folgenden wird am Beispiel des Abhängigkeitsvektors $\mathbf{d}(e)$ die Ermittlung der TWP- und Packoperationen für die zwei möglichen Fälle mit ein oder zwei partitionierten Abhängigkeitsvektoren vorgestellt. Man beachte, dass der Abhängigkeitsvektor $\mathbf{d}(e)$ den Transfer einer Instanz der Variable $y_{\sigma(e)}[.]$ von seiner Enstehung in Zuweisung $\sigma(e) = s'$ und Iteration $\mathbf{i}_{\sigma(e)}$ zur seiner Verwendung in Zuweisung $\delta(e) = s_t$ und Iteration $\mathbf{i}_{\delta(e)}$ beschreibt. Der folgende Algorithmenausschnitt mit dem Abhängigkeitsvektor $\mathbf{d}(e) = \mathbf{d}$ soll betrachtet werden:

$$s' : \quad y_{s'}[\mathbf{i}] = \ldots, \qquad \mathbf{i} \in \mathcal{I}_{s'}, \tag{6.27}$$

$$s : \quad y_s[\mathbf{i}] = F_{s_t}(\ldots, y_{s'}[\mathbf{i} - \mathbf{d}], \ldots), \qquad \mathbf{i} \in \mathcal{I}_{s_t}. \tag{6.28}$$

6.2.1 Ein partitionierter Abhängigkeitsvektor

Entsteht nur ein partitionierter Abhängigkeitsvektor $\mathbf{d}_1^\kappa = \left(\mathbf{d}_1^S, \mathbf{d}_1^P, \widehat{\mathbf{d}}_1^P\right)^t$ durch die Copartitionierung des Abhängigkeitsvektors $\mathbf{d}$, dann gilt $\mathbf{d}_1^P = \mathbf{0}$, wie aus (4.30) und (4.37) auf Seite 46 f. ersichtlich ist. Das heißt, die Position der Teilwörter in den DvBs bleibt unverändert. Somit können die parallel berechneten Teilwörter $y_{s'}[0, \widehat{\boldsymbol{\kappa}}^P], y_{s'}[1, \widehat{\boldsymbol{\kappa}}^P], \ldots,$ $y_{s'}[B-1, \widehat{\boldsymbol{\kappa}}^P]$, welche im DvB $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P]$ enthalten sind, an der Position $\widehat{\boldsymbol{\kappa}}^P + \widehat{\mathbf{d}}_1^P$ direkt verwendet werden. Es wird keine Packoperation benötigt. Die Zuweisungen s' und die

Teilzuweisung s_t in Zuweisung s werden nun wie folgt in TWP-Zuweisungen überführt:

$$s' : \quad \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P] = \ldots, \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s'}, \tag{6.29}$$

$$s : \quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \boldsymbol{F}_{s_t}(\ldots, \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_1^P], \ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t}, \tag{6.30}$$

dabei repräsentiert die Operation $\boldsymbol{F}_{s_t}$ die B-fach parallele Berechnung der ursprünglichen Operation F_{s_t}. Die Partitionsteilräume $\widehat{\mathcal{K}}_{s'}$ und $\widehat{\mathcal{K}}_{s_t}$ erhalten wir durch Fourier-Motzkin-Elimination (kurz: FME) der Variablen κ_i^S und κ^P aus den partitionierten Iterationsräumen $\mathcal{I}_{s'}^{\kappa}$ und $\mathcal{I}_{s_t}^{\kappa}$, wie in Abschnitt 4.2 ab Seite 40 beschrieben. Die Partitionsteilräume $\widehat{\mathcal{K}}_{s'}$ und $\widehat{\mathcal{K}}_{s_t}$ sind die Iterationsräume des transformierten Algorithmus.

Beispiel 6.6
Wir nehmen an, der Abhängigkeitsvektor $\mathbf{d}$ *in der Zuweisung 6.28 habe die Länge* $\mathbf{d} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$. *Für die Iterationsräume* $\mathcal{I}_{s'}$ *und* $\mathcal{I}_{s_t}$ *gelte:*

$$\mathcal{I}_{s'} = \left\{\mathbf{i} = \left(\begin{smallmatrix}i_1\\i_2\end{smallmatrix}\right) \mid 0 \leq i_1 < 8 \wedge 0 \leq i_2 < 4\right\} \quad \textit{und} \tag{6.31}$$

$$\mathcal{I}_{s_t} = \left\{\mathbf{i} = \left(\begin{smallmatrix}i_1\\i_2\end{smallmatrix}\right) \mid 0 \leq i_1 < 8 \wedge 1 \leq i_2 < 4\right\}. \tag{6.32}$$

Der gemeinsame Iterationsraum $\mathcal{I} = \mathcal{I}_{s'} \cup \mathcal{I}_{s_t} = \mathcal{I}_{s'}$ *wird nun mit den Parametern* $\boldsymbol{\Theta}^S = \mathbf{E}$ *und* $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix}4&0\\0&1\end{smallmatrix}\right)$ *partitioniert, um ein DvB mit vier Teilwörtern zu erhalten. Abbildung 6.5(a) zeigt das eben beschriebene Szenario.*

Die Partitionierung des Abhängigkeitsvektors $\mathbf{d} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *ergibt einen partitionierten Abhängigkeitsvektor* $\mathbf{d}^{\kappa} = (\mathbf{d}^S, \mathbf{d}^P, \widehat{\mathbf{d}}^P)^t$ *mit* $\mathbf{d}^S = \mathbf{0}$, $\mathbf{d}^P = \mathbf{0}$ *und* $\widehat{\mathbf{d}}^P = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$. *Wie die Abbildung 6.5(a) für die Iteration* $\widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *zeigt, wurden alle* $y_{s'}$, *die in dieser Iteration benötigt werden, in Iteration* $\widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *berechnet. Die relative Position der Teilwörter im DvB ändert sich zwischen Erzeugung und Verwendung nicht (*$\mathbf{d}^P = \mathbf{0}$*). Somit muss das DvB nur von Iteration* $\widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *nach Iteration* $\widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *transferiert werden, was durch den Teilvektor* $\widehat{\mathbf{d}}^P = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *beschrieben wird.*

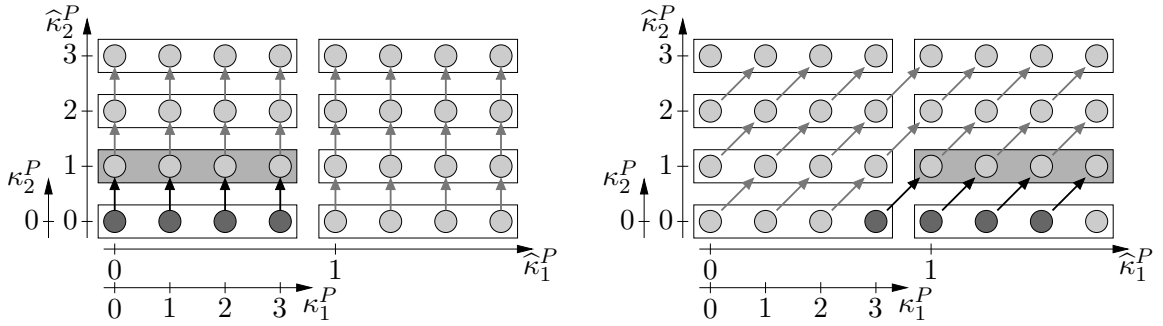
Für die Iterationsräume $\widehat{\mathcal{K}}_{s'}$ *und* $\widehat{\mathcal{K}}_{s_t}$ *erhalten wir:*

$$\widehat{\mathcal{K}}_{s'} = \underset{\kappa_1^S,\kappa_2^S,\kappa_1^P,\kappa_2^P}{\mathrm{FME}}(\mathcal{I}_{s'}^{\kappa}) = \{0 \leq \widehat{\kappa}_1^P < 2 \wedge 0 \leq \widehat{\kappa}_2^P < 4\} \quad \textit{und} \tag{6.33}$$

$$\widehat{\mathcal{K}}_{s_t} = \underset{\kappa_1^S,\kappa_2^S,\kappa_1^P,\kappa_2^P}{\mathrm{FME}}(\mathcal{I}_{s_t}^{\kappa}) = \{0 \leq \widehat{\kappa}_1^P < 2 \wedge 1 \leq \widehat{\kappa}_2^P < 4\}. \tag{6.34}$$

6.2.2 Zwei partitionierte Abhängigkeitsvektoren

Wenn sich zwei partitionierte Abhängigkeitsvektoren $\mathbf{d}_l^{\kappa} = \left(\mathbf{d}_l^S, \mathbf{d}_l^P, \widehat{\mathbf{d}}_l^P\right)^t$, $l = 1, 2$ aus der Copartitionierung des Abhängigkeitsvektors $\mathbf{d}$ ergeben, werden die berechneten Teilwörter des DvBs $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P]$ in zwei verschiedenen DvBs für die weitere Berechnung benötigt, wobei jedes Teilwort $y_{s'}[b, \widehat{\boldsymbol{\kappa}}^P]$ nur einmal verwendet wird.



(a) ohne Packoperation (b) mit Packoperation

Abbildung 6.5: Datentransfer im partitionierten Iterationsraum

Aufgrund der Tatsache, dass die Teilwörter nur in DvBs übertragen werden können, muss für die weitere Berechnung das ursprüngliche DvB $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P]$ zu diesen zwei DvBs übermittelt werden. Diese Übertragung ist bestimmt durch die Teilvektoren $\widehat{\mathbf{d}}_1^P$ und $\widehat{\mathbf{d}}_2^P$.

Eine Packoperation F_p wird nun benötigt, um die Teilwörter, welche für die Berechnung in der Iteration $\widehat{\boldsymbol{\kappa}}^P$ in den zwei DvBs $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_1^P]$ und $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_2^P]$ zur Verfügung stehen, in ein neues DvB $\mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^P]$ zu packen. Das neue DvB $\mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^P]$ mit den richtig angeordneten Teilwörtern $y_{s'}[\mathbf{i}]$ wird anschließend in Teilzuweisung s_t für die parallele Berechnung in Iteration $\widehat{\boldsymbol{\kappa}}^P$ verwendet. Somit ergeben sich folgende Zuweisungen:

$$s': \quad \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P] = \ldots, \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s'}, \tag{6.35}$$

$$p: \quad \mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} F_p(\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_1^P], \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_2^P]), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t} \cap \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_1^P} \cap \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_1^P}, \\ F_p(\mathbf{0}, \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_2^P]), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t} \backslash \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_1^P}, \\ F_p(\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_1^P], \mathbf{0}), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t} \backslash \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_2^P}, \end{cases} \tag{6.36}$$

$$s: \quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \boldsymbol{F}_{s_t}(\ldots, \mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^P], \ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t}, \tag{6.37}$$

mit $\widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_1^P} = f(\widehat{\mathcal{K}}_{s'})$ mit $f(\widehat{\boldsymbol{\kappa}}^P) = \mathbf{E} \cdot \widehat{\boldsymbol{\kappa}}^P + \widehat{\mathbf{d}}_1^P$ und $\widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_2^P} = f(\widehat{\mathcal{K}}_{s'})$ mit $f(\widehat{\boldsymbol{\kappa}}^P) = \mathbf{E} \cdot \widehat{\boldsymbol{\kappa}}^P + \widehat{\mathbf{d}}_2^P$. Die Iterationsräume der $\widehat{\mathcal{K}}_{s'}$ und $\widehat{\mathcal{K}}_{s_t}$ des transformierten Algorithmus ergeben sich wieder durch die Fourier-Motzkin-Elimination aus den partitionierten Iterationsräumen $\mathcal{I}_{s'}^{\kappa}$ und $\mathcal{I}_{s_t}^{\kappa}$.

Die drei Teilzuweisungen in Packzuweisung p sind notwendig, um ungültige Datenzugriffe auf nicht existierende DvBs $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_1^P]$ oder $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_2^P]$ zu vermeiden. Die Existenz eines DvBs kann mit der Mengenoperation $\widehat{\mathcal{K}}_{s_t} \cap \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_1^P} \cap \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_2^P}$ (Vereinigungsmenge) und deren Fehlen durch die Mengenoperationen $\widehat{\mathcal{K}}_{s_t} \backslash \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_1^P}$ und $\widehat{\mathcal{K}}_{s_t} \backslash \widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_2^P}$ (Differenzmenge) ermittelt werden. Existiert das benötigte DvB nicht, so wird es durch ein konstantes DvB „$\mathbf{0}$“ ersetzt, bei dem alle Teilwörter den Wert Null haben, wie in den Teilzuweisungen p_2 und p_3 der Zuweisung p gezeigt. Das konstante DvB mit den Nullwerten in den Teilwörtern bezeichnen wir zukünftig als Null-DvB.

Mit der Packoperation $\mathbf{y}_p = \big(y_p[0], y_p[1], \ldots, y_p[B-1]\big)^t = F_p(\mathbf{x}_1, \mathbf{x}_2)$ werden die Teilwörter der DvBs $\mathbf{x}_1$ und $\mathbf{x}_2$ den Teilwörtern des DvBs $\mathbf{y}_p$ korrekt zugewiesen. Die Operation

kann mittels der inneren Teilvektoren $\mathbf{d}_1^P$ und $\mathbf{d}_2^P$ wie folgt hergeleitet werden:

$$y_p[b] = \begin{cases} x_1[b - d_1^P], & \text{wenn } 0 \leq b - d_1^P < \vartheta^P = B, \\ x_2[b - d_2^P], & \text{wenn } 0 \leq b - d_2^P < \vartheta^P = B, \end{cases} \tag{6.38}$$

mit $d_j^P = \mathbf{\Gamma} \cdot \mathbf{d}_j^P$ und $\vartheta^P = \mathbf{\Gamma} \cdot \boldsymbol{\vartheta}^P$.

Aufgrund der Definition in (6.38) und den Beschränkungen für die Teilvektoren (siehe Abschnitt 4.2.2) gilt folgendes:

(1) $d_1^P \geq 0$,

(2) $d_2^P < 0$ und

(3) $|d_1^P| + |d_2^P| = \vartheta^P = B$, wenn $d_1^P > 0$.

Hilfssatz 6.1
Für die Elemente d_1^P und d_2^P gilt unter der Bedingung, dass $d_1^P > 0$ ist:

$$|d_1^P| + |d_2^P| = B. \tag{6.39}$$

Beweis 6.1
Entsprechend der Vorschrift zur Bestimmung der partitionieren Abhängigkeitsvektoren aus Abschnitt 4.2.2 (siehe (4.36) und (4.37)) gilt:

$$\left| d - B \cdot \left\lfloor \frac{d}{B} \right\rfloor \right| + \left| d - B \cdot \left\lceil \frac{d}{B} \right\rceil \right| = B, \tag{6.40}$$

$$B\Bigg(\underbrace{\left| \frac{d}{B} - \left\lfloor \frac{d}{B} \right\rfloor \right|}_{\geq 0} + \underbrace{\left| \frac{d}{B} - \left\lceil \frac{d}{B} \right\rceil \right|}_{\leq 0} \Bigg) = B, \tag{6.41}$$

$$B\left(\left(\frac{d}{B} - \left\lfloor \frac{d}{B} \right\rfloor \right) - \left(\frac{d}{B} - \left\lceil \frac{d}{B} \right\rceil \right) \right) = B, \tag{6.42}$$

$$B\left(\left\lceil \frac{d}{B} \right\rceil - \left\lfloor \frac{d}{B} \right\rfloor \right) = B, \tag{6.43}$$

wobei $d = \mathbf{\Gamma} \cdot \mathbf{d}$ ist. Für die Differenz in (6.43) gilt:

$$\left\lceil \frac{d}{B} \right\rceil - \left\lfloor \frac{d}{B} \right\rfloor = \begin{cases} 0, & \textit{wenn} \quad \frac{d}{B} = 0 \\ 1, & \textit{sonst.} \end{cases} \tag{6.44}$$

Da unter der Bedingung $d_1^P > 0$ auch $\frac{d}{B} \neq 0$ sein muss, folgt:

$$B = B. \tag{6.45}$$

Zwei Schlussfolgerungen können aus dem Beweis 6.1 geschlossen werden:

(1) Nur eine Variable d_1^P oder d_2^P muss bei die Bestimmung eines Packbefehls betrachtet werden, denn die jeweils andere Variable kann mittels B, der Anzahl an Teilwörtern im DvB, ermittelt werden.

(2) Die Variablen d_1^P bzw. d_2^P können nur B verschiedene Werte annehmen. Somit ergeben sich $B-1$ verschiedene Packoperationen[2].

Aus diesem Grund wollen wir nachfolgend den Parameter d_1^P in der Packoperation $\mathbf{y} = F_p\big(\mathbf{x}_1, \mathbf{x}_2, d_1^P\big)$, direkt mit angeben.

Beispiel 6.7
Der Abhängigkeitsvektor $\mathbf{d}$ *der Zuweisung 6.28 habe nun die Länge* $\mathbf{d} = \binom{1}{1}$. *Die Iterationsräume* $\mathcal{I}_{s'}$ *und* $\mathcal{I}_{s_t}$ *haben die Größe:*

$$\mathcal{I}_{s'} = \Big\{\mathbf{i} = \tbinom{i_1}{i_2} \mid 0 \le i_1 < 8 \wedge 0 \le i_2 < 4\Big\} \quad \textit{und} \tag{6.46}$$
$$\mathcal{I}_{s_t} = \Big\{\mathbf{i} = \tbinom{i_1}{i_2} \mid 1 \le i_1 < 8 \wedge 1 \le i_2 < 4\Big\}. \tag{6.47}$$

Für die Copartitionierung gelten die Vorgaben aus Beispiel 6.6. Wir erhalten somit wieder DvBs mit vier Teilwörtern. Abbildung 6.5(b) illustriert die Copartitionierung des gemeinsamen Iterationsraums $\mathcal{I} = \mathcal{I}_{s'} \cup \mathcal{I}_{s_t} = \mathcal{I}_{s'}$.

Die Copartitionierung des Abhängigkeitsvektors $\mathbf{d} = \binom{1}{1}$ *ergibt zwei partitionierte Abhängigkeitsvektoren* $\mathbf{d}_l^\kappa = (\mathbf{d}_l^S, \mathbf{d}_l^P, \widehat{\mathbf{d}}_l^P)^t$, $l = 1, 2$ *mit:*

$$\mathbf{d}_1^S = \mathbf{0}, \qquad \mathbf{d}_1^P = \tbinom{1}{0}, \qquad \widehat{\mathbf{d}}_1^P = \tbinom{0}{1} \textit{ sowie} \tag{6.48}$$
$$\mathbf{d}_2^S = \mathbf{0}, \qquad \mathbf{d}_2^P = \tbinom{-3}{0}, \qquad \widehat{\mathbf{d}}_2^P = \tbinom{1}{1}. \tag{6.49}$$

Für Iteration $\widehat{\boldsymbol{\kappa}}^P = \binom{1}{1}$ *des transformierten Algorithmus, in der die Teilwörter* $y_{s'}[\binom{3}{0}]$, $y_{s'}[\binom{4}{0}]$, $y_{s'}[\binom{5}{0}]$ *und* $y_{s'}[\binom{6}{0}]$ *benötigt werden, soll der Datentransfer und die Packoperation diskutiert werden. Das Teilwort* $y_{s'}[\binom{3}{0}]$ *wird in Iteration* $\widehat{\boldsymbol{\kappa}}^P = \binom{0}{0}$ *berechnet. Die Erzeugung der anderen Teilwörter findet in Iteration* $\widehat{\boldsymbol{\kappa}}^P = \binom{1}{0}$ *statt. Somit befinden sich die in Iteration* $\widehat{\boldsymbol{\kappa}}^P = \binom{1}{1}$ *benötigten Teilwörter in den DvBs* $\mathbf{y}_{s'}[\binom{0}{0}]$ *und* $\mathbf{y}_{s'}[\binom{1}{0}]$, *welche in den entsprechenden Iterationen erzeugt wurden. Beide DvBs müssen zu Iteration* $\widehat{\boldsymbol{\kappa}}^P = \binom{1}{1}$ *transferiert werden. Die beiden äußeren Teilvektoren* $\widehat{\mathbf{d}}_2^P = \binom{1}{1}$ *und* $\widehat{\mathbf{d}}_1^P = \binom{0}{1}$ *beschreiben diesen DvB-Transfer.*

Mit der Packoperation F_p *müssen nun die für die Berechnung in Iteration* $\widehat{\boldsymbol{\kappa}}^P = \binom{1}{1}$ *benötigten Teilwörter* $y_{s'}[\binom{3}{0}]$, $y_{s'}[\binom{4}{0}]$, $y_{s'}[\binom{5}{0}]$ *und* $y_{s'}[\binom{6}{0}]$ *in das DvB* $\mathbf{y}_p[\binom{1}{1}]$ *gepackt werden. Die allgemeine Packzuweisung für unser Beispiel lautet:*

$$p: \quad \mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} F_p\big(\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \binom{0}{1}], \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \binom{1}{1}], 1\big), & \widehat{\boldsymbol{\kappa}}^P = \widehat{\mathcal{K}}_{s_t} \cap \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_1^P} \cap \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_2^P}, \\ F_p\big(\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \binom{0}{1}], \mathbf{0}, 1\big), & \widehat{\boldsymbol{\kappa}}^P = \widehat{\mathcal{K}}_{s_t} \backslash \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_2^P}. \end{cases} \tag{6.50}$$

[2]Wenn $d_1^P = 0$ ist, wird keine Packzuweisung und somit deine Packoperation benötigt.

Für die Iterationsräume $\widehat{\mathcal{K}}_{s'}$, $\widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_2^P}$ und $\widehat{\mathcal{K}}_{s_t}$ und deren Vereinigungs- und Schnittemengen erhalten wir:

$$\begin{aligned}
\widehat{\mathcal{K}}_{s'} &= \underset{\kappa_1^S,\kappa_2^S,\kappa_1^P,\kappa_2^P}{\mathrm{FME}}(\mathcal{I}_{s'}^{\kappa}) = \left\{\widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix}\widehat{\kappa}_1^P\\ \widehat{\kappa}_2^P\end{pmatrix} \mid 0 \le \widehat{\kappa}_1^P < 2 \wedge 0 \le \widehat{\kappa}_2^P < 4\right\},\\
\widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_1^P} &= \left\{\widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix}\widehat{\kappa}_1^P\\ \widehat{\kappa}_2^P\end{pmatrix} \mid 0 \le \widehat{\kappa}_1^P < 2 \wedge 1 \le \widehat{\kappa}_2^P < 5\right\},\\
\widehat{\mathcal{K}}_{s'}^{\widehat{\mathbf{d}}_2^P} &= \left\{\widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix}\widehat{\kappa}_1^P\\ \widehat{\kappa}_2^P\end{pmatrix} \mid 1 \le \widehat{\kappa}_1^P < 3 \wedge 1 \le \widehat{\kappa}_2^P < 5\right\},\\
\widehat{\mathcal{K}}_{s_t} &= \underset{\kappa_1^S,\kappa_2^S,\kappa_1^P,\kappa_2^P}{\mathrm{FME}}(\mathcal{I}_{s_t}^{\kappa}) = \left\{\widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix}\widehat{\kappa}_1^P\\ \widehat{\kappa}_2^P\end{pmatrix} \mid 0 \le \widehat{\kappa}_1^P < 2 \wedge 1 \le \widehat{\kappa}_2^P < 4\right\},\\
\widehat{\mathcal{K}}_{s_t} \cap \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_1^P} \cap \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_2^P} &= \left\{\widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix}\widehat{\kappa}_1^P\\ \widehat{\kappa}_2^P\end{pmatrix} \mid \widehat{\kappa}_1^P = 1 \wedge 1 \le \widehat{\kappa}_2^P < 4\right\},\\
\widehat{\mathcal{K}}_{s_t} \backslash \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_1^P} &= \emptyset \quad \textit{und}\\
\widehat{\mathcal{K}}_{s_t} \backslash \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_2^P} &= \left\{\widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix}\widehat{\kappa}_1^P\\ \widehat{\kappa}_2^P\end{pmatrix} \mid \widehat{\kappa}_1^P = 0 \wedge 1 \le \widehat{\kappa}_2^P < 4\right\}.
\end{aligned}$$

Da die Differenzmenge $\widehat{\mathcal{K}}_{s_t} \backslash \mathcal{K}_{s'}^{\widehat{\mathbf{d}}_1^P}$ eine leere Menge ist, kann die zugehörige Teilzuweisung in der Packzuweisung p entfallen.

6.3 Packoperationen aufgrund mehrerer Teilzuweisungen in einer Zuweisung

Besteht eine Zuweisung s aus mehreren Teilzuweisungen, so kann es geschehen, dass die Teilwörter eines DvBs $\mathbf{y}_s[.]$ durch unterschiedliche Teilzuweisungen dieser Zuweisung bestimmt werden müssen. Abbildung 6.7 zeigt ein solches Szenario für zwei Teilzuweisungen s_1 und s_2 mit deren Iterationsräumen $\mathcal{I}_{s_1}$ und $\mathcal{I}_{s_2}$. Anhand dieses Beispiels soll das Problem vorgestellt und eine Lösung präsentiert werden. Anschließend erfolgt die Verallgemeinerung für T_s Teilzuweisungen in einer Zuweisung.

6.3.1 Zuweisung mit zwei Teilzuweisungen

Die Zuweisung s mit zwei Teilzuweisungen sieht wie folgt aus:

$$s: \quad y_s[\mathbf{i}] = \begin{cases} F_{s_1}(\ldots), & \mathbf{i} \in \mathcal{I}_{s_1},\\ F_{s_2}(\ldots), & \mathbf{i} \in \mathcal{I}_{s_2}. \end{cases} \tag{6.51}$$

Für die Iterationsräume $\mathcal{I}_{s_1}$ und $\mathcal{I}_{s_2}$ gilt:

$$\mathcal{I}_{s_1} = \left\{\mathbf{i} = \begin{pmatrix}i_1\\ i_2\end{pmatrix} \mid \begin{pmatrix}1 & 0\\ 0 & 1\\ -1 & 1\\ 0 & -1\end{pmatrix}\mathbf{i} \ge \begin{pmatrix}0\\ 0\\ -3\\ -5\end{pmatrix}\right\} \quad \text{und} \tag{6.52}$$

$$\mathcal{I}_{s_2} = \left\{\mathbf{i} = \begin{pmatrix}i_1\\ i_2\end{pmatrix} \mid \begin{pmatrix}1 & -1\\ 0 & 1\\ -1 & 0\\ 0 & -1\end{pmatrix}\mathbf{i} \ge \begin{pmatrix}4\\ 0\\ -11\\ -5\end{pmatrix}\right\}. \tag{6.53}$$

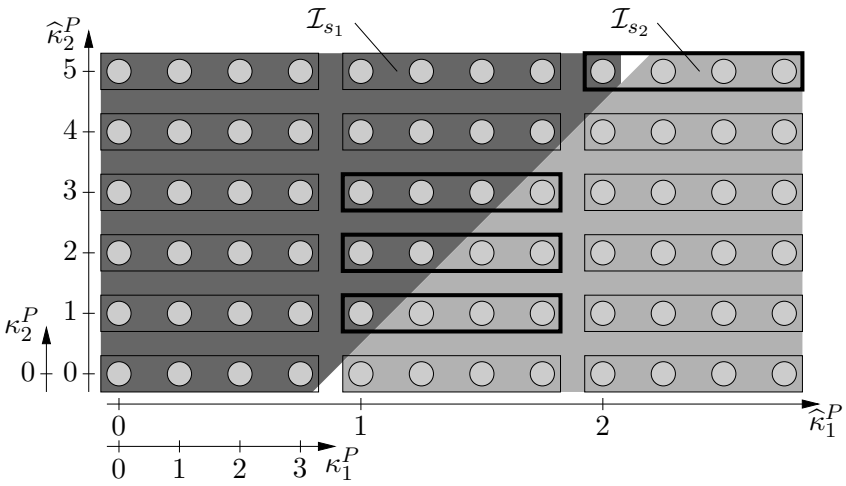


Abbildung 6.6: Partitionierung des Iterationsraums einer Zuweisung mit zwei Teilzuweisungen

Die Zuweisung s soll mit $B = 4$ Teilwörtern teilwortparallelisiert werden. Als Partitionsmatrix $\mathbf{\Theta}^P$ wählen wir $\mathbf{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$. Abbildung 6.6 zeigt die Iterationsräume $\mathcal{I}_{s_1}$ und $\mathcal{I}_{s_2}$ im partitionierten Iterationsraum. Aus dieser Darstellung ist gut zu erkennen, dass drei verschiedene Arten von DvBs unterschieden werden müssen:

(1) DvBs, in denen alle Teilwörter zur Teilzuweisung s_1 gehören.

(2) DvBs, in denen alle Teilwörter zur Teilzuweisung s_2 gehören.

(3) DvBs, in denen mindestens ein Teilwort zur Teilzuweisung s_1 und mindestens ein Teilwort zur Teilzuweisung s_2 gehört.

Die DvBs, welche Teilwörter beider Iterationsräume enthalten, sind in Abbildung 6.6 dick umrandet dargestellt.

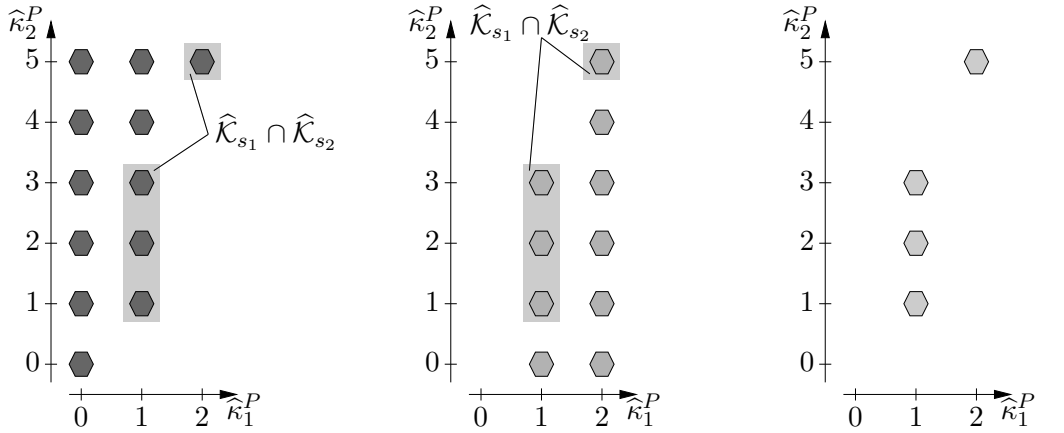
Die Iterationsräume für die drei Versionen können über die Partitionsräume $\widehat{\mathcal{K}}_{s_1}$ und $\widehat{\mathcal{K}}_{s_2}$ ermittelt werden:

$$\widehat{\mathcal{K}}_{s_1} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 4 & 0 \\ 0 & 1 \\ -4 & 1 \\ 0 & -1 \end{pmatrix} \widehat{\boldsymbol{\kappa}}^P \geq \begin{pmatrix} -3 \\ 0 \\ -3 \\ -5 \end{pmatrix} \right\}, \tag{6.54}$$

$$\widehat{\mathcal{K}}_{s_2} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 4 & -1 \\ 0 & 1 \\ -4 & 0 \\ 0 & -1 \end{pmatrix} \widehat{\boldsymbol{\kappa}}^P \geq \begin{pmatrix} 1 \\ 0 \\ -11 \\ -5 \end{pmatrix} \right\}. \tag{6.55}$$

In den Abbildungen 6.7(a) und 6.7(b) sind diese beiden Partitionsräume dargestellt. Die Partitionsräume $\widehat{\mathcal{K}}_{s_t}$ beinhalten alle Partitionen $\widehat{\boldsymbol{\kappa}}^P$, in denen mindestens ein Element b im Iterationsraum $\mathcal{I}_{s_t}$ enthalten ist[3]. Die Schnittmenge $\widehat{\mathcal{K}}_\cap = \widehat{\mathcal{K}}_{s_1} \cap \widehat{\mathcal{K}}_{s_2}$ der beiden Partitionsräume beinhaltet alle Partitionen $\widehat{\boldsymbol{\kappa}}^P$, bei denen mindestens ein Element b_1 im Iterationsraum $\mathcal{I}_{s_1}$ und mindestens ein Element b_2 im Iterationsraum $\mathcal{I}_{s_2}$ enthalten

[3] $\mathbf{i} = \vartheta(\mathbf{i}^\kappa) = \vartheta((\mathbf{0},\ b \cdot \mathbf{\Gamma}^t,\ \widehat{\boldsymbol{\kappa}}^P)^t) \in \mathcal{I}_{s_t}$



(a) Partitionsraum $\widehat{\mathcal{K}}_{s_1}$ (b) Partitionsraum $\widehat{\mathcal{K}}_{s_2}$ (c) Vereinigung $\widehat{\mathcal{K}}_{s_1} \cap \widehat{\mathcal{K}}_{s_2}$

Abbildung 6.7: Partitionsräume einer Zuweisung mit zwei Teilzuweisungen

ist, wie Abbildung 6.7(c) zeigt. Die Räume, in denen die Elemente b einer Partition $\widehat{\boldsymbol{\kappa}}^P$ in dem Iterationsraum $\mathcal{I}_{s_t}$ **einer** Teilzuweisung s_t enthalten sind oder aber außerhalb des Iterationsraums $\mathcal{I}_s$ der Zuweisung s liegen, können somit mit Mengenoperationen $\widehat{\mathcal{K}}_{s_t} \backslash \widehat{\mathcal{K}}_{\cap}$ (Differenzmenge) ermittelt werden.

Gehören alle Teilwörter eines DvBs nur zu einer Teilzuweisung, so kann die Transformation der Teilzuweisung entsprechend der in Abschnitt 6.2 beschriebenen Methode erfolgen. Für die DvBs mit Teilwörtern beider Teilzuweisungen muss zuerst je ein DvB mit Teilwörtern der Teilzuweisung s_1 bzw. s_2 bestimmt werden, welche anschließend mit einer Packoperation F_m zu dem DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ zusammengefügt wird. Die Bestimmung der beiden DvBs mit Teilwörtern der Teilzuweisung s_1 bzw. s_2 erfolgt wieder entsprechend der in Abschnitt 6.2 beschriebenen Methode. Diese DvBs enthalten aber nur teilweise gültige Werte. Erst durch die Packoperation F_m entsteht ein vollständig gültiges DvB.

Somit erfolgt in allen Iterationen $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1}$ die teilwortparallele Berechnung der Teilzuweisung s_1 und in allen Iterationen $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_2}$ die teilwortparallele Berechnung der Teilzuweisung s_2. Da die Partitionsräume $\widehat{\mathcal{K}}_{s_1}$ und $\widehat{\mathcal{K}}_{s_2}$ nicht disjunkt sind, müssen zwei zusätzliche Zuweisungen v_{s_1} und v_{s_2} vor s eingefügt werden. Anschließend kann in der teilwortparallelen Zuweisung s dem DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ das DvB $\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P]$, das DvB $\mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P]$ oder das mit Packoperation F_m aus den DvBs $\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P]$ und $\mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P]$ gepackte DvB zugewiesen werden:

$$v_{s_1}: \quad \mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{F}_{s_1}(\ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1}, \tag{6.56}$$

$$v_{s_2}: \quad \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{F}_{s_2}(\ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_2}, \tag{6.57}$$

$$s: \quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} \mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1} \backslash \widehat{\mathcal{K}}_{s_2}, \\ \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_2} \backslash \widehat{\mathcal{K}}_{s_1}, \\ F_m(\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1} \cap \widehat{\mathcal{K}}_{s_2}. \end{cases} \tag{6.58}$$

Die Packoperation F_m ist im Allgemeinen abhängig von der Iteration $\widehat{\boldsymbol{\kappa}}^P$. Die Iterations-

räume der Teilzuweisungen der teilwortparallelen Zuweisung s sind disjunkt.

Für die Packoperation $\mathbf{y} = F_m(\mathbf{x}_1, \mathbf{x}_2, \widehat{\boldsymbol{\kappa}}^P)$ gilt:

$$y[b] = \begin{cases} x_1[b], & \text{wenn} \quad \begin{pmatrix} 1 \\ -1 \\ \mathbf{A}_{s_1}\boldsymbol{\Gamma}^t \end{pmatrix} \cdot b \geq \begin{pmatrix} 0 \\ -B+1 \\ \mathbf{a}_{0,s_1} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{A}_{s_1}\boldsymbol{\Theta}^P \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P, \\ x_2[b], & \text{wenn} \quad \begin{pmatrix} 1 \\ -1 \\ \mathbf{A}_{s_2}\boldsymbol{\Gamma}^t \end{pmatrix} \cdot b \geq \begin{pmatrix} 0 \\ -B+1 \\ \mathbf{a}_{0,s_2} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{A}_{s_2}\boldsymbol{\Theta}^P \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P, \end{cases} \tag{6.59}$$

wobei $\mathbf{A}_{s_i}$ und $\mathbf{a}_{0,s_i}$, $i = 1, 2$ die Polyedermatrix und der Polyedervektor aus der Polyederbeschreibung der Iterationsräume $\mathcal{I}_{s_i}$ der Teilzuweisung s_i sind. Für unsere Beispielzuweisung (6.51) von Seite 96 mit den Iterationsräumen aus (6.52) und (6.53) erhalten wir folgende Packoperation:

$$y_s[b, \widehat{\boldsymbol{\kappa}}^P] = \begin{cases} y_{v_{s_1}}[b, \widehat{\boldsymbol{\kappa}}^P], & \text{wenn} \quad 0 \leq b \leq 3 \wedge -4\widehat{\kappa}_1^P \leq b \leq 3 - 4\widehat{\kappa}_1^P + \widehat{\kappa}_2^P, \\ y_{v_{s_2}}[b, \widehat{\boldsymbol{\kappa}}^P], & \text{wenn} \quad 0 \leq b \leq 3 \wedge 4 - 4\widehat{\kappa}_1^P + \widehat{\kappa}_2^P \leq b \leq 11 - 4\widehat{\kappa}_1^P. \end{cases} \tag{6.60}$$

Für die Iteration $\widehat{\boldsymbol{\kappa}}^P$ gilt $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1} \cap \widehat{\mathcal{K}}_{s_2}$.

Wie aus den Bedingungen in (6.59) für die Auswahl der Variablen $x_1[b]$ oder $x_2[b]$ für die Variable $y[b]$ ersichtlich ist, ist der Raum, in dem der Iterator b gültig ist, ein eindimensionales konvexes Polyeder. Das heißt, für den Iterator b gilt je Bedingung immer: $b_{\min} \leq b \leq b_{\max}$.

Ausgehend von der Zuweisung s mit zwei Teilzuweisungen soll nun der allgemeine Fall mit T_s Teilzuweisungen diskutiert werden.

6.3.2 Zuweisung mit mehreren Teilzuweisungen

Besitzt die Zuweisung s mehrere Teilzuweisungen, deren Anzahl durch T_s angegeben wird, und hat das DvB B Teilwörter, so kann es möglich sein, dass in dem DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ Teilwörter aus bis zu $P = \min(B, T_s)$ verschiedenen Teilzuweisungen vorhanden sind. Um all diese Vielfalt beherrschen zu können, wird zuerst jede Teilzuweisung s_t als teilwortparallele Zuweisung v_{s_t} beschrieben:

$$v_{s_t}: \quad \mathbf{y}_{v_{s_t}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{F}_{s_t}(\ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t}, \qquad 1 \leq t \leq T_s. \tag{6.61}$$

Nun sollen alle möglichen Kombinationen von Teilzuweisungen s_t ermittelt werden, welche für ein DvB Daten liefern könnten. Hierzu ermitteln wir alle Kombinationen von P aus T_s Teilzuweisungen $\binom{P}{T_s}$, $P-1$ aus T_s Teilzuweisungen $\binom{P-1}{T_s}$, bis zu einer aus T_s Teilzuweisungen $\binom{1}{T_s}$. Alle Kombinationen von p Elementen $1 \leq p \leq P$ aus T_s Elementen sind in der Menge $\mathcal{C}^p_{T_s}$ zusammengefasst. Eine Kombination $\underline{\mathcal{C}}^p_{T_s} \in \mathcal{C}^p_{T_s}$ ist eine Menge von p Elementen s_t, $1 \leq t \leq T_s$.

Für jede Kombination $\underline{\mathcal{C}}^p_{T_s} \in \mathcal{C}^p_{T_s}$, $1 \leq p \leq P$ bestimmen wir die Schnittmenge der Partitionsräume $\widehat{\mathcal{K}}_{s_t}$ der Teilzuweisungen $s_t \in \underline{\mathcal{C}}^p_{T_s}$:

$$\widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^p_{T_s}} = \bigcap_{\forall s_t \in \underline{\mathcal{C}}^p_{T_s}} \widehat{\mathcal{K}}_{s_t}. \tag{6.62}$$

Gilt für die Menge $\widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^p_{T_s}} \neq \emptyset$, so existiert mindestens ein DvB, für das die Teilwörter aus den Teilzuweisungen $s_t \in \underline{\mathcal{C}}^p_{T_s}$ zusammengepackt werden müssen. Das heißt, die teilwortparallele Zuweisung s muss eine Packoperation $F_{m,\underline{\mathcal{C}}^p_{T_s}}$ mit den DvBs $\mathbf{y}_{v_{s_t}}[\widehat{\boldsymbol{\kappa}}^P]$, $s_t \in \underline{\mathcal{C}}^p_{T_s}$ als Variable der Packoperation erhalten. Die Iterationen $\widehat{\boldsymbol{\kappa}}^P$ in denen diese Teilzuweisung ausgeführt werden soll, ergibt sich aus der Differenzmenge $\widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^p_{T_s}} \setminus \widehat{\mathcal{K}}^{p+1}$, wobei $\widehat{\mathcal{K}}^p = \bigcup_{\forall \underline{\mathcal{C}}^p_{T_s} \in \mathcal{C}^p_{T_s}} \widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^p_{T_s}}$ die Vereinigung der Schnittmengen $\widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^p_{T_s}}$ für ein p ist.

Mit der Bildung der Differenzmenge wird gewährleistet, dass die Iterationsräume der Teilzuweisungen der teilwortparallelen Zuweisung s disjunkt sind. Denn wenn z. B. für die Schnittmenge $\widehat{\mathcal{K}}^3_{\underline{\mathcal{C}}^3_{T_s,i}}$ dreier Teilzuweisungen der Ausgangszuweisung s gilt: $\widehat{\mathcal{K}}^3_{\underline{\mathcal{C}}^3_{T_s,i}} \neq \emptyset$, so gibt es auch eine Schnittmenge $\widehat{\mathcal{K}}^2_{\underline{\mathcal{C}}^2_{T_s,j}}$ mit zwei Teilzuweisungen $s_t \in \underline{\mathcal{C}}^2_{T_s,j} \subset \underline{\mathcal{C}}^3_{T_s,i}$. Für die beiden Schnittmengen gilt: $\widehat{\mathcal{K}}^3_{\underline{\mathcal{C}}^3_{T_s,i}} \subset \widehat{\mathcal{K}}^2_{\underline{\mathcal{C}}^2_{T_s,j}}$. In allen Iterationen $\widehat{\boldsymbol{\kappa}}^P$ der Schnittmenge $\widehat{\mathcal{K}}^3_{\underline{\mathcal{C}}^3_{T_s,i}}$ muss die Packoperation $F_{m,\underline{\mathcal{C}}^3_{T_s,i}}$ mit den drei DvBs $\mathbf{y}_{v_{s_t}}$, $s_t \in \underline{\mathcal{C}}^3_{T_s,i}$ verwendet werden.

Allgemein folgt damit für die Teilzuweisung der Zuweisung s:

$$s: \quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} \dots \\ F_{m,\underline{\mathcal{C}}^p_{T_s}}(\mathbf{y}_{v_{s_t}}[\widehat{\boldsymbol{\kappa}}^P], s_t \in \underline{\mathcal{C}}^p_{T_s}, \widehat{\boldsymbol{\kappa}}^P), \quad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^p_{T_s}} \setminus \widehat{\mathcal{K}}^{p+1}, \\ \dots \end{cases}$$

$$\forall \widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^p_{T_s}} \neq \emptyset, \qquad 1 \leq p \leq P. \quad (6.63)$$

Für die allgemeine Packoperation $\mathbf{y} = F_{m,\underline{\mathcal{C}}^p_{T_s}}(\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_p, \widehat{\boldsymbol{\kappa}}^P)$ gilt:

$$y[b] = \begin{cases} x_1[b], & \text{wenn} \quad \begin{pmatrix} 1 \\ -1 \\ \mathbf{A}_{s_1}\boldsymbol{\Gamma}^t \end{pmatrix} \cdot b \geq \begin{pmatrix} 0 \\ -B+1 \\ \mathbf{a}_{0,s_1} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{A}_{s_1}\boldsymbol{\Theta}^P \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P, \\ x_2[b], & \text{wenn} \quad \begin{pmatrix} 1 \\ -1 \\ \mathbf{A}_{s_2}\boldsymbol{\Gamma}^t \end{pmatrix} \cdot b \geq \begin{pmatrix} 0 \\ -B+1 \\ \mathbf{a}_{0,s_2} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{A}_{s_2}\boldsymbol{\Theta}^P \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P, \\ \quad \vdots \\ x_p[b], & \text{wenn} \quad \begin{pmatrix} 1 \\ -1 \\ \mathbf{A}_{s_p}\boldsymbol{\Gamma}^t \end{pmatrix} \cdot b \geq \begin{pmatrix} 0 \\ -B+1 \\ \mathbf{a}_{0,s_p} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{A}_{s_p}\boldsymbol{\Theta}^P \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P. \end{cases} \quad (6.64)$$

Beispiel 6.8 (Teilwortparallelisierung einer Zuweisung mit vier Teilzuw.)
Die folgende Zuweisung s mit $T_s = 4$ Teilzuweisungen soll mit $B = 4$ Teilwörtern teilwortparallelisiert werden:

$$s: \quad y_s[\mathbf{i}] = \begin{cases} F_{s_1}(\dots), & \mathbf{i} \in \mathcal{I}_{s_1}, \\ F_{s_2}(\dots), & \mathbf{i} \in \mathcal{I}_{s_2}, \\ F_{s_3}(\dots), & \mathbf{i} \in \mathcal{I}_{s_3}, \\ F_{s_4}(\dots), & \mathbf{i} \in \mathcal{I}_{s_4}. \end{cases} \quad (6.65)$$

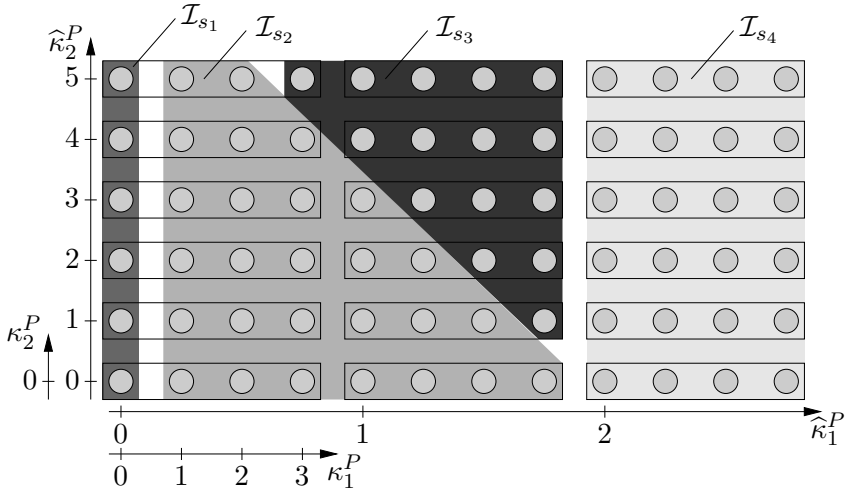


Abbildung 6.8: Teilwortparallelisierung einer Zuweisung mit vier Teilzuweisungen

Die Iterationsräume $\mathcal{I}_{s_i}$, $i = \{1, 2, 3, 4\}$ der Teilzuweisungen haben die folgenden Größen:

$$\mathcal{I}_{s_1} = \left\{ \mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \mathbf{i} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -5 \end{pmatrix} \right\}, \tag{6.66}$$

$$\mathcal{I}_{s_2} = \left\{ \mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \\ 0 & -1 \end{pmatrix} \mathbf{i} \geq \begin{pmatrix} 1 \\ 0 \\ -7 \\ -5 \end{pmatrix} \right\}, \tag{6.67}$$

$$\mathcal{I}_{s_3} = \left\{ \mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \mathbf{i} \geq \begin{pmatrix} 8 \\ -7 \\ -5 \end{pmatrix} \right\} \quad \textit{und} \tag{6.68}$$

$$\mathcal{I}_{s_4} = \left\{ \mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \mathbf{i} \geq \begin{pmatrix} 8 \\ 0 \\ -11 \\ -5 \end{pmatrix} \right\}. \tag{6.69}$$

Abbildung 6.8 zeigt diese Iterationsräume im partitionierten Iterationsraum, wobei die Iterationsräume mit $\mathbf{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ partitioniert wurden. Die Partitionsräume $\widehat{\mathcal{K}}_{s_i}$ für die partitionierten Teilzuweisungen lauten:

$$\widehat{\mathcal{K}}_{s_1} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \widehat{\boldsymbol{\kappa}}^P \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -5 \end{pmatrix} \right\}, \tag{6.70}$$

$$\widehat{\mathcal{K}}_{s_2} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -4 & -1 \\ 0 & -1 \end{pmatrix} \widehat{\boldsymbol{\kappa}}^P \geq \begin{pmatrix} 0 \\ 0 \\ -7 \\ -5 \end{pmatrix} \right\}, \tag{6.71}$$

$$\widehat{\mathcal{K}}_{s_3} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 4 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \widehat{\boldsymbol{\kappa}}^P \geq \begin{pmatrix} 5 \\ -1 \\ -5 \end{pmatrix} \right\} \quad \textit{und} \tag{6.72}$$

$$\widehat{\mathcal{K}}_{s_4} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \widehat{\boldsymbol{\kappa}}^P \geq \begin{pmatrix} 2 \\ 0 \\ -2 \\ -5 \end{pmatrix} \right\}. \tag{6.73}$$

Theoretisch kann ein DvB mit B Teilwörtern aus $P = \min(B, T_s) = 4$ verschiedenen Teilzuweisungen entstehen. Somit bestimmen wir zuerst alle Kombinationsmengen $\mathcal{C}_{4,i}^4$,

$\mathcal{C}^3_{4,i}$, $\mathcal{C}^2_{4,i}$ *und* $\mathcal{C}^1_{4,i}$*:*

$$\mathcal{C}^4_{4,1} = \{s_1, s_2, s_3, s_4\},$$

$$\mathcal{C}^3_{4,1} = \{s_1, s_2, s_3\}, \quad \mathcal{C}^3_{4,2} = \{s_1, s_2, s_4\}, \quad \mathcal{C}^3_{4,3} = \{s_1, s_3, s_4\}, \quad \mathcal{C}^3_{4,4} = \{s_2, s_3, s_4\},$$

$$\mathcal{C}^2_{4,1} = \{s_1, s_2\}, \quad \mathcal{C}^2_{4,2} = \{s_1, s_3\}, \quad \mathcal{C}^2_{4,3} = \{s_1, s_4\},$$
$$\mathcal{C}^2_{4,4} = \{s_2, s_3\}, \quad \mathcal{C}^2_{4,5} = \{s_2, s_4\}, \quad \mathcal{C}^2_{4,6} = \{s_3, s_4\},$$

$$\mathcal{C}^1_{4,1} = \{s_1\}, \quad \mathcal{C}^1_{4,2} = \{s_2\}, \quad \mathcal{C}^1_{4,3} = \{s_3\}, \quad \mathcal{C}^1_{4,4} = \{s_4\}.$$

Für jede Kombinationsmenge $\mathcal{C}^p_{4,i}$ *ermitteln wir nun die zugehörige Schnittmenge* $\widehat{\mathcal{K}}^p_{4,i} = \bigcap_{s_t \in \mathcal{C}^p_{4,i}} \widehat{\mathcal{K}}_{s_t}$*:*

$$\widehat{\mathcal{K}}^4_{4,1} = \emptyset,$$

$$\widehat{\mathcal{K}}^3_{4,1} = \{\binom{0}{5}\}, \quad \widehat{\mathcal{K}}^3_{4,2} = \emptyset, \quad \widehat{\mathcal{K}}^3_{4,3} = \emptyset, \quad \widehat{\mathcal{K}}^3_{4,4} = \emptyset,$$

$$\widehat{\mathcal{K}}^2_{4,1} = \{\binom{0}{0}, \binom{0}{1}, \ldots, \binom{0}{5}\}, \quad \widehat{\mathcal{K}}^2_{4,2} = \{\binom{0}{5}\}, \quad \widehat{\mathcal{K}}^2_{4,3} = \emptyset,$$
$$\widehat{\mathcal{K}}^2_{4,4} = \{\binom{0}{5}, \binom{1}{1}, \binom{1}{2}, \binom{1}{3}\}, \quad \widehat{\mathcal{K}}^2_{4,5} = \emptyset, \quad \widehat{\mathcal{K}}^2_{4,6} = \emptyset,$$

$$\widehat{\mathcal{K}}^1_{4,1} = \widehat{\mathcal{K}}_{s_1}, \quad \widehat{\mathcal{K}}^1_{4,2} = \widehat{\mathcal{K}}_{s_2}, \quad \widehat{\mathcal{K}}^1_{4,3} = \widehat{\mathcal{K}}_{s_3}, \quad \widehat{\mathcal{K}}^1_{4,4} = \widehat{\mathcal{K}}_{s_4}.$$

Aus diesen Schnittmengen können die Räume $\widehat{\mathcal{K}}^p$*,* $2 \leq p \leq P$ *bestimmt werden:*

$$\widehat{\mathcal{K}}^4 = \emptyset,$$
$$\widehat{\mathcal{K}}^3 = \{\binom{0}{5}\},$$
$$\widehat{\mathcal{K}}^2 = \{\binom{0}{0}, \binom{0}{1}, \ldots, \binom{0}{5}, \binom{1}{1}, \binom{1}{2}, \binom{1}{3}\}.$$

Somit erhalten wir nun die folgenden TWP-Zuweisungen durch die Transformation der Zuweisung 6.65:

$$v_{s_1}: \quad \mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{F}_{s_1}(\ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1},$$

$$v_{s_2}: \quad \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{F}_{s_2}(\ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_2},$$

$$v_{s_3}: \quad \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{F}_{s_3}(\ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_3},$$

$$v_{s_4}: \quad \mathbf{y}_{v_{s_4}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{F}_{s_4}(\ldots), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_4},$$

$$s: \quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} F^3_{m,1}(\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}^3_{4,1} \backslash \widehat{\mathcal{K}}^4, \\ F^2_{m,1}(\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}^2_{4,1} \backslash \widehat{\mathcal{K}}^3, \\ F^2_{m,4}(\mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}^2_{4,4} \backslash \widehat{\mathcal{K}}^3, \\ \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_2} \backslash \widehat{\mathcal{K}}^2, \\ \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_3} \backslash \widehat{\mathcal{K}}^2, \\ \mathbf{y}_{v_{s_4}}[\widehat{\boldsymbol{\kappa}}^P], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_4} \backslash \widehat{\mathcal{K}}^2. \end{cases}$$

Die Differenzmengen ergeben:

$$\widehat{\mathcal{K}}^3_{4,1}\backslash\widehat{\mathcal{K}}^4 = \{\binom{0}{5}\}, \qquad \widehat{\mathcal{K}}^2_{4,1}\backslash\widehat{\mathcal{K}}^3 = \{\binom{0}{0}, \binom{0}{1}, \ldots, \binom{0}{4}\},$$
$$\widehat{\mathcal{K}}^2_{4,4}\backslash\widehat{\mathcal{K}}^3 = \{\binom{1}{1}, \binom{1}{2}, \binom{1}{3}\}, \qquad \widehat{\mathcal{K}}_{s_2}\backslash\widehat{\mathcal{K}}^2 = \{\binom{1}{0}\},$$
$$\widehat{\mathcal{K}}_{s_3}\backslash\widehat{\mathcal{K}}^2 = \{\binom{1}{4}, \binom{1}{5}\}, \qquad \widehat{\mathcal{K}}_{s_4}\backslash\widehat{\mathcal{K}}^2 = \{\binom{2}{0}, \binom{2}{1}, \ldots, \binom{2}{5}\}.$$

Die Packoperationen lauten:

$$\mathbf{y} = F^3_{m,1}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \widehat{\boldsymbol{\kappa}}^P) \quad : \quad y[b] = \begin{cases} x_1[b], & b = 0 \\ x_2[b], & 1 \leq b \leq 2 \\ x_3[b], & b = 3 \end{cases}$$

$$\mathbf{y} = F^2_{m,1}(\mathbf{x}_1, \mathbf{x}_2, \widehat{\boldsymbol{\kappa}}^P) \quad : \quad y[b] = \begin{cases} x_1[b], & b = 0, \\ x_2[b], & 1 \leq b \leq 3 \end{cases}$$

$$\mathbf{y} = F^2_{m,4}(\mathbf{x}_1, \mathbf{x}_2, \widehat{\boldsymbol{\kappa}}^P) \quad : \quad y[b] = \begin{cases} x_1[b], & 0 \leq b \leq 3 - \widehat{\kappa}^P_2, \\ x_2[b], & 4 - \widehat{\kappa}^P_2 \leq b \leq 3 \end{cases}$$

In diesem Abschnitt wurde die Entstehung von Packoperationen aufgrund der Teilwortparallelisierung von Zuweisung s mit mehreren Teilzuweisungen beschrieben. Durch die Teilwortparallelisierung können zusätzliche Zuweisungen v_{s_t} entstehen und können Packoperationen in der teilwortparallelen Zuweisung s notwendig sein. Die Packoperationen besitzen bis zu $\min(B, T_s)$ Eingabe-DvBs und können von der Iteration $\widehat{\boldsymbol{\kappa}}^P$ abhängig sein.

6.4 Transformation der internen Zuweisungen

Die Transformation einer internen Zuweisung kann nun durch die Kombination der in den vorherigen beiden Abschnitten vorgestellten Verfahren durchgeführt werden. Zuerst soll ein allgemeiner Ansatz vorgestellt werden, bevor in Abschnitt 6.4.2 die Transformation für Propagierungszuweisungen präsentiert wird.

6.4.1 Allgemeiner Ansatz

Eine interne Zuweisung mit T_s Teilzuweisungen sieht wie folgt aus:

$$s: \quad y[\mathbf{i}] = \begin{cases} F_{s_1}(\ldots), & \mathbf{i} \in \mathcal{I}_{s_1}, \\ \quad\vdots & \\ F_{s_t}(\ldots, y_{s'}[\mathbf{i} - \mathbf{d}^{k_{s'}}_{s's_t}], \ldots), & \mathbf{i} \in \mathcal{I}_{s_t}, \\ \quad\vdots & \\ F_{s_{T_s}}(\ldots), & \mathbf{i} \in \mathcal{I}_{s_{T_s}}. \end{cases}$$

Zuerst führen wir eine Teilwortparallelisierung für jede Teilzuweisung s_t aus. Dadurch erhalten wir die neuen Zuweisungen v_{s_t} mit den neuen linksseitigen Variablen $\mathbf{y}_{v_{s_t}}[\widehat{\boldsymbol{\kappa}}^P]$ als DvBs. Packoperationen aufgrund von Abhängigkeitsvektoren $\mathbf{d}_{s's_t}^{k_{s'}} = \mathbf{d}(e) \in \mathbb{Z}^n$ werden vor der Zuweisung v_{s_t} als Zuweisung p_e hinzugefügt. Die Bestimmung der Packoperation und die Transformation der Teilzuweisungen erfolgt entsprechend der Beschreibung in Abschnitt 6.2.

$$
\begin{aligned}
&&&\vdots\\
p_e:\quad &\mathbf{y}_{p_e}[\widehat{\boldsymbol{\kappa}}^P] = F_p(\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_1^P], \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_2^P]), &&\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t}\\
&&&\vdots\\
v_{s_1}:\quad &\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P] = \boldsymbol{F}_{s_1}(\ldots), &&\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1},\\
&&&\vdots\\
v_{s_t}:\quad &\mathbf{y}_{v_{s_t}}[\widehat{\boldsymbol{\kappa}}^P] = \boldsymbol{F}_{s_t}(\ldots, \mathbf{y}_{p_e}[\widehat{\boldsymbol{\kappa}}^P], \ldots), &&\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t},\\
&&&\vdots\\
v_{s_{T_s}}:\quad &\mathbf{y}_{v_{s_{T_s}}}[\widehat{\boldsymbol{\kappa}}^P] = \boldsymbol{F}s_{T_s}(\ldots), &&\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_{T_s}},
\end{aligned}
$$

wobei die Teilvektoren $\widehat{\mathbf{d}}_1^P$ und $\widehat{\mathbf{d}}_2^P$ durch die Partitionierung des Abhängigkeitsvektors $\mathbf{d}_{s's_t}^{k_{s'}} = \mathbf{d}(e)$, siehe Abschnitt 4.2.2, entstanden.

Anschließend wird mit Hilfe der teilwortparallelen Teilzuweisungen v_{s_t} entsprechend Abschnitt 6.3 die Zuweisung s teilwortparallelisiert:

$$
s:\quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} \vdots \\ F_{m,\underline{\mathcal{C}}_{T_s}^p}(\mathbf{y}_{v_{s_t}}[\widehat{\boldsymbol{\kappa}}^P], s_t \in \underline{\mathcal{C}}_{T_s}^p, \widehat{\boldsymbol{\kappa}}^P), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{\underline{\mathcal{C}}_{T_s}^p}^p \setminus \widehat{\mathcal{K}}^{p+1}, \\ \vdots \end{cases}
$$

Beispiel 6.9 (TWP der internen Zuweisung des FIR-Filters)
Das FIR-Filter entsprechend der Beschreibung in Algorithmus 3.1 besitzt eine interne Zuweisung s_3:

$$
s_3:\quad y_y\begin{bmatrix} i \\ j \end{bmatrix} = \begin{cases} y_a\begin{bmatrix} i \\ j \end{bmatrix} \cdot y_x\begin{bmatrix} i \\ j \end{bmatrix}, & \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{3_1} \\ y_y\begin{bmatrix} i \\ j-1 \end{bmatrix} + y_a\begin{bmatrix} i \\ j \end{bmatrix} \cdot y_x\begin{bmatrix} i \\ j \end{bmatrix}, & \begin{bmatrix} i \\ j \end{bmatrix} \in \mathcal{I}_{3_2} \end{cases}
$$

Eine Partitionierung ist nur mit $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ möglich. Bei einer Partitionierung mit $\boldsymbol{\Theta}^P = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ erhalten wir eine Datenabhängigkeit innerhalb eines DvBs, d. h. die Elemente einer Partition $\widehat{\boldsymbol{\kappa}}^P$ könnten nicht gleichzeitig verarbeitet werden. Eine gleichzeitige Verarbeitung der Daten ist aber eine notwendige Bedingung für eine teilwortparallele Verarbeitung eines DvBs.

Die Transformation der Zuweisung s_3 führt zu folgender TWP-Zuweisung:

$$
s_3:\quad \mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P] \cdot \mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^P], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{3_1}, \\ \mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)] + \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P] \cdot \mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^P], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{3_2}. \end{cases}
$$

Der Abhängigkeitsvektor $\mathbf{d} = \binom{0}{1}$ *zerfällt nicht in zwei partitionierte Abhängigkeitsvektoren, weshalb keine Packoperation* F_p *benötigt wird. Die partitionierten Iterationsräume* $\widehat{\mathcal{K}}_{3_1}$ *und* $\widehat{\mathcal{K}}_{3_2}$ *sind nicht überlappend. Aus diesem Grund konnten die zusätzlichen Zuweisungen* v_{3_1} *und* v_{3_2} *aus der transformierten Algorithmenbeschreibung eliminiert werden.*

Die Multiplikationen und die Addition im transformierten Algorithmus sind vierfach parallele Multiplikationen oder Additionen.

6.4.2 Interne Propagierungszuweisungen

Eine interne Propagierungszuweisung

$$s: \qquad y_s[\mathbf{i}] = \begin{cases} y_{s'}[\mathbf{i}], & \mathbf{i} \in \mathcal{I}_{s_1}, \\ y_s[\mathbf{i} - \mathbf{d}^1_{s\,s_2}], & \mathbf{i} \in \mathcal{I}_{s_2}, \\ \quad\vdots & \quad\vdots \\ y_s[\mathbf{i} - \mathbf{d}^1_{s\,s_{T_s}}], & \mathbf{i} \in \mathcal{I}_{s_{T_s}}. \end{cases}$$

welche **eine** Interndatenteilzuweisung s_1 und eine oder mehrere Propagierungsteilzuweisungen $s_2, s_3, \ldots s_{T_s}$ besitzt. Die Propagierungszuweisungen $s_2, s_3, \ldots s_{T_s}$ verteilen die Daten, welche mit der Interndatenteilzuweisung s_1 der Variablen $y_s[\mathbf{i}]$ zugewiesen wurden, in einem konvexen Polyeder $\mathcal{I}_{s_2} \cup \mathcal{I}_{s_3} \cup \cdots \cup \mathcal{I}_{s_{T_s}}$. Die Verteilung der Variablen erfolgt mit den Propagierungsvektoren $\mathbf{d}^1_{s\,s_2}, \mathbf{d}^1_{s\,s_3}, \ldots \mathbf{d}^1_{s\,s_{T_s}}$.

Wurde die Partitionierung so gewählt, dass für ein Propagierungsvektor der Zuweisung gilt: $|\mathbf{d}^1_{s\,s_t}| = \mathbf{\Gamma}^t$, so kann durch die Verwendung der TWP-Operation *verteileTW()* eine effizientere Transformation als im Abschnitt 6.4.1 beschrieben, verwendet werden. Hierbei wird mit der neuen TWP-Operation das zu propagierende Datum $y_{s'}[b, \widehat{\boldsymbol{\kappa}}^P]$ allen Teilwörter des DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ zugewiesen. Anschließend erfolgt die Propagierung der Daten nur noch auf Basis von DvBs und nicht mehr auf Basis von Teilwörtern. Die Propagierungsvektoren für die DvBs sind die äußeren Teilvektoren $\widehat{\mathbf{d}}^P_i$.

Die Teilwortparallelisierung einer solchen Propagierungszuweisung sieht somit wie folgt aus:

- Die erste Teilzuweisung s_1 erhält das zu verteilende Teilwort und verteilt dieses mittels der TWP-Operation *verteileTW()* auf alle Teilwörter des DvBs:

  $$s_1: \qquad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \text{verteileTW}\big(b, \mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P]\big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1}.$$

  Der Iterationsraum $\mathcal{I}_{s_1}$ dieser Teilzuweisung ist immer nur für **ein** Teilwort $y'_s[b, \widehat{\boldsymbol{\kappa}}^P]$ des DvBs $\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P]$ definiert. Die Position b des gültigen Teilworts kann mit der in Abschnitt 6.1.2 gegebenen Methode ermittelt werden.

- Bei den anderen Teilzuweisungen ermitteln wir die äußeren Teilvektoren, welche durch die Partitionierung des Propagierungsvektors $\mathbf{d}^1_{s\,s_t} = \mathbf{d}$ entstehen. Es können maximal zwei äußere Teilvektoren $\widehat{\mathbf{d}}^P_i$ entstehen.

– Gibt es nur einen äußeren Teilvektor $\widehat{\mathbf{d}}_1^P$ oder ist einer der beiden Teilvektoren ein Nullvektor ($\exists!\ \widehat{\mathbf{d}}_i^P = \mathbf{0}$, $i \in \{1,2\}$), so erhalten wir für die Teilzuweisung s_t:

$$s_t: \qquad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_i^P], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t} \cap f^{\widehat{\mathbf{d}}_i^P}(\widehat{\mathcal{K}}_s),$$

$i \in \{1,2\}$ und $\widehat{\mathbf{d}}_i^P \neq \mathbf{0}$.

– Entstehen durch die Partitionierung zwei äußere Teilvektoren $\widehat{\mathbf{d}}_1^P$ und $\widehat{\mathbf{d}}_2^P$, welche keine Nullvektoren sind, so erhalten wir zwei Teilzuweisungen $s_{t,1}$ und $s_{t,2}$ aus der ursprünglichen Teilzuweisung s_t:

$$\begin{aligned} s_{t,1}: &\quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_1^P], \quad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_t} \cap f^{\widehat{\mathbf{d}}_1^P}(\widehat{\mathcal{K}}_s), \\ s_{t,2}: &\quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \widehat{\mathbf{d}}_2^P], \quad \widehat{\boldsymbol{\kappa}}^P \in \left(\widehat{\mathcal{K}}_{s_t} \cap f^{\widehat{\mathbf{d}}_2^P}(\widehat{\mathcal{K}}_s)\right) \setminus \left(\widehat{\mathcal{K}}_{s_t} \cap f^{\widehat{\mathbf{d}}_1^P}(\widehat{\mathcal{K}}_s)\right). \end{aligned}$$

Die Räume $\widehat{\mathcal{K}}_{s_t}$ und $\widehat{\mathcal{K}}_s$ sind die Partitionsräume der Iterationsräume $\mathcal{I}_{s_t}$ und $\mathcal{I}_s$. Auf den Partitionsraum $\widehat{\mathcal{K}}_s$ wird die Funktion $f^{\widehat{\mathbf{d}}_1^P}(\widehat{\boldsymbol{\kappa}}^P) = \mathbf{E} \cdot \widehat{\boldsymbol{\kappa}}^P + \widehat{\mathbf{d}}_1^P$ oder $f^{\widehat{\mathbf{d}}_2^P}(\widehat{\boldsymbol{\kappa}}^P) = \mathbf{E} \cdot \widehat{\boldsymbol{\kappa}}^P + \widehat{\mathbf{d}}_2^P$ angewendet.

Mittels der Schnittmengenoperationen $\widehat{\mathcal{K}}_{s_t} \cap f^{\widehat{\mathbf{d}}_1^P}(\widehat{\mathcal{K}}_s)$ und $\widehat{\mathcal{K}}_{s_t} \cap f^{\widehat{\mathbf{d}}_2^P}(\widehat{\mathcal{K}}_s)$ wird sichergestellt, dass ein gültiges DvB propagiert wird. Existieren zwei äußere Teilvektoren $\widehat{\mathbf{d}}_1^P$ und $\widehat{\mathbf{d}}_2^P$, so muss weiterhin gewährleistet werden, dass jedes DvB nur einmal definiert wird, was durch die Differenzmengenoperation realisiert wurde.

Abhängig davon, welchen der beiden äußeren Teilvektoren man als $\widehat{\mathbf{d}}_1^P$ definiert, entstehen unterschiedliche Realisierungen. Beide Realisierungen sind gültig. In der weiteren Arbeit gehen wir davon aus, dass gilt: $\widehat{\mathbf{d}}_2^P = \widehat{\mathbf{d}}_1^P + \boldsymbol{\Gamma}^t$.

Beispiel 6.10 (Teilwortparallelisierung einer Propagierungszuweisung)
Wir haben eine Propagierungszuweisung s:

$$s: \qquad y_s[\mathbf{i}] = \begin{cases} y_{s'}[\mathbf{i}], & \mathbf{i} \in \mathcal{I}_1, \\ y_s[\mathbf{i} - \binom{1}{0}], & \mathbf{i} \in \mathcal{I}_2, \\ y_s[\mathbf{i} - \binom{1}{1}], & \mathbf{i} \in \mathcal{I}_3, \end{cases}$$

mit der Interndatenteilzuweisung s_1 und den zwei Propagierungszuweisungen s_2 und s_3. Die Iterationsräume der Teilzuweisungen haben die Größe:

$$\mathcal{I}_1: \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \mathcal{I}_2: \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 1 \\ 0 \\ 12 \\ 5 \end{pmatrix}, \quad \mathcal{I}_3: \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ -1 & 1 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 0 \\ 1 \\ 0 \\ 5 \end{pmatrix}. \quad (6.74)$$

Der Iterationsraum $\mathcal{I}$ der Zuweisung s ist somit:

$$\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \mathcal{I}_3: \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ -1 & 1 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 0 \\ 0 \\ 17 \\ 11 \end{pmatrix}.$$

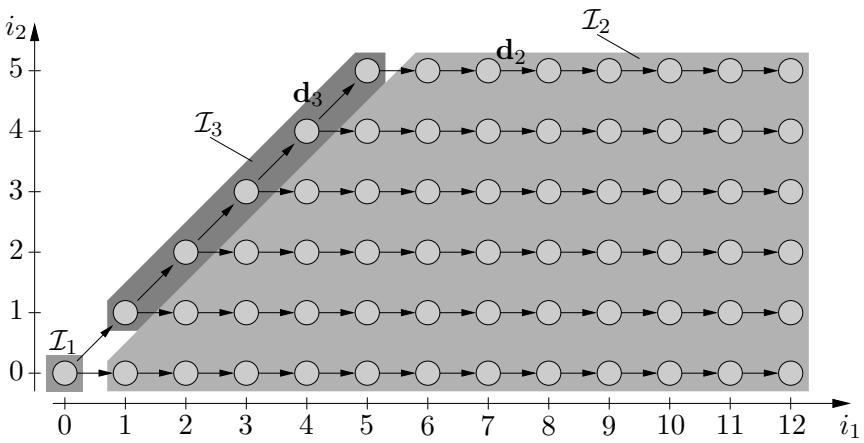


Abbildung 6.9: Iterationsraum der Propagierungszuweisung s

Abbildung 6.9 zeigt den Iterationsraum der Zuweisung s mit den Iterationsräumen der Teilzuweisungen und den Propagierungsvektoren $\mathbf{d}^1_{s\,s_2} = \mathbf{d}_2 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ und $\mathbf{d}^1_{s\,s_3} = \mathbf{d}_3 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$.

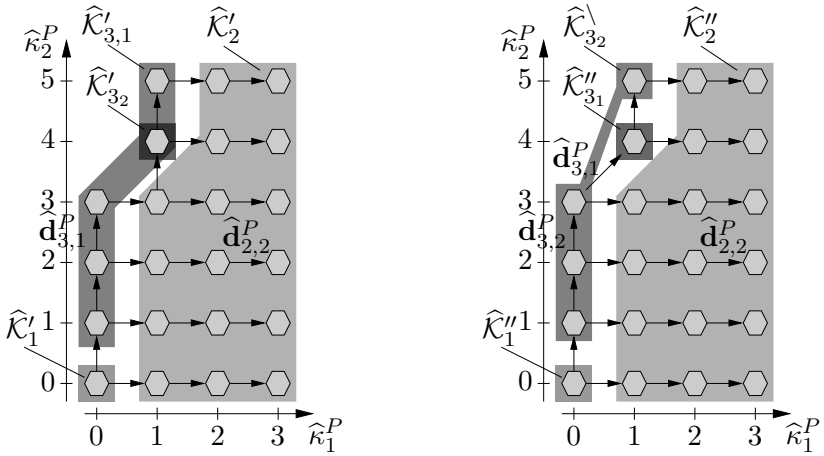
Bei einer Partitionierung mit $\mathbf{\Theta}^P = \left(\begin{smallmatrix}4&0\\0&1\end{smallmatrix}\right)$ kann die effiziente Teilwortparallelisierung für Propagierungsvektoren genutzt werden.

Die Teilzuweisung s_1 von (6.74) erhält die TWP-Operation verteileTW(). *Der Propagierungsvektor $\mathbf{d}_2 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ der Teilzuweisung s_2 zerfällt in zwei partitionierte Abhängigkeitsvektoren mit den äußeren Teilvektoren $\widehat{\mathbf{d}}^P_{2,1} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ und $\widehat{\mathbf{d}}^P_{2,2} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$. Da der Teilvektor $\widehat{\mathbf{d}}^P_{2,1}$ ein Nullvektor ist, erhalten wir eine teilwortparallele Teilzuweisung. Diese besitzt den neuen Propagierungsvektor $\widehat{\mathbf{d}}^P_{2,2} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$.*

Für den Propagierungsvektor $\mathbf{d}_3 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ der Teilzuweisung s_3 erhalten wir die beiden äußeren Teilvektoren $\widehat{\mathbf{d}}^P_{3,1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ und $\widehat{\mathbf{d}}^P_{3,2} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$. Da der Iterationsraum $\widehat{\mathcal{K}}'_{3_1} = \widehat{\mathcal{K}}_3 \cap f^{\widehat{\mathbf{d}}^P_{3,1}}(\widehat{\mathcal{K}})$ den Iterationsraum $\widehat{\mathcal{K}}'_{3_2} = \widehat{\mathcal{K}}_3 \cap f^{\widehat{\mathbf{d}}^P_{3,2}}(\widehat{\mathcal{K}})$ enthält und somit gilt: $\widehat{\mathcal{K}}'_{3_2} \backslash \widehat{\mathcal{K}}'_{3_1} = \emptyset$, erhalten wir nur eine teilwortparallele Teilzuweisung. Diese hat den neuen Propagierungsvektor $\widehat{\mathbf{d}}^P_{3,1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$.

In Algorithmus 6.1 ist die teilwortparallelisierte Zuweisung s als Version 1 angegeben. Abbildung 6.10(a) zeigt den zugehörigen Partitionsraum mit den Teilräumen und den neuen Propagierungsvektoren.

Vertauscht man die beiden äußeren Teilvektoren, die sich aus der Partitionierung des Propagierungsvektors $\mathbf{d}_3 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ ergeben, zu $\widehat{\mathbf{d}}^P_{3,1} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ und $\widehat{\mathbf{d}}^P_{3,2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, so ist der Iterationsraum $\widehat{\mathcal{K}}''_{3_2} = \widehat{\mathcal{K}}_3 \cap f^{\widehat{\mathbf{d}}^P_{3,2}}(\widehat{\mathcal{K}})$ nicht mehr Teilraum des Iterationsraums $\widehat{\mathcal{K}}''_{3_1} = \widehat{\mathcal{K}}_3 \cap f^{\widehat{\mathbf{d}}^P_{3,1}}(\widehat{\mathcal{K}})$, $\widehat{\mathcal{K}}^{\backslash}_{3_2} = \widehat{\mathcal{K}}''_{3_2} \backslash \widehat{\mathcal{K}}''_{3_1} \neq \emptyset$. Somit entstehen zwei teilwortparallele Teilzuweisungen aus der Ausgangsteilzuweisung s_3. Diese Version der teilwortparallelen Propagierungszuweisung ist als Version 2 in Algorithmus 6.1 angegeben. Der Partitionsraum dieser Version mit den Teilräumen und den Abhängigkeitsvektoren wurde in Abbildung 6.10(b) dargestellt.



(a) Version 1 (b) Version 2

Abbildung 6.10: Partitionsraum der Propagierungszuweisung s

Algorithmus 6.1 Teilwortparallelisierung einer Propagierungszuweisung

Version 1 mit $\widehat{\mathbf{d}}^P_{3,1} = \binom{0}{1}$

$$\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} \text{verteileTW}\left(\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P\right), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}'_1, \\ \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \binom{1}{0}], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}'_2, \\ \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \binom{0}{1}], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}'_{3_1}. \end{cases}$$

Version 2 mit $\widehat{\mathbf{d}}^P_{3,1} = \binom{1}{1}$

$$\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} \text{verteileTW}\left(\mathbf{y}_{s'}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P\right), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}''_1, \\ \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \binom{1}{0}], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}''_2, \\ \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \binom{1}{1}], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}''_{3_1}, \\ \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P - \binom{0}{1}], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}^{\setminus}_{3_2}. \end{cases}$$

6.4.3 Zusammenfassung

In diesem Abschnitt wurde die Transformation der internen Zuweisungen vorgestellt. Hierfür wurde in Abschnitt 6.4.1 ein allgemeiner Ansatz vorgestellt, der im Wesentlichen die Methoden aus den Abschnitten 6.2 und 6.3 verwendet. Durch diese Transformation können aufgrund mehrerer Teilzuweisungen in einer Zuweisung und des Zerfalls der Abhängigkeitsvektoren neue Zuweisungen und zusätzliche Packoperationen entstehen.

Ist die interne Zuweisung eine Propagierungszuweisung und existiert ein Propagierungsvektor $\mathbf{d}^1_{s\,s_t}$ für den gilt: $|\mathbf{d}^1_{s\,s_t}| = \mathbf{\Gamma}^t$, so kann die Propagierungszuweisung unter Zuhilfenahme der TWP-Operationen *verteileTW()* transformiert werden, wie in Abschnitt 6.4.2 gezeigt wurde. Diese Transformation der Propagierungszuweisung ist effizienter als die Transformation mittels des allgemeinen Ansatzes.

6.5 Transformation der Ein- und Ausgabezuweisungen

Je nach Zuweisung, Partitionierungsversion und Speicherarchitektur gibt es verschiedene Arten des Lesens oder Schreibens von Daten. Diese unterschiedlichen Arten sollen ermittelt und für diese die TWP-Operationen bestimmt werden.

Ausgangspunkt für unsere Untersuchung ist eine Eingabezuweisung der Form:

$$s: \qquad y_s[\mathbf{i}] = y_m[\mathbf{i}^M] = y_m[m(\mathbf{i})], \qquad \mathbf{i} \in \mathcal{I}_s, \tag{6.75}$$

oder eine Ausgabezuweisung der Form:

$$m: \qquad y_m[\mathbf{i}^M] = y_m[m(\mathbf{i})] = y_s[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_m, \tag{6.76}$$

mit $\mathbf{i}^M = m(\mathbf{i}) = \mathbf{M}\mathbf{i} + \mathbf{m}$.

Beide Zuweisungen besitzen jeweils nur eine Teilzuweisung. Ein- und Ausgabezuweisungen mit mehreren Teilzuweisungen lassen sich durch Hinzufügen von temporären Variablen zum Algorithmus zu Interndatenzuweisungen und Ein- oder Ausgabezuweisungen umwandeln, bei denen die Ein- oder Ausgabezuweisungen jeweils nur noch eine Teilzuweisung besitzen.

Eine Sonderstellung nimmt die Eingabe- und Propagierungszuweisung ein, welche aufgrund ihrer besonderen Eigenschaften effizienter teilwortparallelisiert werden kann. In Abschnitt 6.5.4 wird diese Transformation vorgestellt.

Die Daten $y_m[.]$ im Speicher können sowohl einzeln, als Teilwörter, oder gemeinsam, als DvBs, gelesen oder geschrieben werden. Hierbei muss die Anordnung der Daten im Speicher beachtet werden, denn aufgrund der linearen Anordnung der Daten im Speicher, siehe Abschnitt 2.4, kann der Lese- oder Schreibaufwand sehr unterschiedlich sein.

Die internen Daten $y_s[.]$ werden hingegen immer auf Basis von DvBs verarbeitet. Aus diesem Grund wird bei der Transformation der Ein- und Ausgabezuweisungen zwischen

Ein- und Ausgabedaten in Form von Teilwörtern oder in Form von DvBs unterschieden. Diese Transformationen werden in den Abschnitten 6.5.2 und 6.5.3 vorgestellt. Aktuelle Architekturen erlauben entweder nur das Lesen und Schreiben von Speicherdaten in Form von DvBs oder das Lesen und Schreiben ist sowohl in Form von Teilwörtern als auch in Form von DvBs möglich.

Vor der Beschreibung der Transformation der Ein- und Ausgabezuweisungen muss im nachfolgenden Abschnitt das Thema Speicherzugriffe behandelt werden. Durch die Partitionierung des Indexraums $\mathcal{I}^M$ der globalen Variablen können bei einem Speicherzugriff auf Basis DvBs nicht definierte Teilwörter gelesen oder geschrieben werden. Diese ungültigen Speicherzugriffe müssen vermieden werden. Eine Methode dafür wird im nächsten Abschnitt vorgestellt.

6.5.1 Gültige Speicherzugriffe

Ein gültiger Speicherzugriff ist ein Speicherzugriff, bei dem jedes zu lesende Teilwort $y_m[b^M, \widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M] = y_m[\mathbf{i}^M]$ eines DvBs $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ definiert ist. Wird der Indexraum $\mathcal{I}^M$ einer globalen Variable $y_m[\mathbf{i}^M]$, $\mathbf{i}^M \in \mathcal{I}^M$ partitioniert, entstehen Partitionen $\widehat{\boldsymbol{\kappa}}^M \in \widehat{\mathcal{K}}^M$, in denen Teilwörter $y_m[b^M, \widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M] = y_m[\mathbf{i}^M]$, $\mathbf{i}^M \notin \mathcal{I}^M$ eines DvBs $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$, $\widehat{\boldsymbol{\kappa}}^M \in \widehat{\mathcal{K}}^M$ nicht definiert sind. Wird auf diese Teilwörter mit einem DvB-Speicherzugriff zugegriffen, erhalten wir einen ungültigen Speicherzugriff. Existieren die Speicherplätze, auf die mit den ungültigen Speicherzugriffen zugegriffen wird, und wird beim Schreiben von Daten sichergestellt, dass die nicht definierten Teilwörter in den Speicher-DvBs erhalten bleiben, so kann mit ungültigen Speicherzugriffen gearbeitet werden.

Ungültige Speicherzugriffe sind im Allgemeinen sehr risikoanfällig. Deshalb verbieten moderne Betriebssysteme solche Speicherzugriffe.

Im Allgemeinen führt jede Partitionierung des Indexraums $\mathcal{I}^M$ zu ungültigen Speicherzugriffen, wenn anschließend auf Basis von DvBs auf den Speicher zugegriffen wird. Um das zu verhindern, kann nur der Indexraum $\mathcal{I}^M$ oder der Partitionsraum $\widehat{\mathcal{K}}^M$ für die Speicherdaten $y_m[\mathbf{i}^M]$, $\mathbf{i}^M \in \mathcal{I}^M$ oder $\mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M]$, $\widehat{\boldsymbol{\kappa}}^M \in \widehat{\mathcal{K}}^M$ erweitert werden, sodass die Speicherzugriffe immer gültig sind.

Bei einer Organisation des Speichers auf Basis von DvBs und der Möglichkeit unausgerichteter Speicherzugriffe ist dem Partitionsraum $\widehat{\mathcal{K}}^M$ in der Raumrichtung in der Daten zu DvBs zusammengepackt sind ein zusätzliches DvB voran und ein zusätzliches DvB hintenan zu stellen. Für den erweiterten Partitionsraum $\widehat{\mathcal{K}}^{M,\boxdot}$ erhalten wir somit:

$$\widehat{\mathcal{K}}^{M,\boxdot} = \widehat{\mathcal{K}}^M \cup f^{\boldsymbol{\Gamma}^{M,t}}(\widehat{\mathcal{K}}^M) \cup f^{-\boldsymbol{\Gamma}^{M,t}}(\widehat{\mathcal{K}}^M). \tag{6.77}$$

Die auf die Elemente des Partitionsraums $\widehat{\mathcal{K}}^M$ angewendeten Funktionen lauten:

$$f^{\boldsymbol{\Gamma}^{M,t}}(\widehat{\boldsymbol{\kappa}}^M) = \mathbf{E} \cdot \widehat{\boldsymbol{\kappa}}^M + \boldsymbol{\Gamma}^{M,t} \quad \text{und} \quad f^{-\boldsymbol{\Gamma}^{M,t}}(\widehat{\boldsymbol{\kappa}}^M) = \mathbf{E} \cdot \widehat{\boldsymbol{\kappa}}^M - \boldsymbol{\Gamma}^{M,t}, \qquad \widehat{\boldsymbol{\kappa}}^M \in \widehat{\mathcal{K}}^M.$$

Sind nur ausgerichtete Speicherzugriffe erlaubt, muss der Partitionsraum nicht erweitert werden.

Ist das Speichern auf Basis von Teilwörtern organisiert und sind unausgerichtete Speicherzugriffe möglich, müssen dem Indexraum $\mathcal{I}^M$ in der Raumrichtung, in der die Daten zu DvBs zusammengepackt sind, jeweils $B-1$ Teilwörter voran und hintenan gestellt werden. Der erweiterte Indexraum $\mathcal{I}^{M,\boxdot}$ ergibt sich somit durch:

$$\mathcal{I}^{M,\boxdot} = \widehat{\mathcal{I}}^M \cup f^{(B-1)\mathbf{\Gamma}^{M,t}}(\mathcal{I}^M) \cup f^{-(B-1)\mathbf{\Gamma}^{M,t}}(\mathcal{I}^M). \tag{6.78}$$

Die auf die Elemente des Indexraums $\mathcal{I}^M$ angewendeten Funktionen lauten:

$$f^{(B-1)\mathbf{\Gamma}^{M,t}}(\mathbf{i}^M) = \mathbf{E}\cdot\mathbf{i}^M + (B-1)\cdot\mathbf{\Gamma}^{M,t} \quad \text{und}$$
$$f^{-(B-1)\mathbf{\Gamma}^{M,t}}(\mathbf{i}^M) = \mathbf{E}\cdot\mathbf{i}^M - (B-1)\cdot\mathbf{\Gamma}^{M,t}$$

mit $\mathbf{i}^M \in \mathcal{I}^M$.

Erlaubt die Speicherarchitektur nur ausgerichtete Speicherzugriffe, so ist der Indexraum $\mathcal{I}^M$ nur so zu erweitern, dass alle Teilwörter $y_m[b^M, \widehat{\boldsymbol{\kappa}}^M, 0]$ der DvBs $\mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, 0]$ gültig sind:

$$\mathcal{I}^{M,\boxdot} = \left\{\mathbf{i} = \mathbf{\Theta}^P\cdot\widehat{\boldsymbol{\kappa}}^P + \mathbf{\Gamma}^t\cdot b \mid \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}^M, b\in\mathcal{K}^b\right\}.$$

Durch diese Erweiterung sind nun alle Speicherzugriffe auf Basis von DvBs möglich, ohne dass ein Teilwort des zu lesenden oder zu schreibenden DvBs außerhalb des erweiterten Definitionsbereichs $\mathcal{I}^{M,\boxdot}$ liegt.

Bei der Erweiterung des Definitionsbereichs ist darauf zu achten, dass andere Algorithmen einer Anwendung, welche ebenfalls auf diese Daten zugreifen, weiterhin nur auf die ursprünglichen Daten zugreifen.

Beispiel 6.11 (Gültige Speicherzugriffe bei Matrixmultiplikation)
Die Eingabedaten $a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right]$ bei der Matrixmultiplikation $\mathbf{C} = \mathbf{A}\cdot\mathbf{B}$ sind in einem Speicher auf Basis von Teilwörtern und DvBs organisiert. Es sind sowohl ausgerichtete als auch unausgerichtete Speicherzugriffe möglich. Der Indexraum dieser Eingabedaten ist gegeben mit: $\mathcal{I}_a^M = \left\{\mathbf{i}^M = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \mid 0 \le i \le 12 \wedge 0 \le j \le 5\right\}$.

Für sichere Speicherzugriffe erweitern wir den Definitionsbereich um zweimal $(B-1)$ Teilwörter in der Raumrichtung, in der die Daten zu DvBs zusammengepackt sind. Diese Raumrichtung wird durch die Allokationsmatrix $\mathbf{\Gamma}_a^M = \begin{pmatrix} 1 & 0 \end{pmatrix}$ angegeben. Bei $B=4$ vergrößert sich damit der Indexraum zu: $\mathcal{I}_a^{M,\boxdot} = \left\{\mathbf{i}^M = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \mid -3 \le i \le 15 \wedge 0 \le j \le 5\right\}$.

Für die weitere Arbeit gehen wir davon aus, dass alle Speicherzugriffe gültige Speicherzugriffe sind. Die Erweiterung des Definitionsbereichs der globalen Variablen wird stillschweigend vorausgesetzt.

6.5.2 Ein- und Ausgabedaten als Teilwörter

Ist der Speicher auf Basis von Teilwörtern organisiert, kann auf die Ein- und Ausgabedaten einzeln zugegriffen werden. In diesem Fall muss nur der Iterationsraum $\mathcal{I}$ mit

$\mathbf{i} = \boldsymbol{\Theta}^P \widehat{\boldsymbol{\kappa}}^P + \boldsymbol{\Gamma}^t \cdot b$ partitioniert werden. Für die Indexfunktion $m(\mathbf{i})$ gilt somit:

$$\mathbf{i}^M = \mathbf{M}\mathbf{i} + \mathbf{m}, \tag{6.79}$$

$$\mathbf{i}^M = \mathbf{M}\left(\boldsymbol{\Theta}^P \cdot \widehat{\boldsymbol{\kappa}}^P + \boldsymbol{\Gamma}^t \cdot b\right) + \mathbf{m} \tag{6.80}$$

$$\mathbf{i}^M = \underbrace{\mathbf{M} \cdot \boldsymbol{\Theta}^P}_{\mathbf{M}^\kappa} \cdot \widehat{\boldsymbol{\kappa}}^P + \underbrace{\mathbf{M} \cdot \boldsymbol{\Gamma}^t \cdot b + \mathbf{m}}_{\mathbf{m}^\kappa(b)}. \tag{6.81}$$

Die Indexfunktion entspricht nun nicht mehr der allgemeinen Form: $\mathbf{i}^M = \mathbf{B} \cdot \mathbf{i} + \mathbf{b}$, wie sie in Definition 3.1 für UItA angegeben wurde. Da diese Form der Indexfunktion nur bei Ein- oder Ausgabedaten auftreten kann, soll sie trotzdem verwendet werden. Anderenfalls müsste die Algorithmenklasse so stark eingeschränkt werden, sodass relevante Algorithmen kaum noch transformiert werden könnten.

Transformation der Eingabezuweisung

Mit der Operation *leseTW()* werden die Teilwörter einzeln eingelesen, dabei wird das gelesene Teilwort an die erste Position des Berechnungs-DvBs $\mathbf{y}_{1,b}[\widehat{\boldsymbol{\kappa}}^P]$ geschrieben:

$$t_{1,b}: \qquad \mathbf{y}_{1,b}[\widehat{\boldsymbol{\kappa}}^P] = \text{leseTW}\big(y_m[\mathbf{M}^\kappa \widehat{\boldsymbol{\kappa}}^P + \mathbf{m}^\kappa(b)]\big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s, \tag{6.82}$$

mit $b_{\min} \leq b \leq b_{\max}$.

Anschließend müssen die gültigen Teilwörter in den DvBs $\mathbf{y}_{1,b}[\widehat{\boldsymbol{\kappa}}^P]$ zu dem gewünschten Berechnungs-DvB $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P]$ zusammengestellt werden. Hierfür sollen $b_{\max} - b_{\min}$ F_p-Operationen, siehe (6.38), genutzt werden. Hierbei wird mit jeder Operation ein Teilwort $y_{1,b}[0, \widehat{\boldsymbol{\kappa}}^P]$ von rechts in das Ergebnis-DvB eingeschoben. Mit der letzten F_p-Operation und der Verschiebung um $B - b_{\max}$ erreichen wir zusätzlich, dass die Teilwörter richtig im DvB positioniert sind. Bei den anderen F_p-Operationen werden die Daten jeweils um ein Teilwort nach rechts verschoben:

$$t_{2,b}: \qquad \mathbf{y}_{2,b}[\widehat{\boldsymbol{\kappa}}^P] = F_p\big(\mathbf{y}_{2,b-1}[\widehat{\boldsymbol{\kappa}}^P], y_{1,b}[\widehat{\boldsymbol{\kappa}}^P], 1\big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s, \tag{6.83}$$

$$s: \qquad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = F_p\big(\mathbf{y}_{2,b_{\max}-1}[\widehat{\boldsymbol{\kappa}}^P], y_{1,b_{\max}}[\widehat{\boldsymbol{\kappa}}^P], B - b_{\max}\big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s, \tag{6.84}$$

mit $b_{\min} \leq b < b_{\max}$, wobei $\mathbf{y}_{2,b_{\min}-1}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{0}$ ein Null-DvB ist.

Beispiel 6.12 (Eingabezuweisung des FIR-Filters)
Wir betrachten die Eingabeteilzuweisung $s_{1,1}$ aus der Eingabezuweisung s_1 des FIR-Filters aus Beispiel 3.1 von Seite 23:

$$s_{1,1}: \qquad y_a[\mathbf{i}] = a[\begin{pmatrix}0 & 1\end{pmatrix}\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_{1_1}.$$

Der Iterationsraum $\mathcal{I} \supset \mathcal{I}_{1_1}$ des Algorithmus soll mit $\boldsymbol{\Theta}_1^P = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ $(\boldsymbol{\Gamma}_1 = \begin{pmatrix}1 & 0\end{pmatrix})$ oder $\boldsymbol{\Theta}_2^P = \left(\begin{smallmatrix}1 & 0\\ 0 & 4\end{smallmatrix}\right)$ $(\boldsymbol{\Gamma}_2 = \begin{pmatrix}0 & 1\end{pmatrix})$ partitioniert werden.

Wird die Partitionsmatrix $\mathbf{\Theta}_1^P = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ *gewählt, so besitzt jedes DvB* $\mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P]$ *genau ein Teilwort* $y_a[0, \widehat{\boldsymbol{\kappa}}^P] = a[\begin{pmatrix}0 & 1\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P]$*, das eingelesen werden muss. Dieses Teilwort wird mit der Zuweisung*

$$s_{1,1}: \qquad \mathbf{y}_{a,1}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{leseTW}\big(a[\begin{pmatrix}0 & 1\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P]\big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1_1}.$$

gelesen. Die Vervollständigung des Berechnungs-DvBs $\mathbf{y}_{a,1}[\widehat{\boldsymbol{\kappa}}^P]$ *mit den jeweiligen Teilwörtern* $y_a[b, \widehat{\boldsymbol{\kappa}}^P]$, $1 \leq b < B$*, siehe Teilzuweisung* $s_{1,2}$ *des Algorithmus, zum DvB* $\mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P]$ *erfolgt mit Hilfe der Packoperationen aus Abschnitt 6.3.*

Erfolgt eine Partitionierung mit $\mathbf{\Theta}_2^P = \left(\begin{smallmatrix}1 & 0\\ 0 & 4\end{smallmatrix}\right)$*, so sind alle Teilwörter* $y_a[b, \widehat{\boldsymbol{\kappa}}^P]$ *des DvBs* $\mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P]$ *Eingabedaten. Diese werde nacheinander eingelesen und anschließend zum Berechnungs-DvB* $\mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P]$ *zusammengepackt:*

$$\begin{aligned}
t_{1,0}: &\quad \mathbf{y}_{a,1,0}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{leseTW}\big(a[\begin{pmatrix}0 & 4\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P]\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1},\\
t_{1,1}: &\quad \mathbf{y}_{a,1,1}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{leseTW}\big(a[\begin{pmatrix}0 & 4\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P + 1]\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1},\\
t_{1,2}: &\quad \mathbf{y}_{a,1,2}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{leseTW}\big(a[\begin{pmatrix}0 & 4\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P + 2]\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1},\\
t_{1,3}: &\quad \mathbf{y}_{a,1,3}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{leseTW}\big(a[\begin{pmatrix}0 & 4\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P + 3]\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1},
\end{aligned}$$

$$\begin{aligned}
t_{2,0}: &\quad \mathbf{y}_{a,2,0}[\widehat{\boldsymbol{\kappa}}^P] = F_p\big(\mathbf{0}, \mathbf{y}_{a,1,0}[\widehat{\boldsymbol{\kappa}}^P], 1\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1},\\
t_{2,1}: &\quad \mathbf{y}_{a,2,1}[\widehat{\boldsymbol{\kappa}}^P] = F_p\big(\mathbf{y}_{a,2,0}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{a,1,1}[\widehat{\boldsymbol{\kappa}}^P], 1\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1},\\
t_{2,2}: &\quad \mathbf{y}_{a,2,2}[\widehat{\boldsymbol{\kappa}}^P] = F_p\big(\mathbf{y}_{a,2,1}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{a,1,2}[\widehat{\boldsymbol{\kappa}}^P], 1\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1},\\
s_{1,1}: &\quad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = F_p\big(\mathbf{y}_{a,2,2}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{a,1,3}[\widehat{\boldsymbol{\kappa}}^P], 1\big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1,1}.
\end{aligned}$$

Transformation der Ausgabezuweisung

Formal erfolgt das Schreiben der Ausgabedaten mit der Zuweisung:

$$m_b: \qquad y_m[\mathbf{M}^\kappa \widehat{\boldsymbol{\kappa}}^P + \mathbf{m}^\kappa(b)] = \text{schreibeTW}\big(b, \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]\big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s. \quad (6.85)$$

Für jedes Teilwort $y_s[b, \widehat{\boldsymbol{\kappa}}^P]$, das in den Speicher geschrieben werden soll, müsste eine Zuweisung m_b erzeugt werden. Das hätte aber zur Folge, dass mehrere Zuweisungen m_b, $b_{\min} \leq b \leq b_{\max}$, mit der gleicher linksseitigen Variable aber mit jeweils unterschiedlichen Indexfunktionen entstehen würden. Da dies laut der Algorithmendefinition 3.1, Seite 21 f. nicht zulässig ist, darf nur ein Teilwort je DvB in den Speicher geschrieben werden. Es muss somit gelten: $b_{\min} = b_{\max} = b$. Eine allgemeine Methode zur Ermittlung, ob alle DvBs nur ein gültige Teilwort besitzen, wurde in Abschnitt 6.1.2 ab Seite 88 vorgestellt.

Das Schreiben von mehreren Teilwörtern eines DvBs in den Speicher kann nur auf Basis von DvBs erfolgen. Die Transformation dafür wird im nachfolgenden Abschnitt vorgestellt.

Beispiel 6.13 (Ausgabezuweisung des FIR-Filters)
Die Ausgabezuweisung des FIR-Filters lautet:

$$s_4: \qquad y[\begin{pmatrix}1 & 0\end{pmatrix}\mathbf{i}] = y_y[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_{4_1}.$$

Bei einer Partitionierung des Iterationsraums $\mathcal{I} \supset \mathcal{I}_{4_1}$ mit $\mathbf{\Theta}_1^P = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ ist eine Transformation der Ausgabezuweisung nicht möglich, denn jedes Teilwort $y_y[b, \widehat{\boldsymbol{\kappa}}^P]$, $0 \leq b < B$ des DvBs $\mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^P]$ müsste in den Speicher geschrieben werden.

Die Partitionierung mit $\mathbf{\Theta}_2^P = \left(\begin{smallmatrix}1 & 0\\ 0 & 4\end{smallmatrix}\right)$ führt zu Berechnungs-DvBs, von denen ein Teilwort als Ausgabe in den Speicher geschrieben werden muss. Hier ergibt die Transformation der Ausgabezuweisung:

$$s_4: \qquad \mathbf{y}[\begin{pmatrix}1 & 0\end{pmatrix}\widehat{\boldsymbol{\kappa}}^P] = schreibeTW\big(b, \mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^P]\big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{4_1},$$

wobei $b = (J-1) - 4 \cdot \widehat{\kappa}_2^P$ ist.

6.5.3 Ein- und Ausgabedaten als DvBs

Sind die Ein- und Ausgabedaten DvBs, so wird der Indexraum $\mathcal{I}_m$ der Speicherdaten wie folgt partitioniert:

$$\mathbf{i}^M = \mathbf{\Theta}^M \widehat{\boldsymbol{\kappa}}^M + \mathbf{\Gamma}^{M,t} \cdot b^M + \mathbf{\Gamma}^{M,t} \vartheta_0^M. \tag{6.86}$$

Mit der Partitionierung des Iterationsraums $\mathcal{I}$ der Iteration $\mathbf{i} \in \mathcal{I}$ mit

$$\mathbf{i} = \mathbf{\Theta}^P \widehat{\boldsymbol{\kappa}}^P + \mathbf{\Gamma}^t \cdot b \tag{6.87}$$

erhalten wir für die Indexfunktion $m(\mathbf{i})$:

$$\mathbf{i}^M = \mathbf{M}\mathbf{i} + \mathbf{m}, \tag{6.88}$$

$$\mathbf{\Theta}^M \widehat{\boldsymbol{\kappa}}^M + \mathbf{\Gamma}^{M,t} \cdot b^M + \mathbf{\Gamma}^{M,t} \cdot \vartheta_0^M = \mathbf{M}\left(\mathbf{\Theta}^P \widehat{\boldsymbol{\kappa}}^P + \mathbf{\Gamma}^t \cdot b\right) + \mathbf{m}. \tag{6.89}$$

Den Index des Speicherteilwortes $y_M[b^M, \widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ ermitteln wir aus dem Index des Berechnungsteilworts $y[b, \widehat{\boldsymbol{\kappa}}^P]$ und der Verschiebung ϑ_0^M mittels:

$$\widehat{\boldsymbol{\kappa}}^M = m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M) \tag{6.90}$$

$$= \left\lfloor \mathbf{\Theta}^{M,-1} \cdot \left(\mathbf{M} \cdot \mathbf{\Gamma}^t \cdot b + \mathbf{M} \cdot \mathbf{\Theta}^P \cdot \widehat{\boldsymbol{\kappa}}^P + \mathbf{m} - \mathbf{\Gamma}^{M,t} \cdot \vartheta_0^M\right) \right\rfloor, \tag{6.91}$$

$$b^M = m_b(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M) \tag{6.92}$$

$$= \left(\mathbf{\Gamma}^M \left(\mathbf{M} \cdot \mathbf{\Gamma}^t \cdot b + \mathbf{M} \cdot \mathbf{\Theta}^P \cdot \widehat{\boldsymbol{\kappa}}^P + \mathbf{m}\right) - \vartheta_0^M\right) \mod B. \tag{6.93}$$

Prinzipiell ist es nun mittels der Funktionen $m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M)$ und $m_b(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M)$ möglich, die Teilwörter $y[b, \widehat{\boldsymbol{\kappa}}^P]$ eines Berechnungs-DvBs einzeln zu lesen oder zu schreiben.

Um die Teilwortparallelität aber voll auszunutzen, werden nachfolgend zwei Sonderfälle der Datenanordnung, die gleichartig angeordneten Daten und die ausgerichteten Daten, vorgestellt. Beide Datenanordnungen ermöglichen das parallele Lesen und Schreiben von mehreren Teilwörtern. Weiterhin werden Methoden präsentiert, mit denen überprüft werden kann, ob die Daten diese Anordnung besitzen.

Anschließend werden die Methoden zur Transformationen der Ein- und Ausgabezuweisungen vorgestellt.

Gleichartig angeordnete und ausgerichtete Daten

Liegen die Daten, die gelesen oder geschrieben werden sollen, in der gleichen Reihenfolge im Berechnungs- und im Speicher-DvB, so spricht man von gleichartig angeordneten Daten. Die Position b und b^M der Teilwörter im Berechnungs-DvB und Speicher-DvB können dabei unterschiedlich sein. Ist die relative Position eines Datums im Berechnungs- und im Speicher-DvB gleich ($b = b^M$), so spricht man von ausgerichteten Daten. Nachfolgend wird vorgestellt, wie ermittelt werden kann, ob die Daten gleichartig angeordnet oder sogar ausgerichtet sind.

Gleichartig angeordnete Daten Es ist zu ermitteln, ob die durch eine Partitionierung des Iterationsraums $\mathcal{I}$ entstehenden Berechnungs-DvBs $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P]$ die gleichen Teilwörter mit der gleichen Reihenfolge enthalten können, wie die Speicher-DvBs $\mathbf{y}_M[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$, wobei wir von unausgerichteten Speicherzugriffen ($0 \leq \vartheta_0^M < B$) ausgehen.

Satz 6.1 (Gleichartige Anordnung der Teilwörter in den DvBs)
Gilt die Gleichung

$$\boldsymbol{\Gamma}^{M,t} = \mathbf{M} \cdot \boldsymbol{\Gamma}^t, \tag{6.94}$$

so sind die Teilwörter in den Berechnungs- und in den Speicher-DvBs gleichartig angeordnet.

Beweis 6.2
Die Indexfunktion $\mathbf{i}^M = m(\mathbf{i}) = \mathbf{M} \cdot \mathbf{i} + \mathbf{m}$, $m : \mathcal{I} \rightarrow \mathcal{I}_m$ beschreibt die Zuordnung der Iterationen $\mathbf{i}$ des Iterationsraums $\mathcal{I}$ zu den Indizes $\mathbf{i}^M$ des Indexraums $\mathcal{I}_m$. Die Abbildung $m(\mathbf{i})$ ist laut Algorithmendefinition 3.1 von Seite 21 eine bijektive Abbildung. Somit erhalten wir für jede Iteration $\mathbf{i} \in \mathcal{I}$ eineindeutig einen Index $\mathbf{i}^M \in \mathcal{I}_m$.

Wir nehmen an, wir haben zwei Iterationen $\mathbf{i}_1$ und $\mathbf{i}_2$, deren Differenz $\boldsymbol{\Gamma}^t = \mathbf{i}_2 - \mathbf{i}_1$ beträgt. Somit liegen diese beiden Iterationen in der Raumrichtung j ($\gamma_{1,j} = 1$) nebeneinander, in der die Iterationen zu Partitionen, den Berechnungs-DvBs, zusammengefasst werden. Mit Hilfe der Indexfunktion $m(\mathbf{i})$, kann nun die Differenz $\mathbf{i}_2^M - \mathbf{i}_1^M$ der Indizes $\mathbf{i}_1^M$ und $\mathbf{i}_2^M$ im Indexraum $\mathcal{I}_m$ ermittelt werden:

$$\mathbf{i}_2^M - \mathbf{i}_1^M = \mathbf{M} \cdot \mathbf{i}_2 + \mathbf{m} - (\mathbf{M} \cdot \mathbf{i}_1 + \mathbf{m}) = \mathbf{M} \cdot (\mathbf{i}_2 - \mathbf{i}_1) = \mathbf{M} \cdot \boldsymbol{\Gamma}^t.$$

Liegen die beiden Indizes $\mathbf{i}_1^M$ *und* $\mathbf{i}_2^M$ *im Indexraum* $\mathcal{I}_m$ *in der gleichen Weise nebeneinander, wie die Iterationen* $\mathbf{i}_1$ *und* $\mathbf{i}_2$ *im Iterationsraum* $\mathcal{I}$*, so muss für die Differenz der Indizes gelten:* $\mathbf{i}_2^M - \mathbf{i}_1^M = \mathbf{\Gamma}^{M,t}$*, denn* $\mathbf{\Gamma}^M$ *gibt die Raumrichtung* j_m *(*$\gamma_{1,j_m}^M = 1$*) an, in der die Daten im Speicher gepackt sind.*

Gilt für die Gleichung $\mathbf{M} \cdot \mathbf{\Gamma}^t = -\mathbf{\Gamma}^{M,t}$, so sind die Daten spiegelverkehrt im Speicher angeordnet.

Beispiel 6.14 (Eingabedaten des FIR-Filters)
Die Eingabedaten $x[.]$ *des FIR-Filters, siehe Beispiel 3.1 auf Seite 23 sind im Speicher linear hintereinander angeordnet (*$\mathbf{\Gamma}^M = \begin{pmatrix}1\end{pmatrix}$*). Die Indexfunktion für diese Eingabedaten lautet:* $\mathbf{i}^M = m(\mathbf{i}) = \begin{pmatrix}1 - 1\end{pmatrix} \mathbf{i}$*. Bei einer Partitionierung des zweidimensionalen Iterationsraum mit* $\mathbf{\Theta}_1^P = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ *erhalten wir eine Allokationsmatrix* $\mathbf{\Gamma}_1 = \begin{pmatrix}1 & 0\end{pmatrix}$*. Daraus folgt:* $\begin{pmatrix}1 - 1\end{pmatrix} \left(\begin{smallmatrix}1\\ 0\end{smallmatrix}\right) = 1 = \mathbf{\Gamma}^{M,t}$*. Die Teilwörter in den Berechnungs- und in den Speicher-DvBs sind somit gleichartig angeordnet.*

Wird dagegen die Partitionierung $\mathbf{\Theta}_2^P = \left(\begin{smallmatrix}1 & 0\\ 0 & 4\end{smallmatrix}\right)$*, mit der Allokationsmatrix* $\mathbf{\Gamma}_2 = \begin{pmatrix}0 & 1\end{pmatrix}$ *gewählt, so erhalten wir* $\begin{pmatrix}1 - 1\end{pmatrix} \left(\begin{smallmatrix}0\\ 1\end{smallmatrix}\right) = -1 = -\mathbf{\Gamma}^{M,t}$*. Die Eingabedaten sind spiegelverkehrt im Speicher angeordnet.*

Ausgerichtete Daten Sind die Teilwörter in den Berechnungs- und in den Speicher-DvBs gleichartig angeordnet, interessiert nun, ob die Berechnungs-DvBs ausgerichtet gelesen oder geschrieben werden können. Ein Speicherzugriff ist ausgerichtet, wenn die Teilwörter in den Berechnungs- und in den Speicher-DvBs gleichartig angeordnet sind und wenn das Teilwort $y[0, \widehat{\boldsymbol{\kappa}}^P]$ des Berechnungs-DvBs $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P]$ dem Teilwort $y_M[0, \widehat{\boldsymbol{\kappa}}^M, 0]$ des Speicher-DvBs $\mathbf{y}_M[\widehat{\boldsymbol{\kappa}}^M, 0]$ zugeordnet ist.

Somit muss für Gleichung 6.93 gelten:

$$b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P, 0) = \left(\mathbf{\Gamma}^M \cdot \mathbf{M} \cdot \mathbf{\Theta}^P \cdot \widehat{\boldsymbol{\kappa}}^P + \mathbf{\Gamma}^M \cdot \mathbf{m}\right) \mod B = 0, \quad \forall\, \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}. \quad (6.95)$$

Um nun zu ermitteln, ob $b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P, 0) = 0$ für alle $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}$ gilt, kann wie folgt vorgegangen werden:

- Berechnung der unteren $\widehat{\kappa}_{\min,j}^P$ und oberen $\widehat{\kappa}_{\max,j}^P$ Grenzen der Iterationselemente $\widehat{\kappa}_j^P$ mittels der FME der Variablen $\widehat{\kappa}_i^P$, $i \neq j$ aus dem Partitionsraum $\widehat{\mathcal{K}}$. Die Grenzen werden in den Vektoren $\widehat{\boldsymbol{\kappa}}_{\min}^P$ und $\widehat{\boldsymbol{\kappa}}_{\min}^P$ zusammengefasst.
- Bestimmung der Zeilenmatrix $\mathbf{H} = \mathbf{\Gamma}^M \cdot \mathbf{M} \cdot \mathbf{\Theta}^P \in \mathbb{Z}^{1 \times n}$.
- Wenn $\widehat{\kappa}_{\min,j}^P < \widehat{\kappa}_{\max,j}^P$ ist, so muss $H_{1,j} \mod B = 0$ sein.
- Ist $\widehat{\kappa}_{\min,j}^P = \widehat{\kappa}_{\max,j}^P$, so wird das Produkt $h_j = H_{1,j} \cdot \widehat{\kappa}_{\min,j}^P$ bestimmt. Anderenfalls gilt: $h_j = 0$.

- Die Summe der Werte von h_j plus dem Produkt $\mathbf{\Gamma}^M \cdot \mathbf{m}$ muss immer ein Vielfaches von B sein, damit b^M immer Null ist:

$$\left(\sum_{1\le j\le n} h_j + \mathbf{\Gamma}^M \cdot \mathbf{m}\right) \mod B = 0. \tag{6.96}$$

Nur wenn die beiden Bedingungen:

$$H_{1,j} \mod B = 0 \text{ für } \widehat{\kappa}^P_{\min,j} < \widehat{\kappa}^P_{\max,j} \qquad \text{und} \qquad \left(\sum_{1\le j\le n} h_j + \mathbf{\Gamma}^M \cdot \mathbf{m}\right) \mod B = 0$$

erfüllt sind, ist $b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P, 0) = 0$ für alle $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}$.

Transformation der Eingabezuweisungen

Ein allgemeiner Ansatz für die Transformation der Eingabezuweisung mit einer Teilzuweisung

$$s: \qquad y_s[\mathbf{i}] = y_M[m(\mathbf{i})], \qquad \mathbf{i} \in \mathcal{I}_s,$$

bei der nur ausgerichtete Speicherzugriffe ($\vartheta_0^M = 0$) zulässig sind, sieht wie folgt aus:

(1) Lies für jedes Teilwort $y_s[b, \widehat{\boldsymbol{\kappa}}^P]$ das zugehörige DvB $\mathbf{y}_M[\widehat{\boldsymbol{\kappa}}^M]$ aus dem Speicher:

$$t_{1,b}: \qquad \mathbf{y}_{t_{1,b}}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_M[\widehat{\boldsymbol{\kappa}}^M], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s,$$

mit $b_{\min} \le b \le b_{\max}$ und $\widehat{\boldsymbol{\kappa}}^M = m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, 0)$.

(2) Verschiebe das DvB $\mathbf{y}_{t_1,b}[\widehat{\boldsymbol{\kappa}}^P]$ so, dass das benötigte Teilwort an der richtigen Position ist:

$$t_{2,b}: \qquad \mathbf{y}_{t_2,b}[\widehat{\boldsymbol{\kappa}}^P] = \text{schiebeDvB}\Big(\mathbf{y}_{t_1,b}[\widehat{\boldsymbol{\kappa}}^P], b - b^M\Big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s,$$

mit $b_{\min} \le b \le b_{\max}$ und $b^M = m_b(b, \widehat{\boldsymbol{\kappa}}^P, 0)$.

(3) Setze Ergebnis-DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ aus den Teilwörtern $y_{t_2,b}[b, \widehat{\boldsymbol{\kappa}}^P]$ zusammen:

$$t_{3,0}: \qquad \mathbf{y}_{t_3,0}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_{t_2,0}[\widehat{\boldsymbol{\kappa}}^P], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s,$$
$$t_{3,b}: \qquad \mathbf{y}_{t_3,b}[\widehat{\boldsymbol{\kappa}}^P] = \text{ersetzeTW}\Big(b, \mathbf{y}_{t_2,b}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{t_3,b-1}[\widehat{\boldsymbol{\kappa}}^P]\Big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s,$$

mit $b_{\min} + 1 \le b \le b_{\max}$.

Die Bedingung, dass in einem UItA jede Variable nur einmal zugewiesen wird, macht die Verwendung der Hilfs-DvBs $\mathbf{y}_{t_3,b_{\min}}[.], \mathbf{y}_{t_3,b_{\min}+1}[.], \ldots, \mathbf{y}_{t_3,b_{\max}}[.]$ notwendig.

(4) Schreibe Ergebnis-DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$:

$$s: \qquad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_{t_3,b_{\max}}[\widehat{\boldsymbol{\kappa}}^P], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s.$$

Die Funktionen der TWP-Operationen sehen wie folgt aus:

- $\mathbf{y} = \text{schiebeDvB}(\mathbf{x}, s)$

$$y[b] = \begin{cases} x[b-s] & \text{wenn } 0 \le b-s < B \\ 0 & \text{sonst.} \end{cases} \tag{6.97}$$

- $\mathbf{y} = \text{ersetzeTW}(s, \mathbf{x}_1, \mathbf{x}_2)$

$$y[b] = \begin{cases} x_1[b] & \text{wenn } b = s \\ x_2[b] & \text{sonst.} \end{cases} \tag{6.98}$$

Somit erhalten wir aus den $b_{\text{eff}} = b_{\max} - b_{\min} + 1$ Einzeldatenoperationen des Ausgangsalgorithmus $3 \cdot b_{\text{eff}} + 1$ TWP-Operationen im transformierten Algorithmus. Doch je nach der Vollständigkeit des Berechnungs-DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$, der Anzahl der zu lesenden Speicher-DvBs $\mathbf{y}_M[\widehat{\boldsymbol{\kappa}}^M]$ und der notwendigen Verschiebung $b - b^M$ der Teilwörter kann sich die Anzahl der TWP-Operationen deutlich reduzieren. Nachfolgend sollen diese Versionen vorgestellt werden.

Speicher-DvBs können unausgerichtet gelesen werden Erlaubt die Speicherarchitektur einen unausgerichteten Speicherzugriff, so können durch unausgerichtetes Lesen der Speicher-DvBs die b_{eff} Zuweisungen $t_{2,b_{\min}}, t_{2,b_{\min}+1}, \ldots, t_{2,b_{\max}}$ zur Positionierung der Teilwörter vermieden werden.

$$t_{1,b}: \quad \mathbf{y}_{t_1,b}[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} \mathbf{y}_M[m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, 0), m_b(b, \widehat{\boldsymbol{\kappa}}^P, 0) - b], & \text{wenn} \quad b \le b^M, \\ \mathbf{y}_M[m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, 0) - \boldsymbol{\Gamma}^M, B + m_b(b, \widehat{\boldsymbol{\kappa}}^P, 0) - b], & \text{sonst.} \end{cases}$$

Nur ein Teilwort muss je Berechnungs-DvB gelesen werden Muss nur ein Teilwort des Berechnungs-DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ gelesen werden ($b_{\text{eff}} = 1$), so werden maximal nur noch zwei TWP-Operationen benötigt, denn das „Zusammensetzen“ des DvBs kann entfallen.

Gleiche Speicher-DvBs Wird das gleiche Speicher-DvB für mehrere Teilwörter des Berechnungs-DvBs benötigt, so muss das Speicher-DvB nur einmal gelesen werden.

Relative Position des Teilworts ist gleich Ist die relative Position b^M des Teilworts $y_M[b^M, \widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ im Speicher-DvB und die relative Position b des Teilworts $y_s[b, \widehat{\boldsymbol{\kappa}}^P]$ im Berechnungs-DvB gleich, so kann die Schiebeoperation entfallen.

Gleiche Verschiebung Müssen nach dem Lesen der Speicher-DvBs die temporären DvBs $\mathbf{y}_{t_1,b}$ mit dem gleichen Verschiebungswert verschoben werden, so können diese Teilwörter in einem DvB zusammengefasst werden und anschließend gemeinsam verschoben werden.

Gleichzeitiges Ersetzen mehrerer Teilwörter Befinden sich durch das gemeinsame Lesen von Teilwörtern und/oder dem gemeinsamen Verschieben der Teilwörter mehrere Teilwörter nebeneinander in einem temporären DvB $\mathbf{y}_{t_2,x}$, so können diese gemeinsam in das Ziel-DvB $\mathbf{y}_{t_3,x}$ eingefügt werden. Hierzu definieren wir eine neue Operation $\mathbf{y} = \text{ersetzeTWs}\big(s, \mathbf{x}_1, \mathbf{x}_2\big)$

$$y[b] = \begin{cases} x_1[b], & \text{wenn } b \geq s, \\ x_2[b], & \text{sonst.} \end{cases} \tag{6.99}$$

Diese Operation kann nur verwendet werden, wenn die zu ersetzenden Teilwörter lückenlos im DvB $\mathbf{x}_1$ nebeneinander angeordnet sind.

Daten sind gleichartig angeordnet Sind die Daten im Speicher-DvB und im Berechnungs-DvB gleichartig angeordnet und müssen mehrere Teilwörter für das Berechnungs-DvB gelesen werden, so vereinfacht sich der Lesevorgang weiter.

Erlaubt die Speicherarchitektur unausgerichtete Speicherzugriffe ($0 \leq \vartheta_0^M < B$) oder sind die Daten ausgerichtet, so können alle Teilwörter des Berechnungs-DvBs mit einer TWP-Zuweisung

$$s: \qquad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_M[m_{\widehat{\kappa}}(0, \widehat{\boldsymbol{\kappa}}^P, 0), m_b(0, \widehat{\boldsymbol{\kappa}}^P, 0)], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_s.$$

gelesen werden.

Für den Fall, dass die Daten unausgerichtet sind und ein solcher Speicherzugriff nicht möglich ist, müssen zwei benachbarte Speicher-DvBs ausgerichtet gelesen und daraus das Berechnungs-DvB erzeugt werden. Eine elegante Methode zur Bestimmung der zu lesenden Speicher-DvBs und der Packoperation ist das Hinzufügen einer zusätzlichen Eingabezuweisung i, welche mit $\mathbf{C} = \mathbf{E}$ und $\mathbf{c}_i \in \mathbb{Z}^n$, siehe Abschnitt 4.1 ab Seite 38, reindiziert wird, sodass der Speicherzugiff ein ausgerichteter Speicherzugiff ist:

$$\begin{aligned} i: &\qquad y_i[\mathbf{i}] = y_M[\mathbf{M} \cdot \mathbf{i} + \mathbf{m} - \mathbf{M} \cdot \mathbf{c}_i], &&\mathbf{i} \in \widetilde{\mathcal{I}}_s, \\ s: &\qquad y_s[\mathbf{i}] = y_i[\mathbf{i} + \mathbf{c}_i], &&\mathbf{i} \in \mathcal{I}_s. \end{aligned}$$

Hierdurch entsteht der Abhängigkeitsvektor $\mathbf{d}_{i,s}^{k_s} = \mathbf{d} = -\mathbf{c}_i$, welcher im Allgemeinen bei der Partitionierung in zwei partitionierte Abhängigkeitsvektoren zerfällt und damit eine Packoperation entsprechend Abschnitt 6.2 verursacht. Die Zuweisung s ist nun eine interne Zuweisung und wird, wie in Abschnitt 6.4 gezeigt, transformiert.

Im Allgemeinen gibt es unendlich viele Möglichkeiten des Verschiebens, sodass der Speicherzugriff ein ausgerichteter wird. Wir wollen nur zwei Verschiebungen betrachten, die

kürzeste mit nur positiven und Nullelementen und die kürzeste mit nur negativen und Nullelementen. Hierzu legen wir für den Reindizierungsvektor $\mathbf{c}_i = c_i \cdot \mathbf{\Gamma}^t$ fest.

Bestimmen wir nun für die reindizierte Zuweisung entsprechend (6.96) die DvB-Verschiebung $\widetilde{b}^M$, welche für einen ausgerichteten Speicherzugriff Null sein muss, so folgt:

$$\left(\sum_{1 \le j \le n} h_j + \mathbf{\Gamma}^M \cdot \mathbf{m} - \mathbf{\Gamma}^M \cdot \mathbf{M} \cdot \mathbf{\Gamma}^t \cdot c_i \right) \mod B = 0. \tag{6.100}$$

Da die Daten gleichartig angeordnet sind, erhalten wir für $\mathbf{M} \cdot \mathbf{\Gamma}^t = \mathbf{\Gamma}^{M,t}$. Das Produkt der beiden Matrizen $\mathbf{\Gamma}^M$ und $\mathbf{\Gamma}^{M,t}$ ergibt wiederum '1'. Somit folgt:

$$\left(\sum_{1 \le j \le n} h_j + \mathbf{\Gamma}^M \cdot \mathbf{m} - c_i \right) \mod B = 0, \tag{6.101}$$

$$\Bigg(\underbrace{\left(\sum_{1 \le j \le n} h_j + \mathbf{\Gamma}^M \cdot \mathbf{m} \right) \mod B}_{\text{DvB-Verschiebung } b^M \text{ ohne Reindizierung}} - c_i \Bigg) \mod B = 0. \tag{6.102}$$

Der kleinste positive Wert und der kleinste negative Wert für c_i, der zu einem ausgerichteten Speicherzugriff führt sind: $c_i^+ = b^M$ und $c_i^- = B - b^M$. Für den Reindizierungsvektor erhalten wir damit $\mathbf{c}_i^+ = c_i^+ \cdot \mathbf{\Gamma}^t$ oder $\mathbf{c}_i^- = c_i^- \cdot \mathbf{\Gamma}^t$ und der Abhängigkeitsvektor $\mathbf{d}$ beträgt $\mathbf{d}^+ = c_i^- \cdot \mathbf{\Gamma}^t$ oder $\mathbf{d}^- = c_i^+ \cdot \mathbf{\Gamma}^t$.

Die Entscheidung zwischen diesen beiden Reindizierungsvektoren ist nun abhängig von den anderen Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ des zu transformierenden Algorithmus.

(1) Existiert bereits ein Abhängigkeitsvektor $\exists_{e \in \mathcal{E}}\ \mathbf{d}(e) = k^+ \cdot \mathbf{\Gamma}^t$ oder $\exists_{e \in \mathcal{E}}\ \mathbf{d}(e) = -k^+ \cdot \mathbf{\Gamma}^t$, $k^+ \in \mathbb{N}^+$, so muss der Reindizierungsvektor $\mathbf{c}_i$ so gewählt werden, dass für den neuen Abhängigkeitsvektor $\mathbf{d} = -c_i \cdot \mathbf{\Gamma}$ ebenfalls gilt: $\mathbf{d} = k^+ \cdot \mathbf{\Gamma}^t$ oder $\mathbf{d} = k^- \cdot \mathbf{\Gamma}^t$.

(2) Anderenfalls überprüfen wir alle Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$, in der Raumrichtung j, in der die Teilwörter zu DvBs gepackt werden ($\gamma_{1,j} = 1$), ob diese Vektorelemente $d(e)_j$ mehrheitlich positiv oder negativ sind und wählen den Reinizierungsvektor $\mathbf{c}_i$ so aus, dass sich der Abhängigkeitsvektor $\mathbf{d} = -\mathbf{c}_i$ der Mehrheit der anderen Abhängigkeitsvektoren anpasst.

Beispiel 6.15
Wir haben die Eingabezuweisung

$$s: \qquad \mathbf{y}_s \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \mathbf{y}_M \begin{bmatrix} i_1+1 \\ i_2 \end{bmatrix}, \qquad \mathbf{i} \in \mathcal{I}_s.$$

Bei einer Teilwortparallelisierung mit $\mathbf{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ *erhalten wir Speicher- und Berechnungs-DvBs die gleichartig angeordnet, aber nicht ausgerichtet* ($b^M = 1$) *sind. Durch das Hinzufügen der neuen Zuweisung* i*, welche wir mit* $\mathbf{c}_i^+ = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ *oder* $\mathbf{c}_i^- = \left(\begin{smallmatrix} -3 \\ 0 \end{smallmatrix}\right)$ *reindizieren,*

erhalten wird ausgerichtete Speicher- und Berechnungs-DvBs. Der neue Abhängigkeitsvektor beträgt: $\mathbf{d}^+ = \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)$ *oder* $\mathbf{d}^- = \left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)$.

Wir nehmen an, der Algorithmus, zu der Zuweisung s gehört, besitzt folgende Abhängigkeitsvektoren: $\mathbf{d}_1 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$, $\mathbf{d}_2 = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$, $\mathbf{d}_3 = \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)$ *und* $\mathbf{d}_4 = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$.

Für keinen Abhängigkeitsvektor $\mathbf{d}_i$, $i \in \{1,2,3,4\}$ *gilt:* $\mathbf{d}_i = \pm k \cdot \mathbf{\Gamma}^t$. *So muss für Raumrichtung* $j = 1$ *überprüft werden, ob die Vektorelemente mehrheitlich positiv oder negativ sind. Zwei Abhängigkeitsvektoren* $\mathbf{d}_1$ *und* $\mathbf{d}_2$ *besitzen ein positives Vektorelement* $d_{1,1} = 1$ *bzw.* $d_{2,1} = 2$ *und einen Abhängigkeitsvektor* $\mathbf{d}_3$ *besitzt ein negatives Vektorelement* $d_{3,1} = -1$ *in der Raumrichtung* $j = 1$. *Somit wählen wir den Reinizierungsvektor* $\mathbf{c}_i^- = \left(\begin{smallmatrix}-3\\0\end{smallmatrix}\right)$ *aus, der zu einem positiven Abhängigkeitsvektor* $\mathbf{d}^+ = \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)$ *führt.*

Zusammenfassung Je nach Anzahl der Teilwörter $y_M[\mathbf{i}^M]$, die für ein Berechnungs-DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ gelesen werden müssen, der Anordnung der Daten im Speicher und der Lesemöglichkeit der Architektur werden 1 bis $3 \cdot B + 1$ TWP-Operationen zum Lesen der maximal B Teilwörter eines Berechnungs-DvBs benötigt. Sind die Berechnungs- und Speicherdaten gleichartig angeordnet, so reduziert sich die maximale Anzahl der TWP-Operationen auf zwei. Die Reduzierung dieser TWP-Operationen durch entsprechende Reinizierung und Coartitionierung des Algorithmus ist auch Bestandteil der Abbildungsstrategie, welche in Kapitel 7 vorgestellt wird.

Nicht gleichartig angeordnet Daten können durch ein Umorganisieren der Speicherdaten zu gleichartigen Daten umgewandelt werden. Damit reduziert sich die Anzahl an TWP-Operationen meist merklich. Ruby Lee hat für das Umorganisieren von Daten effiziente Befehle entwickelt [Lee96, Lee00]. Ein Umorganisieren der Daten lohnt sich aber meist nur, wenn bei einer Eingabezuweisung die B Teilwörter des Berechnungs-DvBs aus B verschiedenen Speicher-DvBs ausgelesen werden müssen.

Beispiel 6.16 (Transformation der Eingabezuweisung des FIR-Filters)
Die Eingabezuweisung s_2 für die x-Daten des FIR-Filters, siehe Algorithmus 3.1 auf Seite 23, besitzt drei Teilzuweisungen. Die ersten beiden Teilzuweisungen sind Eingabeteilzuweisungen. Für diese Teilzuweisung wollen wir die Transformation vorstellen, wobei wir die beiden Teilzuweisungen als zwei unabhängige Zuweisungen $s_{2,1}$ und $s_{2,2}$ betrachten:

$$s_{2,1}: \quad y_{x,1}[\mathbf{i}] = x[\left(\begin{smallmatrix}1 & -1\end{smallmatrix}\right)\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_{s_{2,1}},$$
$$s_{2,2}: \quad y_{x,2}[\mathbf{i}] = x[\left(\begin{smallmatrix}1 & -1\end{smallmatrix}\right)\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_{s_{2,2}}.$$

Die Partitionierung des zweidimensionalen Iterationsraums $\mathcal{I} \supset \mathcal{I}_{2,1}, \mathcal{I}_{2,2}$ kann mit $\mathbf{\Theta}_1^P = \left(\begin{smallmatrix}4 & 0\\0 & 1\end{smallmatrix}\right)$, $\mathbf{\Gamma}_1 = \begin{pmatrix}1 & 0\end{pmatrix}$ *oder* $\mathbf{\Theta}_2^P = \left(\begin{smallmatrix}1 & 0\\0 & 4\end{smallmatrix}\right)$, $\mathbf{\Gamma}_2 = \begin{pmatrix}0 & 1\end{pmatrix}$ *erfolgen. Die eindimensionalen Eingabedaten sind im Speicher mit* $\mathbf{\Theta}^M = \begin{pmatrix}4\end{pmatrix}$, $\mathbf{\Gamma}^M = \begin{pmatrix}1\end{pmatrix}$ *partitioniert. Es sind nur ausgerichtete Speicherzugriffe ($\vartheta_0^M = 0$) möglich.*

Wird die Partitionierung $\mathbf{\Theta}_1^P = \left(\begin{smallmatrix}4 & 0\\0 & 1\end{smallmatrix}\right)$ *gewählt, so muss bei der teilwortparallelen Zuweisung $s_{2,1}$ ein Teilwort für jedes DvB $\mathbf{y}_{x,1}[\widehat{\boldsymbol{\kappa}}^P]$ gelesen werden. Die Position des Teilworts*

ist $b = 0$. Da im Allgemeinen die relative Position des Teilworts im Speicher-DvBs und im Berechnungs-DvB nicht übereinstimmt, muss zuerst das zugehörige Speicher-DvB gelesen werden und anschließend das Teilwort auf Position $b = 0$ geschoben werden. Die Teilwortparallelisierung der Zuweisung $s_{2,2}$ führt zu einem ausgerichteten Speicherzugriff, denn es gilt $\mathbf{\Gamma}^M = \begin{pmatrix}1 & 0\end{pmatrix} \cdot \mathbf{\Gamma}^{M,t}$ und $b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P, 0) = 0$ für alle $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_{2,2}}$. Somit erhalten wir folgende teilwortparallele Zuweisungen für unsere beide Ausgangszuweisungen:

$$
\begin{aligned}
t_{1,1_0}: &\quad \mathbf{y}_{1,1_0}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{x}[\lfloor \widehat{\kappa}_2^P/4 \rfloor], && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
s_{2,1}: &\quad \mathbf{y}_{x,1}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{schiebeTW}(\mathbf{y}_{1,1_0}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\kappa}_2^P \mod 4), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
s_{2,2}: &\quad \mathbf{y}_{x,2}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{x}[\widehat{\kappa}_1^P], && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,2}}.
\end{aligned}
$$

Die Iterationsräume $\mathcal{K}_{s_{2,1}}$ und $\mathcal{K}_{s_{2,2}}$ werden entsprechend Abschnitt 4.2 bestimmt.

Bei einer Transformation der Zuweisungen mit $\mathbf{\Theta}_2^P = \left(\begin{smallmatrix}1 & 0\\ 0 & 4\end{smallmatrix}\right)$ erhalten wir hingegen folgende Zuweisungen:

$$
\begin{aligned}
t_{1,1_0}: &\quad \mathbf{y}_{1,1_0}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{x}[-\widehat{\kappa}_2^P], && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{1,1_1}: &\quad \mathbf{y}_{1,1_1}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{x}[-\widehat{\kappa}_2^P - 1], && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{1,2_1}: &\quad \mathbf{y}_{1,2_1}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{schiebeDvB}\big(\mathbf{y}_{1,1_1}[\widehat{\boldsymbol{\kappa}}^P], 2\big), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{1,2_2}: &\quad \mathbf{y}_{1,2_2}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{schiebeDvB}\big(\mathbf{y}_{1,1_1}[\widehat{\boldsymbol{\kappa}}^P], 0\big), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{1,2_3}: &\quad \mathbf{y}_{1,2_{B-1}}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{schiebeDvB}\big(\mathbf{y}_{1,1_1}[\widehat{\boldsymbol{\kappa}}^P], -2\big), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{1,3_0}: &\quad \mathbf{y}_{1,3_0}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_{1,1_0}[\widehat{\boldsymbol{\kappa}}^P], && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{1,3_1}: &\quad \mathbf{y}_{1,3_1}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{ersetzeTW}\big(1, \mathbf{y}_{1,2_1}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{1,3_0}[\widehat{\boldsymbol{\kappa}}^P]\big), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{1,3_2}: &\quad \mathbf{y}_{1,3_2}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{ersetzeTW}\big(2, \mathbf{y}_{1,2_2}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{1,3_1}[\widehat{\boldsymbol{\kappa}}^P]\big), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
s_{2,1}: &\quad \mathbf{y}_{x,1}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{ersetzeTW}\big(3, \mathbf{y}_{1,2_3}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{1,3_2}[\widehat{\boldsymbol{\kappa}}^P]\big), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,1}},\\
t_{2,1_0}: &\quad \mathbf{y}_{2,1_0}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{x}[\lfloor \widehat{\kappa}_1^P/4 \rfloor], && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,2}},\\
s_{2,2}: &\quad \mathbf{y}_{x,2}[\widehat{\boldsymbol{\kappa}}^P] = \mathit{schiebeTW}(\mathbf{y}_{2,1_0}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\kappa}_1^P \mod 4), && \widehat{\boldsymbol{\kappa}}^P \in \mathcal{K}_{s_{2,2}}.
\end{aligned}
$$

Die Transformation der Zuweisung $s_{2,1}$ führt zu neun TWP-Zuweisungen. Da die Daten nicht gleichartig angeordnet sind, müssen zuerst die beiden Speicher-DvBs $\mathbf{x}[-\widehat{\kappa}_2^P]$ und $\mathbf{x}[-\widehat{\kappa}_2^P - 1]$ gelesen werden, in denen sich die Teilwörter für das Berechnungs-DvB $\mathbf{y}_{x,1}[\widehat{\boldsymbol{\kappa}}^P]$ befinden. Anschließend werden die Teilwörter an die richtige Position geschoben und abschließend findet das Zusammensetzen des Berechnungs-DvBs statt. Für die Zuweisung $s_{2,2}$ erhalten wir zwei TWP-Zuweisungen, hierbei muss je Berechnungs-DvB $\mathbf{y}_{x,2}[\widehat{\boldsymbol{\kappa}}^P]$ nur ein Teilwort gelesen werden und diese an Position $b = 0$ platziert werden.

Transformation der Ausgabezuweisungen

Die Transformation einer Ausgabezuweisungen ist der schwierigste Teil der Algorithmentransformation. Der Grund dafür ist, dass im transformierten Algorithmus nur eine

Ausgabezuweisung je Variable existieren darf, damit der transformierte Algorithmus weiterhin ein UItA ist.

Deshalb soll zuerst nur für Ausgabezuweisungen mit gleichartig angeordneten Berechnungs- und Speicherdaten die Zuweisungstransformation vorgestellt werden.

Die nachfolgenden beiden allgemeinen Ansätze für die Transformation von Ausgabezuweisungen zeigen die Gründe für die Beschränkung der Transformation auf Ausgabezuweisungen mit gleichartig angeordneten Berechnungs- und Speicherdaten. Im Allgemeinen entstehen bei beiden Ansätzen durch die Transformation TWP-Zuweisungen, die nicht mehr den Bedingungen eines UItA genügen. So ist bei dem ersten allgemeinen Ansatz die Einmal-Zuweisungs-Form nicht mehr gegeben und bei dem zweiten allgemeinen Ansatz müsste der Iterationsraum des transformierten Algorithmus mittels lineare begrenzter Gitter beschrieben werden. Beide Verstöße hätten weitreichende Folge bei einer Abbildung des transformierten Algorithmus auf ein Rechenfeld und sollen deshalb in dieser Arbeit nicht weiter betrachtet werden.

Transformation für gleichartig angeordnete Daten Sind die Daten des Speicher- und des Berechnungs-DvBs gleichartig angeordnet, d. h. es wurde eine Partitionierung gewählt,bei der für die Ausgabedaten $\mathbf{\Gamma}^{M,t} = \mathbf{M} \cdot \mathbf{\Gamma}^t$ gilt, so kann die Ausgabezuweisung

$$m: \qquad y_m[\mathbf{i}^M] = y_s[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_m \tag{6.103}$$

wie folgt transformiert werden.

Sind die Speicher- und Berechnungs-DvBs nicht nur gleichartig angeordnet, sondern auch noch ausgerichtet ($b_M = m_b(0, \widehat{\boldsymbol{\kappa}}^P) = 0$) oder erlaubt die Speicherarchitektur unausgerichtetn Speicherzugriffe, so kann das Berechnungs-DvB direkt in den Speicher geschrieben werden und man erhält die folgende Ausgabezuweisung

$$m: \qquad \mathbf{y}_m[m_{\widehat{\kappa}}(0, \widehat{\boldsymbol{\kappa}}^P), m_b(0, \widehat{\boldsymbol{\kappa}}^P)] = \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m. \tag{6.104}$$

Erlaubt die Speicherarchitektur nur ausgerichtete Speicherzugriffe und sind die Speicher-DvBs $\mathbf{y}_M[\widehat{\boldsymbol{\kappa}}^M]$ und die Berechnungs-DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ zueinander nicht ausgerichtet ($b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P) \neq 0$), so muss zuerst ein ausgerichtetes Berechnungs-DvBs $\mathbf{y}_o[\widehat{\boldsymbol{\kappa}}^P]$ aus den Berechnungs-DvBs $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ erzeugt werden, bevor das neue Berechnungs-DvBs $\mathbf{y}_o[\widehat{\boldsymbol{\kappa}}^P]$ ausgerichtet in den Speicher geschrieben werden kann. Hierzu fügen wir in den Ausgangsalgorithmus eine Transferzuweisung o ein:

$$o: \qquad y_o[\mathbf{i}] = y_s[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_m, \tag{6.105}$$

$$m: \qquad y_M[\mathbf{M} \cdot \mathbf{i} + \mathbf{m}] = y_o[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_m. \tag{6.106}$$

Diese Transferzuweisung o und die Ausgabezuweisung m reindizieren wir mit $\mathbf{C} = \mathbf{E}$ und $\mathbf{c}_m = \mathbf{c}_s = \mathbf{d}$, wie in Abschnitt 4.1 ab Seite 38 beschrieben:

$$o: \qquad y_o[\mathbf{i}] = y_s[\mathbf{i} - \mathbf{d}], \qquad \mathbf{i} \in \widetilde{\mathcal{I}}_m, \tag{6.107}$$

$$m: \qquad y_M[\mathbf{M} \cdot \mathbf{i} + \mathbf{m} - \mathbf{M} \cdot \mathbf{d}] = y_o[\mathbf{i}], \qquad \mathbf{i} \in \widetilde{\mathcal{I}}_m. \tag{6.108}$$

Die Teilwortparallelisierung der Ausgabezuweisung des reindizierten Algorithmus führt nun zu einem ausgerichteten Datentransfer und kann somit mit (6.104) teilwortparallelisiert werden.

Wie bei der Eingabezuweisung soll der kürzeste positive Reindizierungsvektor $\mathbf{d}^+ = d^+ \cdot \mathbf{\Gamma}^t$ und der kürzeste negative Reindizierungsvektor $\mathbf{d}^- = d^- \cdot \mathbf{\Gamma}^t$ ermittelt werden. Hierzu können die Gleichungen 6.100 - 6.102 auf Seite 120 der Reindizierung der Eingabezuweisungen verwendet werden.

Die Auswahl des Reindizierungsvektors aus den beiden Reindizierungsvektoren $\mathbf{d}^+$ und $\mathbf{d}^+$ erfolgt wie bei der Reindizierung der Eingabezuweisung entsprechnend der anderen Abhängigkeitsvektoren $\mathbf{d}(e), e \in \mathcal{E}$ des Algorithmus, siehe das Vorgehen für Eingabezuweisungen auf Seite 120.

Einzigster Unterschied der Reindizierung der Transfer- und Ausgabezuweisung gegenüber der Reindizierung der Eingabezuweisung ist, dass ein positiver Reindizierungsvektor $\mathbf{d}^+$ für die Transfer- und Ausgabezuweisung zu einem positiven Abhängigkeitsvektor $\mathbf{d}^+$ in Transferzuweisung o führt und dass wir mit dem negativen Reindizierungsvektor $\mathbf{d}^-$ einen negativen Abhängigkeitsvektor $\mathbf{d}^-$ erhalten.

Bei der vorgestellten Transformation gehen wir davon aus, dass mehrdimensionale Daten entsprechend Abschnitt 6.5.1 erweitert wurden. Sind die Ausgabedaten linearisierte mehrdimensionale Daten, so muss die Zeilenlänge der linearisierten Daten ebenfalls erweitert werden, sodass auf jedes DvB nur einmal schreibend zugegriffen wird. Hierzu wird der mehrdimensionale Indexraum erst entsprechend Abschnitt 6.5.1 erweitert, bevor der erweiterte mehrdimensionale Raum linearisiert werden kann.

Sind die Ausgabedaten z. B. zweidimensionale Daten $y_m[\mathbf{i}]$, $\mathbf{i} = \binom{i_1}{i_2} \in \mathcal{I}$ mit $0 \leq i_1 < I_1$ und $0 \leq i_2 < I_2$, welche im Speicher als eindimensionales Feld $y'_m[i_1 + I_1 \cdot i_2] = y'_m[i']$, $0 \leq i' < I_1 \cdot I_2 - 1$ liegen, so muss zur Erhaltung der Einmal-Zuweisungs-Form für die transformierte Zuweisung der zweidimensionale Indexraum $\mathcal{I}$ um $2 \cdot (B-1)$ Iterationen in Raumrichtung i_1 erweitert werden. Das eindimensionale Feld vergrößert sich damit zu $\big(I_1 + 2(B-1)\big) \cdot I_2$ und der Datenzugriff auf die ursprünglichen Daten lautet: $y'_m[(B-1) + i_1 + \big(I_1 + 2 \cdot (B-1)\big) \cdot i_2]$.

Allgemeiner Ansatz: Direkte Ausgabe der Teilwörter Ein erster allgemeiner Ansatz für die Transformation der Ausgabezuweisung mit einer Teilzuweisung, siehe (6.103), könnte wie folgt aussehen:

(1) Lies für jedes Teilwort $y_s[b, \widehat{\boldsymbol{\kappa}}^P]$ das zugehörige Speicher-DvB $\mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ aus dem Speicher:

$$t_{1,b}: \qquad \mathbf{y}_{t_1,b}[\widehat{\boldsymbol{\kappa}}^P] = \mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m,$$

mit $\widehat{\boldsymbol{\kappa}}^M = m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, 0)$.

(2) Verschiebe das Berechnungs-DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ so, dass das zu schreibende Teilwort an der richtigen Position ist:

$$t_{2,b}: \qquad \mathbf{y}_{t_2,b}[\widehat{\boldsymbol{\kappa}}^P] = \text{schiebeDvB}\Big(\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P], b - b^M\Big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m,$$

mit $b^M = m_b(b, \widehat{\boldsymbol{\kappa}}^P, 0)$.

(3) Ersetze das Teilwort $y_{t_2,b}[b, \widehat{\boldsymbol{\kappa}}^P]$ im zukünftigen Ausgabe-DvB $\mathbf{y}_{t_3,b}[\widehat{\boldsymbol{\kappa}}^P]$:

$$t_{3,b}: \qquad \mathbf{y}_{t_3,b}[\widehat{\boldsymbol{\kappa}}^P] = \text{ersetzeTW}\Big(b, \mathbf{y}_{t_2,b}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{t_1,b}[\widehat{\boldsymbol{\kappa}}^P]\Big), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m.$$

(4) Schreibe das DvB $y_{t_3,b}[b, \widehat{\boldsymbol{\kappa}}^P]$ in den Speicher:

$$m_b: \qquad \mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M] = \mathbf{y}_{t_3,b}[\widehat{\boldsymbol{\kappa}}^P], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m.$$

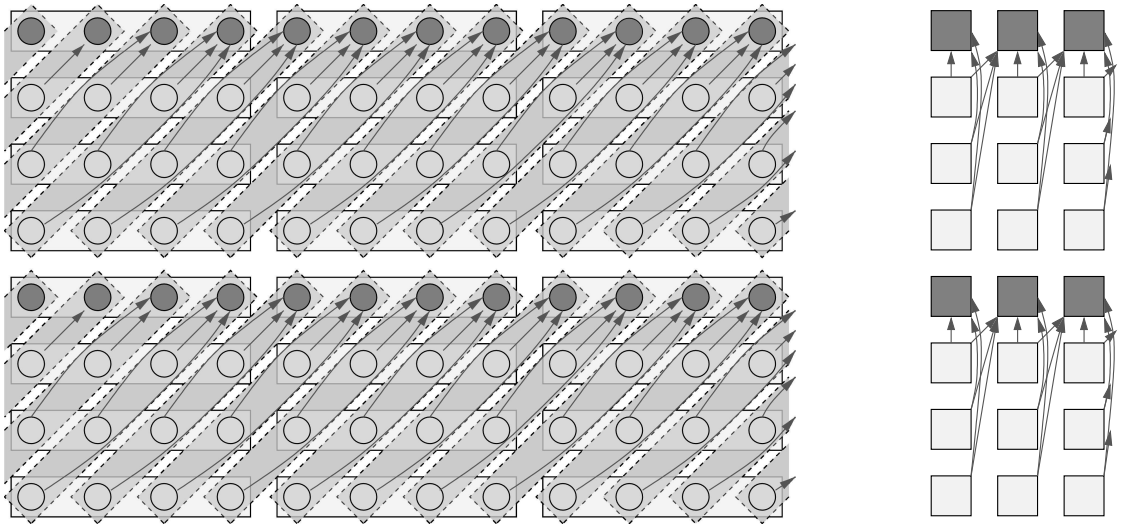
Für b gilt: $b_{\min} \leq b \leq b_{\max}$. Die TWP-Operationen sind die gleichen Operationen, die bei der Transformation der Eingabezuweisungen entstehen. Die TWP-Operation $\mathbf{y} = \text{schiebeDvB}\big(\mathbf{x}, s\big)$ verschiebt die Teilwörter des DvBs $\mathbf{x}$ um s Teilwörter innerhalb des DvBs nach links oder rechts, siehe (6.97) auf Seite 118, und die TWP-Operation $\mathbf{y} = \text{ersetzeTW}\big(s, \mathbf{x}_1, \mathbf{x}_2\big)$ ersetzt das Teilwort $x_2[s]$ in DvB $\mathbf{x}_2$ durch Teilwort $x_1[s]$ und gibt das veränderte DvB zurück, siehe (6.98) auf Seite 118.

Ein in dieser Weise transformierter Algorithmus hat im Allgemeinen keine Einmal-Zuweisungs-Form mehr und ist somit auch kein UItA. Der Grund hierfür ist das mehrmalige Schreiben des Ausgabe-DvBs $\mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ in Zuweisung m_b, auch wenn die Teilwörter in den Ausgabe-DvBs weiterhin jeweils nur einmal geschrieben werden. Bei der Implementierung eines solchen Algorithmus ist sicherzustellen, dass ein Datenausgabeprozess (Lesen des DvBs, Einfügen des Teilworts, Schreiben des DvBs) für ein Teilwort $y_m[b_1, \widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ abgeschlossen sein muss, bevor der nächste Datenausgabeprozess für das nächste Teilwort $y_m[b_2, \widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ dieses Ausgabe-DvBs begonnen werden kann.

Diese Beschränkung bei der Implementierung behindert die Parallelisierung des Algorithmus merklich, besonders bei einer nachfolgenden Abbildung des Algorithmus auf ein Rechenfeld. Deshalb wird dieser Ansatz in dieser Arbeit nicht weiter betrachtet.

Allgemeiner Ansatz: Zusammenbau der Speicher-DvBs vor Speicherung Die Grundidee dieses zweiten Ansatzes ist die Erzeugung des Ausgabe-DvBs in einer Iteration $\widehat{\boldsymbol{\kappa}}^P$ und das einmalige Schreiben dieses DvBs. Hierfür ist es notwendig, dass alle zu diesem DvB gehörigen Teilwörter zu dieser Iteration propagiert werden und dass für jedes Ausgabe-DvB eine Iteration festgelegt wird, in der das Zusammenstellen und Schreiben des Ausgabe-DvBs erfolgt.

Betrachten wir die Ausgabezuweisung: $y_m[\mathbf{i}^M] = y_m[\left(\begin{smallmatrix}1 & -1\\ 0 & 1\end{smallmatrix}\right)\mathbf{i}] = y[\mathbf{i}]$ mit $\mathbf{i} \in \mathcal{I} \subset \mathbb{Z}^2$ und $\mathbf{i}^M \in \mathcal{I}^M \subset \mathbb{Z}^2$. Der Iterationsraum $\mathcal{I}$ der Zuweisung wird mit $\boldsymbol{\Theta} = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ und der Indexraum $\mathcal{I}^M$ der Ausgabevariablen wird mit $\boldsymbol{\Theta}^M = \left(\begin{smallmatrix}1 & 0\\ 0 & 4\end{smallmatrix}\right)$ und $\vartheta_0^M = 0$ partitioniert.



(a) Überlagerung der Partitionen des Iterationsraums (durchgängige Linie) und des Indexraums (gestrichelte Linie)

(b) Partitionsraum

Abbildung 6.11: Alternative Transformation der Ausgabezuweisung

Das Ausgabe-DvB $\mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, 0]$ soll in der Iteration $\mathbf{i}$ zusammengebaut werden, in der die Variable $y[\mathbf{i}]$ für das letzte Teilwort $y_m[3, \widehat{\boldsymbol{\kappa}}^M, 0]$ des DvBs erzeugt wurde. Das ist immer in den Iterationen $\mathbf{i} = \binom{i_1}{4 \cdot j + 3}$, $i_1, j \in \mathbb{Z}$ der Fall. Somit müssen die anderen Teilwörter für das Speicher-DvB $\mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^M, 0]$ zu dieser Iteration transferiert werden, um das DvB zu erstellen und anschließend zu schreiben.

Abbildung 6.11(a) zeigt dieses Szenario. Die dunkelgrau dargestellten Iterationen, sind diejenigen Iterationen, in denen die Zusammenstellungen und Speicherungen der Speicher-DvBs erfolgen. Die Transfers der Teilwörter wurden durch Pfeile illustriert.

Betrachtet man das Szenario im partitionierten Iterationsraum $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m$ auf Basis von DvBs, so ergibt sich das in Abbildung 6.11(b) dargestellte Bild.

In den Iterationen $\widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ 4 \cdot j + 3 \end{pmatrix}$, $\widehat{\kappa}_1^P, j \in \mathbb{Z}$ werden jeweils vier Speicher-DvBs zusammengestellt und geschrieben. Der zugehörige transformierte Algorithmus ist in Algorithmus 6.2 angegeben.

Der transformierte Algorithmus hat zwar nun eine Einmal-Zuweisungs-Form, doch der Algorithmus erfüllt trotzdem nicht die Bedingungen für einen UItA, siehe Definition 3.1 auf Seite 21. Zum einem ist der Iterationsraum als linear begrenztes Gitter beschrieben und zum anderen erfolgt das Schreiben der Ausgabedaten mittels vier Gleichungen mit jeweils der gleichen linksseitigen Variable aber unterschiedlichen Indexfunktionen. Somit wird dieser Ansatz ebenfalls nicht weiter verfolgt.

Zusammenfassung Nicht jede Ausgabezuweisung kann teilwortparallelisiert werden, sodass der transformierte Algorithmus wieder ein UItA ist. Die beiden vorgestellten allgemeinen Ansätze für die Transformation von Ausgabezuweisungen erfordern eine Er-

Algorithmus 6.2 Beispiel für die alternative Ausgabetransformation

$$
\begin{aligned}
t_1: &\quad \mathbf{y}_{t_1}[\widehat{\boldsymbol{\kappa}}^P] = \mathrm{pack1}\Big(\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}1\\3\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}1\\2\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P]\Big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m\\
t_2: &\quad \mathbf{y}_{t_2}[\widehat{\boldsymbol{\kappa}}^P] = \mathrm{pack2}\Big(\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}1\\3\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}1\\2\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P]\Big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m\\
t_3: &\quad \mathbf{y}_{t_3}[\widehat{\boldsymbol{\kappa}}^P] = \mathrm{pack3}\Big(\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}1\\3\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}0\\2\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P]\Big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m\\
t_4: &\quad \mathbf{y}_{t_4}[\widehat{\boldsymbol{\kappa}}^P] = \mathrm{pack4}\Big(\mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}0\\3\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}0\\2\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)], \mathbf{y}[\widehat{\boldsymbol{\kappa}}^P]\Big), && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m\\
m_1: &\quad \mathbf{y}_M[m_{\widehat{\kappa}}(\widehat{\boldsymbol{\kappa}}^P, 0)] = \mathbf{y}_{t_1}[\widehat{\boldsymbol{\kappa}}^P], && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m\\
m_2: &\quad \mathbf{y}_M[m_{\widehat{\kappa}}(\widehat{\boldsymbol{\kappa}}^P, 1)] = \mathbf{y}_{t_2}[\widehat{\boldsymbol{\kappa}}^P], && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m\\
m_3: &\quad \mathbf{y}_M[m_{\widehat{\kappa}}(\widehat{\boldsymbol{\kappa}}^P, 2)] = \mathbf{y}_{t_3}[\widehat{\boldsymbol{\kappa}}^P], && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m\\
m_4: &\quad \mathbf{y}_M[m_{\widehat{\kappa}}(\widehat{\boldsymbol{\kappa}}^P, 3)] = \mathbf{y}_{t_4}[\widehat{\boldsymbol{\kappa}}^P], && \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_m
\end{aligned}
$$

$$\text{mit } \widehat{\mathcal{K}}_m = \left\{ \widehat{\boldsymbol{\kappa}}^P = \begin{pmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{pmatrix} \;\middle|\; \widehat{\kappa}_2^P = 4 \cdot j + 3 \wedge \widehat{\kappa}_1^P, j \in \mathbb{Z} \right\}.$$

weiterung der Algorithmenklasse, was wiederum weitreichenden Einfluss auf die Folgen der Algorithmentransformationen und damit die Abbildungsmethode hat. Aus diesem Grund schränken wir die Transformation der Ausgabezuweisungen auf Zuweisungen und Partitionierungen ein, die zu gleichartig angordneten Daten führen, denn damit ist der transformierte Algorithmus wieder ein UItA. Bei diesen Zuweisungen entstehen aus einer Einzeldatenzuweisung maximal zwei teilwortparallele Zuweisungen.

Durch die Verwendung der Packoperationen von Ruby Lee [Lee96] ist es möglich, die Ausgabedaten effizient umzusortieren. Somit können die Daten des Algorithmus gleichartig angeordnet gespeichert werden und mittels dieser Packoperationen werden die Daten anschließend in die ursprünglich vorgesehene Reihenfolge gebracht.

Beispiel 6.17 (Transformation der Ausgabezuweisung des FIR-Filters)
Die Ausgabezuweisung des FIR-Filters lautet:

$$s_4: \qquad y[\begin{pmatrix}1 & 0\end{pmatrix} \mathbf{i}] = y_y[\mathbf{i}], \qquad \mathbf{i} \in \mathcal{I}_{4_1}.$$

Bei einer Partitionierung des Iterationsraums $\mathcal{I} \supset \mathcal{I}_{4_1}$ mit $\boldsymbol{\Theta}_1^P = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ und damit $\boldsymbol{\Gamma}_1 = \begin{pmatrix}1 & 0\end{pmatrix}$, so erhalten wir gleichartig angeordnete Berechnungs- und Speicher-DvBs: $\boldsymbol{\Gamma}^{M,t} = \begin{pmatrix}1\end{pmatrix} = \mathbf{M} \cdot \boldsymbol{\Gamma}_1^t = \begin{pmatrix}1 & 0\end{pmatrix} \cdot \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$. Die teilwortparallele Zuweisung lautet damit:

$$s_4: \qquad \mathbf{y}[\begin{pmatrix}1 & 0\end{pmatrix} \widehat{\boldsymbol{\kappa}}^P, 0] = \mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^P], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{4_1}.$$

Die Speicherzugriffe sind immer ausgerichtete Speicherzugriffe. Der Indexraum $\mathcal{I}^M$ der Variable $y[\mathbf{i}^M]$ wurde soweit erweitert, dass alle Speicherzugriffe gültig sind.

Die Partitionierung des Iterationsraums $\mathcal{I} \supset \mathcal{I}_{4_1}$ mit $\boldsymbol{\Theta}_2^P = \left(\begin{smallmatrix}1 & 0\\ 0 & 4\end{smallmatrix}\right)$ und damit $\boldsymbol{\Gamma}_2 = \begin{pmatrix}0 & 1\end{pmatrix}$ führt zu Berechnungs-DvBs, von denen nur ein Teilwort als Ausgabe in den Speicher geschrieben werden muss. Die Speicherung der Ausgabedaten auf Basis von

DvBs kann nicht direkt erfolgen, da die Daten zueinander nicht gleichartig angeordnet sind: $\begin{pmatrix}1\end{pmatrix} \neq \begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}$. *Deshalb definieren wir eine Hilfsausgabevariable* $y_m\left[\begin{smallmatrix}i_1\\i_2\end{smallmatrix}\right]$, *mit* $\mathcal{I}_m^M = \{\begin{pmatrix}i_1\\i_2\end{pmatrix} \mid 0 \leq i_1 < B \wedge 0 \leq i_2 < I\}$. *Diese Daten sind im Speicher in der Raumrichtung* i_1 *gepackt:* $\mathbf{\Gamma}_m^M = \begin{pmatrix}1 & 0\end{pmatrix}$. *Mit Hilfe der Indexfunktion* $l_m(\mathbf{i}) = \begin{pmatrix}0&1\\1&0\end{pmatrix} \cdot \mathbf{i} + \mathbf{0}$ *sind nun die Speicher-DvBs* $\mathbf{y}_m[.]$ *und die Berechnungs-DvBs* $\mathbf{y}_y[.]$ *gleichartig angeordnet:* $\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0&1\\1&0\end{pmatrix} \cdot \begin{pmatrix}0\\1\end{pmatrix}$. *Damit erhalten wir die folgende teilwortparalle Ausgabefunktion:*

$$s_4: \qquad \mathbf{y}_m[\widehat{\boldsymbol{\kappa}}^P, 0] = \mathbf{y}_y[\begin{pmatrix}0&1\\1&0\end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P], \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{4_1}.$$

In einem extra Algorithmus werden anschließend die eigentlichen Ausgabedaten $y_m\left[\begin{smallmatrix}b\\i_2\end{smallmatrix}\right]$ *in die Variable* $y[i_2]$ *überführt:* $y[i_2] = y_m\left[\begin{smallmatrix}b\\i_2\end{smallmatrix}\right]$. *Den Wert für den Index* b *erhalten wir wieder mit:* $b = (J-1) \mod 4$.

6.5.4 Eingabe- und Propagierungszuweisungen

Eine Eingabe- und Propagierungszuweisung ist eine Kombination aus einer Eingabeteilzuweisung und mehreren Propagierungsteilzuweisungen, siehe Abschnitt 3.3.3 ab Seite 33. Dabei ist die erste Teilzuweisung immer die Eingabeteilzuweisung.

Wurde die Partitionierung so gewählt, dass ein Propagierungsvektor $\mathbf{d}^1_{s,s_t}$ der Zuweisung gleich dem Allokationsmatrix $\mathbf{\Gamma}^t$ ist, so kann eine effizientere Teilwortparallelisierung verwendet werden. Diese Parallelisierung entspricht weitestgehend der Parallelisierung der internen Propagierungszuweisung, siehe Abschnitt 6.4.2 ab Seite 105. Nur für die erste Teilzuweisung, die Eingabeteilzuweisung, ist eine anderen Teilwortparallelisierung notwendig. Diese ist davon abhängig, ob die Daten als DvBs oder als Teilwörter gelesen werden können.

Unabhängig davon wird bei der Eingabeteilzuweisung s_1 immer nur ein Speicherteilwort $y_m[\mathbf{i}^M]$ je Berechnungs-DvB $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P]$ eingelesen. Dieses Teilwort wird dann mit der TWP-Operation *verteileTW()* auf die anderen Teilwörter des Berechnungs-DvBs verteilt. Die Position b des einzulesenden Teilworts im Berechnungs-DvB kann mit Hilfe der in Abschnitt 6.1.2 ab Seite 88 vorgestellten Methode bestimmt werden. Die Eingabeteilzuweisungen lauten nun, abhängig von der Speicherorganisation:

$$s_1: \qquad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \text{verteileTW}\left(y_m[\mathbf{M}^\kappa \cdot \widehat{\boldsymbol{\kappa}}^P + \mathbf{m}^\kappa(b)]\right), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1},$$

oder

$$s_1: \qquad \mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^P] = \text{verteileTW}\left(m_b(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M), \mathbf{y}_m[m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M)]\right), \qquad \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{s_1}.$$

Beispiel 6.18 (Eingabe- und Propagierungszuweisung des FIR-Filters)
Die Zuweisung s_1 *des FIR-Filters, siehe 3.1, ist eine Eingabe- und Propagierungszuweisung:*

$$s_1: \qquad y_a[\mathbf{i}] = \begin{cases} a[\begin{pmatrix}0&1\end{pmatrix}\mathbf{i}], & \mathbf{i} \in \mathcal{I}_{1_1}, \\ y_a[\mathbf{i} - \begin{pmatrix}1\\0\end{pmatrix}], & \mathbf{i} \in \mathcal{I}_{1_2}. \end{cases}$$

Wird der Iterationsraum $\mathcal{I} \supset \mathcal{I}_1$ mit $\mathbf{\Theta}^P = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ partitioniert, so kann diese Zuweisung effizient teilwortparallelisiert werden.

Werden die Eingabedaten auf Basis von Teilwörtern eingelesen, so erhalten wir die folgende transformierte Zuweisung:

$$s_1: \qquad \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} verteileTW\left(a[\left(\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right)\widehat{\boldsymbol{\kappa}}^P]\right), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1_1}, \\ \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1_2}. \end{cases}$$

Die Partitionsräume $\widehat{\mathcal{K}}_{1_1}$ und $\widehat{\mathcal{K}}_{1_2}$ haben die Größe:

$$\widehat{\mathcal{K}}_{1_1} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{smallmatrix}\right) \;\middle|\; \widehat{\kappa}_1^P = 0 \wedge 0 \leq \widehat{\kappa}_2^P < J \right\} \quad und$$
$$\widehat{\mathcal{K}}_{1_2} = \left\{ \widehat{\boldsymbol{\kappa}}^P = \left(\begin{smallmatrix} \widehat{\kappa}_1^P \\ \widehat{\kappa}_2^P \end{smallmatrix}\right) \;\middle|\; 1 \leq \widehat{\kappa}_1^P \leq \left\lceil \frac{I}{4} \right\rceil \wedge 0 \leq \widehat{\kappa}_2^P < J \right\}.$$

Das Lesen von DvBs aus dem Speicher erfolgt mit:

$$s_1: \qquad \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P] = \begin{cases} verteileTW\left(b^M, \mathbf{a}[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]\right), & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1_1}, \\ \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^P - \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)], & \widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}_{1_2}. \end{cases}$$

Die Indizes $b^M = m_b(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M)$ und $\widehat{\boldsymbol{\kappa}}^M = m_{\widehat{\kappa}}(b, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M)$ sind abhängig von der Iteration $\widehat{\boldsymbol{\kappa}}^P$, der Position b des zulesenden Teilworts und der Verschiebung ϑ_0^M, wobei für die Position b des Teilworts gilt $b = 0$:

$$\begin{aligned} b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M) &= (\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P - \vartheta_0^M) \mod 4, \\ \widehat{\boldsymbol{\kappa}}^M = m_{\widehat{\kappa}}(0, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M) &= \left\lfloor \frac{1}{4} \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P - \vartheta_0^M \right) \right\rfloor. \end{aligned}$$

Erlaubt die Architektur nur ausgerichtete Speicherzugriffe ($\vartheta_0^M = 0$) so erhalten wir:

$$\begin{aligned} b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P, 0) &= (\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P) \mod 4, \\ \widehat{\boldsymbol{\kappa}}^M = m_{\widehat{\kappa}}(0, \widehat{\boldsymbol{\kappa}}^P, 0) &= \left\lfloor \frac{1}{4} \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P \right) \right\rfloor. \end{aligned}$$

Sind hingegen auch unausgerichtete Speicherzugriffe ($0 \leq \vartheta_0^M < 4$) möglich, so könnten die Daten so gelesen werden, sodass das benötigte Teilwort $a[b^M, \widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ sich immer an Position $b^M = m_b(0, \widehat{\boldsymbol{\kappa}}^P, \vartheta_0^M) = 0$ des Speicher-DvBs $\mathbf{a}[\widehat{\boldsymbol{\kappa}}^M, \vartheta_0^M]$ befindet. Für die Verschiebung ϑ_0^M muss damit gelten:

$$\vartheta_0^M = \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P \right) \mod 4,$$

und für den Index $\widehat{\boldsymbol{\kappa}}^M$ erhalten wir dadurch:

$$\begin{aligned} \widehat{\boldsymbol{\kappa}}^M &= m_{\widehat{\kappa}}\Big(0, \widehat{\boldsymbol{\kappa}}^P, \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P \right) \mod 4\Big), \\ &= \left\lfloor \frac{1}{4} \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P - \left(\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \widehat{\boldsymbol{\kappa}}^P \right) \mod 4 \right) \right\rfloor. \end{aligned}$$

6.5.5 Zusammenfassung

Die Transformation der Ein- und Ausgabezuweisungen eines UItA sind von der Art des Speicherzugriffs, in Form von Teilwörtern oder in Form von DvBs, und der Anordnung der Teilwörter in den Speicher-DvB bezüglich der Anordnung der Teilwörter im Berechnungs-DvBs abhängig. In den Abschnitten 6.5.2 und 6.5.3 wurde die Transformation der Ein- und Ausgabezuweisungen abhängig von der Art des Speicherzugriffs vorgestellt.

Hierbei zeigte sich, dass durch die Transformation einer Eingabezuweisung bis zu $3 \cdot B + 1$ teilwortparallelen Zuweisungen entstehen können und, dass diese Anzahl, abhängig von der Partitionierung des Iterationsraums und der Art des Speicherzugriffs, auf eins reduziert werden kann.

Eine direkte Transformation einer Ausgabezuweisung ist nur möglich, wenn bei einem Datenzugriff in Form von Teilwörtern ein Teilwort je Berechnungs-DvB gespeichert werden muss, oder, wenn bei einem Datenzugriff in Form von DvBs die Daten in den Speicher- und Berechnungs-DvBs gleichartig angeordnet sind. Die maximale Anzahl an teilwortparallelen Zuweisungen nach der Transformation beträgt hierbei zwei. Kann keine der beiden Bedingungen durch die Partitionierung des Iterationsraums erfüllt werden, so müssen die Daten temporär entsprechend einer der beiden Bedingungen gespeichert werden. Anschließend muss ein Umsortieren der temporär gespeicherten Daten erfolgen, sodass die Daten danach in der vorgegebenen Form im Speicher vorliegen.

Die Beschreibung der Transformation der Eingabezuweisungen in den Abschnitten 6.5.2 und 6.5.3 zeigte, dass die Transformation der Eingabezuweisungen zu vielen teilwortparallelen Zuweisungen führen kann. Aus diesem Grund wurde für Eingabe- und Propagierungszuweisung in Abschnitt 6.5.4 eine effizientere Transformation unter Verwendung der TWP-Operation *verteileTW()* vorgestellt.

Beim Lesen und Schreiben der Daten aus dem Speicher ist weiterhin sicherzustellen, dass die Speicherzugriffe gültige Speicherzugriffe sind. In Abschnitt 6.5.1 wurde eine Erweiterung des Indexraums des globalen Variablen vorgestellt, die das gewährleistet.

Bei der Abbildungsstrategie im nachfolgenden Kapitel werden Optimierungsmethoden vorgestellt, welche die Parameter für die Reindizierung und Copartitionieurung des Algorithmus bestimmen, mit deren Hilfe die Anzahl an TWP-Operationen minimal ist.

6.6 Packbefehle

Wie in den Abschnitten 6.2 bis 6.5 gezeigt wurde, sind neun teilwortparallele Operationen für eine Teilwortparallelisierung eines UItA notwendig. Zwei dieser neun TWP-Operationen sind Ein-/Ausgabeoperationen zum Lesen und Schreiben von Teilwörtern bzw. DvBs. Für die anderen sieben TWP-Operationen sollen drei Packbefehle vorgestellt werden, mit deren Hilfe diese TWP-Operationen umgesetzt werden können:

- **Doppelwortschiebebefehl:** Dieser Befehl basiert auf der Operation $F_p(\mathbf{x}_1, \mathbf{x}_2, d)$. Zwei DvBs $\mathbf{x}_1$ und $\mathbf{x}_2$ werden gelesen und anschleißend gemeinsam teilwortweise

nach links verschoben. Das Ergebnis-DvB ist das um d Teile nach links verschobene Eingabe-DvB $\mathbf{x}_1$, wobei die oberen d Teilwörter des Eingabe-DvBs $\mathbf{x}_2$ die unteren d Teilwörter des Ausgabe-DvBs sind. Diesen Befehl können wir auch für die TWP-Operation $schiebeDvB(\mathbf{x}, d)$ verwenden, welche das Verschieben der Teilwörter innerhalb eines DvBs beschreibt.

- **Teilwortauswahlbefehl:** Mit diesem Befehl kann die teilwortparallele Operation $ersetzeTWs(b,\ \mathbf{x}_1, \mathbf{x}_2)$ direkt ausgeführt werden. Dabei werden die oberen b Teilwörter des Eingabe-DvBs $\mathbf{x}_2$ durch die oberen b Teilwörter des Eingabe-DvBs $\mathbf{x}_1$ ersetzt. Durch mehrfache Verwendung dieses Befehls können auch die TWP-Operationen $F_{m,\underline{\mathcal{C}}^p_{T_s}}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p, \widehat{\boldsymbol{\kappa}}^P)$ und $ersetzeTW(b, \mathbf{x}_1, \mathbf{x}_2)$ realisiert werden.
- **Datenverteilungsbefehl:** Dieser Befehl verteilt ein übergebenes Teilwort auf alle Teilwörter des Ausgabe-DvBs. Die beiden TWP-Operationen $verteileTW(x)$ und $verteileTW(b, \mathbf{x})$ können mit diesem Befehl realisiert werden. Wobei das zu verteilende Teilwort sich an der ersten Position $b = 0$ des Eingabe-DvBs $\mathbf{x}$ befinden muss. Gegebenenfalls ist das zu verteilende Teilwort mittels einer Verschiebeoperation an diese Position zu transferieren.

Besitzt eine Architektur diese drei Befehle plus teilwortparallele Lese- und Schreibbefehle, so können alle Algorithmen, welche mit der Teilwortparallelisierung in diesem Kapitel transformiert wurden, auf dieser Architekutur abgearbeitet werden. Alternativ kann auch gesagt werden, dass diese drei Befehle in ein Prozessorelement implementiert werden müssen, damit Algorithmen, welche entsprechend der in diesem Kapitel vorgestellten Methode teilwortparallelisiert wurden, auf diesem Prozessorelement abgebildet werden können.

Solche auf die Bedürfnisse von teilwortparallelisierten UItAs zugeschnittene Befehle stehen in bekannten Architekturen [Int07a][Fre06][Tex06] meist nicht oder nur teilweise zur Verfügung, oder der Rechenaufwand ist sehr groß. Aus diesem Grund soll in den nachfolgenden Abschnitten für die drei oben vorgestellten Befehle eine möglich Implementierung vorgestellt werden.

6.6.1 Doppelwortschiebebefehl „schiebeTWinDvBs“

Die Packoperation $\mathbf{y} = F_p(\mathbf{x}_1, \mathbf{x}_2, d)$, siehe (6.38), verschiebt die Teilwörter des Eingabe-DvBs $\mathbf{x}_1$ um d Teilwörter nach links und des Eingabe-DvBs $\mathbf{x}_2$ um $B-d$ Teilwörter nach rechts. Anschließend werden die unteren d Teilwörter des verschobenen Eingabe-DvBs $\mathbf{x}_2$ und die oberen $B-d$ Teilwörter des verschobenen Eingabe-DvBs $\mathbf{x}_1$ zum Ausgabe-DvBs zusammengefügt.

Alternativ könnten die d Teilwörter des Eingabe-DvBs $\mathbf{x}_2$ beim Verschieben der Teilwörter des Eingabe-DvBs $\mathbf{x}_1$ gleich von rechts in das Ausgabe-DvB eingeschoben werden. So erhalten wir mit einer Schiebeoperation bereits das gewünschte Ausgabe-DvB $\mathbf{y}$. Diese Beschreibung soll die Grundlage für unseren Doppelwortschiebebefehl

$$\mathbf{y} = \text{schiebeTWinDvBs}(\mathbf{x}_1, \mathbf{x}_2, d)$$

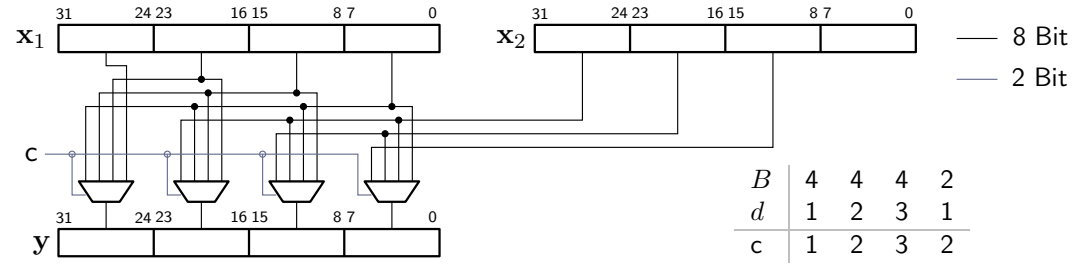


Abbildung 6.12: Schaltplan für Doppelwortschiebebefehl *schiebeTWinDvBs*

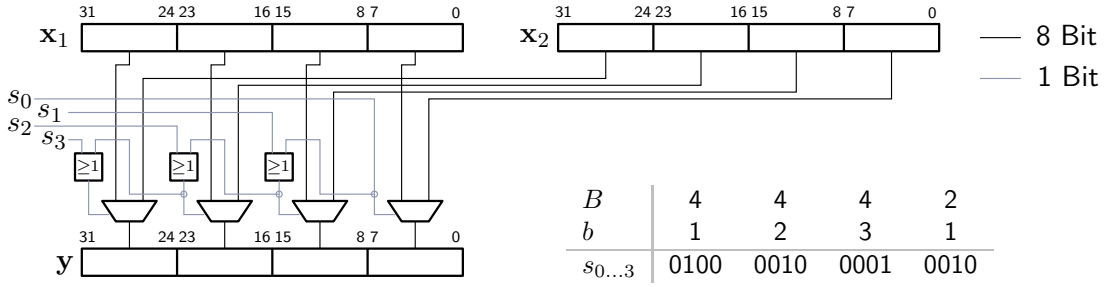


Abbildung 6.13: Schaltplan für Teilwortauswahlbefehl *ersetzeTWsinDvB*

sein.

Abbildung 6.12 zeigt eine Realisierung für einen solchen Doppelwortschiebebefehl für 32 Bit DvBs mit 4x8 Bit und 2x16 Bit Teilwortparallelität. Beide Formen der Teilwortparallelität lassen sich dabei mit einer Realisierung umsetzen.

Das Steuersignal 'c' für die Multiplexer lässt sich eindeutig aus der Verschiebung d und der Anzahl B der Teilwörter in den DvBs ermitteln. Bei einer Verschiebung um $d = 0$ Teilwörter, kann das Eingabe-DvB $\mathbf{x}_1$ direkt dem Ausgabe-DvB $\mathbf{y}$ zugewiesen werden.

Der Doppelwortschiebebefehl kann auch für die TWP-Operation $schiebeDvB(\mathbf{x}, s)$, welche zum Beispiel bei der Transformation der Eingabezuweisungen auf Seite 117 genutzt wird, verwendet werden. Ist der Schiebewert s positiv, so findet eine Linksverschiebung mittels schiebeTWinDvBs$(\mathbf{x}, \mathbf{0}, s)$ statt. Bei negativen Schiebewert s wird die Rechtsverschiebung mittels schiebeTWinDvBs$(\mathbf{0}, \mathbf{x}, B - s)$ realisiert.

6.6.2 Teilwortauswahlbefehl „ersetzeTWsinDvB“

Der Teilwortauswahlbefehl $\mathbf{y} =$ ersetzeTWsinDvB$(b, \mathbf{x}_1, \mathbf{x}_2)$ basiert auf der TWP-Operation $\mathbf{y} =$ ersetzeTWs$(b, \mathbf{x}_1, \mathbf{x}_2)$, siehe (6.99) auf Seite 119. Bei dieser Operation werden die oberen $B - b$ Teilwörter des Eingabe-DvBs $\mathbf{x}_2$ durch die oberen $B - b$ Teilwörter des Eingabe-DvBs $\mathbf{x}_1$ ersetzt. Abbildung 6.13 zeigt eine Realisierung für diesen Befehl.

Diese Realisierung beherrscht die Teilwortauswahl für DvBs mit 4x8 Bits und 2x16 Bit Teilwörter. Die Steuersignale $s_0, s_1, \ldots, s_3$ sind abhängig vom Parameter b und der Anzahl B der Teilwörter im DvB.

Dieser Befehl kann auch für die Operationen $ersetzeTW(b, \mathbf{x}_1, \mathbf{x}_2)$ und $F_{m,\underline{\mathcal{C}}^p_{T_s}}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p, \widehat{\boldsymbol{\kappa}}^P)$ genutzt werden. Die erste Operation erhalten wir durch zweifache Verwendung des $ersetzeTWsinDvB()$ Befehls:

$$\begin{aligned} s_{b,t}: &\quad \mathbf{y}_{b,t} = \text{ersetzeTWsinDvB}(b, \mathbf{x}_1, \mathbf{x}_2), \\ s_b: &\quad \mathbf{y}_b = \text{ersetzeTWsinDvB}(b+1, \mathbf{x}_2, \mathbf{y}_t). \end{aligned}$$

Aus folgenden Grund ist der zweite Befehl aber nicht notwendig: Wird die Operation $ersetzeTW()$ wie in Abschnitt 6.5.3 beschrieben verwendet, bei der ein DvB vom unteren zum oberen Teilwort systematisch aufgebaut wird, so werden die oberen $B-(b+1)$ Teilwörter, welche in Zuweisung s_b ersetzt werden, bei der nächstfolgenden Zuweisung $s_{b+1,t}$ erneut ersetzt.

Mit der Operation $\mathbf{y} = F_{m,\underline{\mathcal{C}}^p_{T_s}}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p, \widehat{\boldsymbol{\kappa}}^P)$, siehe (6.64) auf Seite 100, wird auch ein Zusammenbau eines DvBs aus den Teilwörtern mehrerer DvBs realisiert, wobei die Teilwörter sich bereits an der richtigen Position innerhalb der jeweiligen DvBs befinden. Bei einem systematischen Zusammenbau des DvBs $\mathbf{y}$ vom unteren zum oberen Teilwort kann wieder der Teilwortauswahlbefehl benutzt werden. Die Reihenfolge des Zusammenbaus ist für alle Iterationen $\widehat{\boldsymbol{\kappa}}^P \in \widehat{\mathcal{K}}^p_{\underline{\mathcal{C}}^P_{T_s}} \setminus \widehat{\mathcal{K}}^{p+1}$ gleich. Die Anzahl $b_{\text{eff}} = b_{\max} - b_{\min} + 1$ der zu ersetzenden Teilwörter und deren Anfangsposition b_{min} kann sich aber je Iteration $\widehat{\boldsymbol{\kappa}}^P$ unterscheiden. Bei der Verwendung des Teilwortauswahlbefehls muss für jedes DvB $\mathbf{x}_l$, $1 \leq l \leq p$ nur die Anfangsposition $b_{\min}$ ermittelt werden.

Beispiel 6.19
In dem Beispiel 6.8 ab Seite 100 entsteht durch die Partitionierung einer Zuweisung s mit vier Teilzuweisungen eine teilwortparallele Zuweisung s mit sechs Teilzuweisungen, von denen drei Teilzuweisungen eine Operation $F_{m,\underline{\mathcal{C}}^p_{T_s}}()$ besitzen. Für diese drei Operationen sollen nachfolgend die TWP-Befehle angegeben werden.

- $\mathbf{y}_{s_1}[\widehat{\boldsymbol{\kappa}}^P] = F^3_{m,1}(\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P)$:

  $$\begin{aligned} \mathbf{y}_{s_1,t}[\widehat{\boldsymbol{\kappa}}^P] &= ersetzeTWsinDvB(b_1, \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P]), \\ \mathbf{y}_{s_1}[\widehat{\boldsymbol{\kappa}}^P] &= ersetzeTWsinDvB(b_2, \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{s_1,t}[\widehat{\boldsymbol{\kappa}}^P]), \end{aligned}$$

  mit $b_1 = 1$ und $b_2 = 3$.

- $\mathbf{y}_{s_2}[\widehat{\boldsymbol{\kappa}}^P] = F^2_{m,1}(\mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P)$:

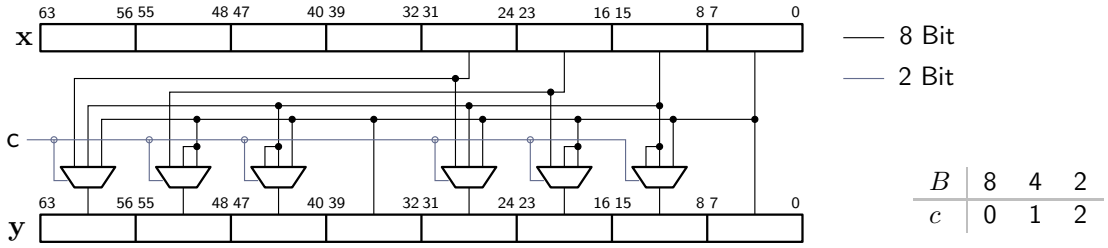
  $$\mathbf{y}_{s_2}[\widehat{\boldsymbol{\kappa}}^P] = ersetzeTWsinDvB(b, \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_1}}[\widehat{\boldsymbol{\kappa}}^P]),$$

  mit $b = 1$.

- $\mathbf{y}_{s_3}[\widehat{\boldsymbol{\kappa}}^P] = F^2_{m,4}(\mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P], \widehat{\boldsymbol{\kappa}}^P)$:

  $$\mathbf{y}_{s_3}[\widehat{\boldsymbol{\kappa}}^P] = ersetzeTWsinDvB(b, \mathbf{y}_{v_{s_3}}[\widehat{\boldsymbol{\kappa}}^P], \mathbf{y}_{v_{s_2}}[\widehat{\boldsymbol{\kappa}}^P]),$$

  mit $b = 8 - \widehat{\kappa}^P_2 - 4 \cdot \widehat{\kappa}^P_1$.



B	8	4	2
c	0	1	2

Abbildung 6.14: Schaltplan für *Datenverteilungsbefehl*

6.6.3 Datenverteilungsbefehl „verteileTWinDvB“

Der Datenverteilungsbefehl $\mathbf{y} = \text{verteileTWinDvB}(\mathbf{x})$ soll das Propagieren der Teilwörter innerhalb eines DvBs realisieren. Abbildung 6.14 zeigt eine Schaltung für einen solchen Befehl, wobei das zu propagierende Teilwort sich immer an Position $b = 0$ des DvBs befindet. Abhängig von der Anzahl B der Teilwörter im DvB wird mittels der Variablen c die Propagierung der Daten gesteuert. Der Eingabeparameter für diesen Befehl ist ein DvB.

Bei der teilwortparallelen Operation $\mathbf{y} = \text{verteileTW}(b, \mathbf{x})$ kann sich das zu propagierende Teilwort an jeder Stelle b des DvBs $\mathbf{x}$ befinden. Ist die Postion $b \neq 0$, so muss mit dem Befehl *schiebeTWinDvBs* das Teilwort an die erste Position geschoben werden, bevor der Datenverteilungsbefehl ausgeführt werden kann.

Die zweite teilwortparallele Operation $\mathbf{y} = \text{verteileTW}(x)$ hat als Eingabeparameter ein Teilwort. Dieses Teilwort kann nur ein Eingabeteilwort sein. Wird das Eingabeteilwort mit der Operation $\mathbf{y}_r = \text{leseTW}(x)$ gelesen, so befindet sich das Teilwort an der Position $b = 0$ des DvBs $\mathbf{y}_r$. Daran anschließend kann nun dieses Teilwort mittels des Datenverteilungsbefehls in dem DvB verteilt werden.

6.7 Existierende Strategien zur Teilwortparallelisierung

Seit der Implementierung von teilwortparallelen Befehlen in Funktionseinheiten gibt es Bestrebungen, die Fähigkeiten dieser Befehle automatisch auszunutzen, denn die verfügbaren Compiler konnten damals und können heute die durch die teilwortparallelen Befehle angebotene Parallelität nur unzureichend ausnutzen.

Im Jahr 2000 wurden gleich drei Abbildungsstrategien, der Musterübereinstimmungsansatz [MKC00], der Ansatz mit Hilfe der ganzzahligen linearen Optimierung [Leu00] und der Ansatz der Superwortebenenparallelität [LA00] veröffentlicht.

Bei dem Musterübereinstimmungsansatz (engl. Pattern Matching Approach) [MKC00] wird der Quelltext einer Anwendung nach Abschnitten durchsucht, welche ein bestimmtes Muster haben. Diese Abschnitte können dann durch teilwortparallele Ausdrücke ersetzt werden. Dieser Ansatz konzentiert sich auf die Parallelisierung von Schleifen, da

diese meist die rechenintensivsten Teile eines Programms sind. Durch den Musterübereinstimmungsansatz kann meist nur eine teilweise Parallelisierung erreicht werden, deren Beschleunigung aber beträchtlich sein kann. Die Muster und die zugehörigen teilwortparallelen Ausdrücke müssen von Hand erarbeitet werden.

In [Leu00] beschreibt Rainer Leupers, wie unter Verwendung von ganzzahliger linearer Optimierung die Teilwortparallelität für eine Anwendung ermittelt werden kann. Ausgehend vom Datenflussgraphen (DFG) dieser Anwendung werden bei der Optimierung die Operationen ausgewählt, die mit halber Wortbreite parallel ausgeführt werden können. Die Optimierung bezüglich Teilwortparallelität erfolgt dabei sowohl für Schleifen als auch für Basisblöcke. Ein Problem dieses Ansatzes ist die Größe des Optimierungsproblems und der damit verbundene hohe Zeitaufwand bei deren Lösung. Erschwerend kommt hinzu, dass mit steigender Anzahl an Teilwörtern im DvB die Komplexität stark zunimmt und damit die Lösung noch schwerer gefunden wird.

Eine große Beachtung fand der von Larsen vorgestellte Ansatz mit dem Namen Superwortebenenparallelität [LA00] (engl. Superword level parallelism, kurz: SLP). Bei diesem Ansatz werden ebenfalls die Zuweisungen von Basisblöcken als auch von Schleifen bezüglich Teilwortparallelität optimiert. Hierbei werden jeweils zwei Zuweisungen, welche gleichzeitig ausgeführt werden können, zu einem Paar zusammengefasst. Eine Zuweisung kann dabei in mehreren Paaren vorhanden sein. Anschließend werden die Paare, bei denen die obere Zuweisung des einen Paares die untere Zuweisung des anderen Paares ist, zu Paketen zusammengefasst bis die Anzahl der Zuweisungen in den Paketen der Anzahl an Teilwörtern in den DvBs entspricht. Diese Pakete stellen nun eine teilwortparallele Zuweisung dar und können dementsprechend durch eine teilwortparallele Operation ersetzt werden. Um in den Schleifen ausreichend unabhängige Zuweisungen für die Teilwortparallelisierung zu finden, werden die inneren Schleifen der Schleifenprogrammen um den Faktor B, der Anzahl der Teilwörter in einem DvB, aufgerollt. Anschließend kann mit der oben beschriebenen Methode innerhalb der Schleifen nach teilwortparallelen Zuweisungen gesucht werden.

Die Superwortebenenparallelität ist Grundlage für die Optimierungswerkzeuge zur Teilwortparallelisierung im GNU Compiler *GCC* [GCC]. Wie sich aber für praktische Beispiele zeigt, kann nur ein geringer Teil des Quelltextes einer Anwendung durch den Compiler automatisch parallelisiert werden.

In mehreren Arbeiten wurde der Ansatz der Superwortebenenparallelität erweitert. In dem Beitrag [TPP+05] lag der Schwerpunkt auf der erweiterten Untersuchung bezüglich Teilwortparallelität innerhalb von Schleifen. Hierzu wurde nicht nur die innere Schleife, sondern zusätzlich auch noch die zweite Schleife von innen aufgerollt. Damit konnte für diese beiden Raumrichtungen untersucht werden, in welcher Raumrichtung die Teilwortparallelisierung effektiver ist. Ein allgemeinerer Ansatz wird in [NZ08] verfolgt, bei dem mehrere bekannte Vektorisierungsmethoden für äußere Schleife an die Besonderheiten der Teilwortparallelität angepasst wurden. Die Arbeit von Shin [SHC05] erweitert den SLP Ansatz auf sogenannte datenabhängige *If-Else*-Anweisungen.

Ein weiteres Problem bei der Teilwortparallelisierung stellen verschachtelte Daten (z. B. die Variablen $y[2 \cdot i]$ und $y[2 \cdot i + 1]$ beim Viterbi-Algorithmus für den DVB-T Standard)

dar. In [NRZ06] wird hierfür eine Lösungmethode präsentiert.

Der IBM XL C und C++ Compiler [IBM][EOO+06], welcher unter anderem für den CELL-Prozessor genutzt wird, beinhaltet ebenfalls einen Teilwortparallelisierer. Dessen Grundmodell basiert auf der virtuellen Vektorrepräsentation [WEWZ05] für eine erfolgreiche Ermittlung von Teilwortparallelität im Quelltext. Virtuelle Register haben eine beliebige Länge und keine Ausrichtungsbeschränkungen. Die virtuellen Vektoren werden in den aufeinander folgenden Devirtualisierungsschritten [EWO04][WEW05] dann schrittweise an die physikalischen DvB-Register der Zielarchitektur angepasst. Teilwortparallelität kann mit dieser Methode sowohl aus Schleifen als auch aus Basisblöcken extrahiert werden.

Unsere Abbildungsstrategie konzentiert sich auf die Parallelisierung von Schleifen, welche als UItA dargestellt werden können. Die Teilwortparallelität kann ohne Beschränkungen für alle Raumrichtungen des Iterationsraums des Algorithmus überprüft werden. Die vorgestellten Transformation stellt sicher, dass der transformierte Algorithmus wieder ein UItA ist. Das erlaubt es uns, den transformierten Algorithmus auf ein Rechenfeld abzubilden, wie im nächsten Kapitel gezeigt wird.

Eine Analyse der Effizienz von Teilwortparallelität bezüglich Energieverbrauch und Leistung findet in [LRNB+07] statt. Hierbei werden sinnvolle Wortbreiten für die Teilwortparallelität ermittelt, welche zu einer geringer Verlustleistung und zu einen maximalen Durchsatz führen. Für eine freie Wahl der Wortbreite der Teilwörter wird hierbei auf die Softwarerealiserung der Teilwortparallelität [BDT06] zurückgegriffen.

In [TOdBM+05] wird ein Konzept zur Organsition der Teilwörter im Hintergrundspeicher vorgestellt, um eine energieeffizente Implementierung des Algorithmus auf der Zielarchitektur zu erhalten. Die Methode basiert auf dem klassischen Rechenfeldentwurf mit Ort-Zeitransformation und Partitionierung [Rao85, Eck01]. Dieses Konzept kann für solche Untersuchungen auf die in dieser Arbeit verwendete modifizierte Copartitionierung übertragen werden.

In [Lee00], [ML01] und [BSK+99] werden jeweils zusätzliche Packbefehle vorgestellt, um verschiedene Anwendungen beschleunigen zu können. Da es für verschiedene Architekturen unterschiedliche TWP-Befehle gibt, ist eine genaue Analyse dieser TWP-Befehle insbesondere der Packbefehle notwendig, um effizienten Code erzeugen zu können, wie auch in [CEV01] gezeigt wird. Alternativ wird in [HM03] eine neue Architektur mit dem Namen *Single Instructions Multiple Disjoint Data* ($\text{SIM}_\text{d}\text{D}$) vorgeschlagen, bei der mittels Zeiger ein flexibleres Lesen und Schreiben von Teilwörtern in ein Datenwort möglich ist und somit das aufwändige Datenpacken hinfällig wird. Diese Vereinfachung wird jedoch durch einen erhöhten Steueraufwand erkauft.

Offensichtlich ist eine effiziente Datenwiederverwendung bzw. die lokale Speicherung von wiederverwendbaren Daten entscheidend für die Beschleunigung der Berechnung, da die begrenzte Datenrate zwischen Speicher und Verarbeitungseinheit ein Hauptproblem bei der Programmbeschleunigung mittels SIMD-Befehlen ist [SDT01].

Kapitel 7

Abbildungsstrategie

Der neue Abbildungsfluss für parallele Rechenfelder mit der mehrstufigen modifizierten Copartitionierung als wesentlichen Bestandteil ist Basis für unsere Abbildungsstrategie zur Abbildung eines UItA auf ein Rechenfeld mit Parallelverarbeitung auf mehreren Ebenen. In Abbildung 7.1 ist diese Strategie graphisch dargestellt.

Ausgangspunkt der Abbildungsstrategie ist ein UItA, wie er in Kapitel 3 vorgestellt wurde. Ein solcher Algorithmus soll auf eine Zielarchitektur, ein Rechenfeld mit Parallelverarbeitung auf mehreren Ebenen, wie in Kapitel 2 präsentiert, abgebildet werden. Die Abbildung gliedert sich in sechs Schritte, wobei die ersten beiden Schritte zur Vorverarbeitung zusammengefasst werden und die anderen vier Schritte Teilschritte einer mehrstufigen modifizierten Copartitionierung sind.

Mit der Reindizierung des Algorithmus, siehe Abschnitt 4.1, wird die Lage und Form des Iterationsraums des Algorithmus modifiziert und es werden die Abhängigkeitsvektoren $\mathbf{d}(e)$ des Algorithmus in Länge und Richtung verändert. In Abschnitt 7.1.1 werden Strategien zur Bestimmung von Reindizierungen vorgestellt, die den Algorithmus so verändern, dass er bei der nachfolgenden mehrstufigen modifizierten Copartitionierung (MMC) möglichst effektiv auf die Zielarchitektur abgebildet wird. Die wichtigsten Ziele der MMC und damit auch der Reindizierung sind eine geringe Abarbeitungsdauer, wenige Packoperationen und kurze Datenkanäle zu erreichen. Diese Ziele können durch eine gute Reindizierung beeinflusst und mit MMC dann realisiert werden. Für die Reindizierung wurde eine teilautomatisierte Optimierung entwickelt, welche die schnelle Ermittlung guter Reindizierungen erlaubt.

Die Anpassung des UItA an die Algorithmenkonvention im Rahmen der Vorverarbeitung ist ein Umformulieren des Algorithmus, sodass der Algorithmus die in Abschnitt 3.4 beschriebenen Eigenschaften besitzt. Für einen möglichst einfachen Abbildungsprozess ist es von Vorteil, wenn die internen Berechnungzuweisungen in lokale Berechnungzuweisungen überführt werden, denn auf diese Weise können unnötige Datentransfers im Rechenfeld und zwischen Rechenfeld und Speicherarchitektur vermieden werden, wie in Abschnitt 7.1.2 gezeigt wird. Ein weiterer Vorteil ist, dass die lokale Berechnungszuwei-

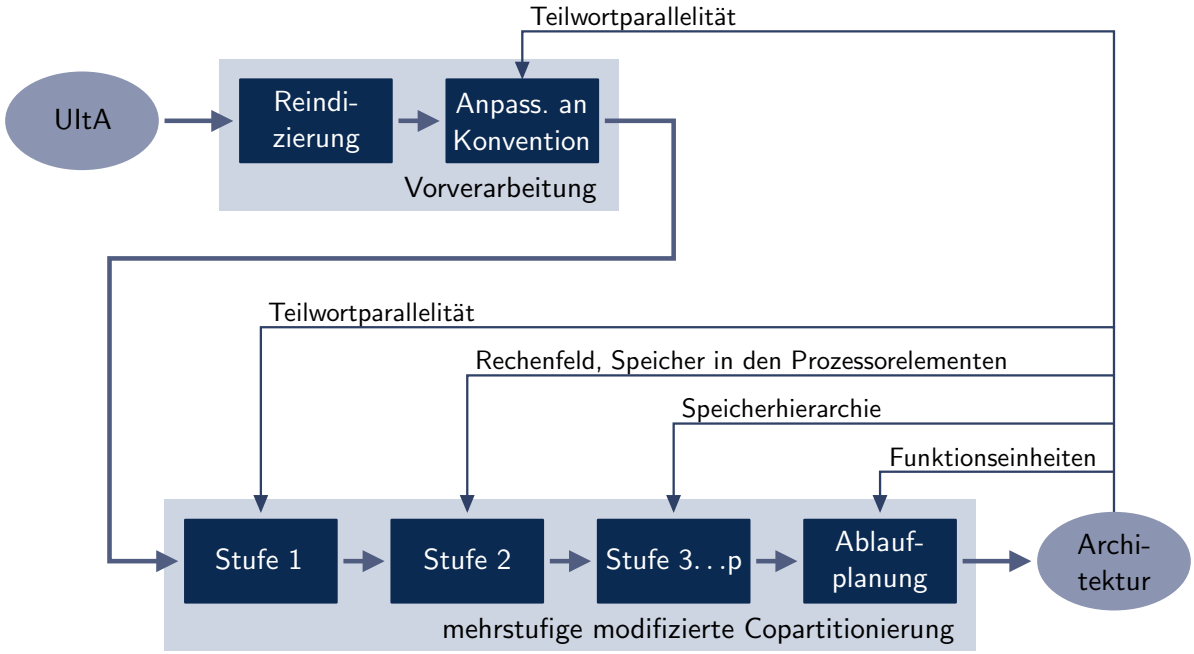


Abbildung 7.1: Abbildungsstrategie zur Abbildung eines UItA auf ein Rechenfeld mit Parallelverarbeitung auf mehreren Ebenen

sungen bei der mehrstufigen modifizierte Copartitionierung **nicht** in mehrere Teilzuweisungen zerfallen, da alle Abhängigkeitsvektoren dieser Zuweisungen $\mathbf{d}(e) = \mathbf{0}$ sind.

Die eigentliche Abbildung des Algorithmus auf das Rechenfeld wird mit der mehrstufigen modifizierten Copartitionierung erreicht. Die MMC wurde in Kapitel 5 vorgestellt. Mit jeder Stufe der mehrstufigen modifizierten Copartitionierung erfolgt eine stückweise Anpassung an die Zielarchitektur, wie in Abbildung 7.1 dargestellt wurde:

- **Stufe 1:** Die erste Stufe der mehrstufigen modifizierten Copartitionierung wird zur Ausnutzung der Teilwortparallelität genutzt. Im vorherigen Kapitel wurde ausführlich die Transformation eines Algorithmus von Einzeldatenoperationen zu einem Algorithmus mit teilwortparallelen Operationen beschrieben. Im Abschnitt 7.2.1 sollen nun Strategien für eine effiziente Ausnutzung der Teilwortparallelität vorgestellt werden. Ziel ist es, die Anzahl der Teilwörter in den DvBs voll auszunutzen und möglichst keine Packoperationen zu erzeugen. Für die erste Stufe der mehrstufigen modifizierten Copartitionierung gelten die Beschränkungen $\boldsymbol{\Theta}^{S_1} = \mathbf{E}$ und $\boldsymbol{\tau}^{\text{offs},1} = \mathbf{0}$, welche in Kapitel 6 vorgestellt und begründet wurden.

- **Stufe 2:** Das Hauptziel der zweiten Copartitionierungsstufe ist die Ausnutzung der Parallelität des Rechenfeldes. Zusätzlich soll der lokale Speicher in den Prozessorelementen optimal ausgenutzt werden, um den Energiebedarf aufgrund von Speicherzugriffen auf die Speicherarchitektur zu senken. Mit die Anpassung des Algorithmus an die lokalen Speicher wird eine Abarbeitungsreihenfolge für die Iterationen des Algorithmus festgelegt.

- **Stufe 3 bis p:** Mit der dritten Copartitionierungsstufe und den nachfolgenden Copartitionierungsstufen kann eine Anpassung des Datentransfers an die Größe der Zwischenspeicher L_l durch Veränderung der Abarbeitungsreihenfolge der Iterationen des Algorithmus vorgenommen werden. Hierbei wird mit jeder Copartitionierungsstufe k der Algorithmus an einen Zwischenspeicher L_{k-2} angepasst. Da auf dieser Ebene keine Parallelverarbeitung möglich ist, muss für die Partitionierungsmatrizen $\mathbf{\Theta}^{P_k} = \mathbf{E}$ gelten.

- **Ablaufplanung:** In jeder Copartitionierungsstufe wurde bisher mittels der Abfolgevektoren $\boldsymbol{\alpha}^{S_k}$ und $\widehat{\boldsymbol{\kappa}}^{P_k}$ die Abarbeitungsreihenfolge für die Iterationen des Algorithmus festgelegt. Die abschließende *Ablaufplanung* ermittelt nun den exakten Ablaufplan für den Algorithmus. Zusätzlich werden bei der Ablaufplanung die Funktionseinheiten $m \in \mathcal{M}$ in den Prozessorelementen an die Funktionen F_{s_t} des Algorithmus gebunden. Das heißt, es wird festgelegt, wann welche Funktionseinheit m die Funktion F_{s_t} der Teilzuweisung s_t abarbeitet. Diese Festlegung wird auch als lokale Ablaufplanung bezeichnet. Ziel ist die maximale Auslastung der Funktionseinheiten in den Prozessorelementen. Besitzen die Prozessorelemente mehrere Funktionseinheiten, so findet auf dieser Ebene ebenfalls eine Parallelverarbeitung statt.

Diese Reihenfolge der Anpassung des Algorithmus an die Architekturparameter wurde gewählt, damit die in der Zielarchitektur vorhandenen Möglichkeiten der Parallelverarbeitung maximal ausgenutzt werden können. Durch die Partitionierung von innen nach außen, kann die Partitionsgröße $\mathbf{\Theta}^{S_i}$ bzw. $\mathbf{\Theta}^{P_i}$ je Copartitionierungsstufe i frei gewählt werden. Dabei kann es aber geschehen, dass Partitionen unvollständig sind, was wiederum zu einer Verlängerung der Abarbeitungsdauer des Algorithmus führen kann. Im Abschnitt 7.2.3 wird für die Anpassung an die Speicherarchitektur eine Methode vorgestellt, mit der das Problem der unvollständigen Partitionen minimiert werden kann. Diese Methode ist prinzipiell auch auf die Partitionen der 1. und 2. Copartitionierungsstufe anwendbar ist.

Nachfolgend werden die Abbildungsschritte einzeln vorgestellt.

7.1 Vorverarbeitung

Die Vorverarbeitung besteht aus den zwei Schritten *Reindizierung* und *Anpassung an die Algorithmenkonvention*. Die Reindizierung, welche in Abschnitt 4.1 ab Seite 38 vorgestellt wurde, modifiziert die Iterationsräume $\mathcal{I}_s$ der Zuweisungen s des Algorithmus. Dadurch verändern sich die Abhängigkeitsvektoren $\mathbf{d}(e)$ der Datenabhängigkeiten $e \in \mathcal{E}$.

Ziel der Reindizierung ist es, die Länge der Abhängigkeitsvektoren zu reduzieren, denn kurze Abhängigkeitsvektoren führen zu kurzen Datentransfers (Datenkanälen) im Rechenfeld. Soll der Algorithmus teilwortparallelisiert werden, so wollen wir die Anzahl der Packoperationen minimieren, welche aufgrund der Partitionierung von Abhängigkeitsvektoren $\mathbf{d}(e)$ entstehen. Die Raumrichtung für die Teilwortparallelisierung kann dabei

vorgegeben werden oder es wird die Raumrichtung mit den wenigsten Packoperationen ausgewählt.

Die Reindizierung verändert weiterhin die Größe und die Form des gemeinsamen Iterationsraums $\mathcal{I}$. Wird ein gemeinsamer Iterationsraum $\widetilde{\mathcal{I}}$ gefunden, der bei der nachfolgenden mehrstufigen modifizierten Copartitionierung zu weniger Partitionen $\widehat{\boldsymbol{\kappa}}^{P_p} \in \widehat{\mathcal{K}}$ führt, so verringert sich im Allgemeinen die Abarbeitungszeit für diesen Algorithmus.

Mit der Anpassung an die Algorithmenkonvention wird der Algorithmus entsprechend der Konvention aus Abschnitt 3.4 umgewandelt. Hierbei kann durch Verwendung von ausschließlich lokalen Berechnungzuweisungen die Komplexität des Algorithmus in Bezug auf die nachfolgende mehrstufige modifizierte Copartitionierung merklich reduziert werden. Weiterhin können mehrfache Datentransfers gleicher Daten aufgedeckt und eliminiert werden.

7.1.1 Reindizierung

Die Reindizierung, vorgestellt in Abschnitt 4.1, erlaubt die Veränderung der Lage und der Form der Iterationsräume $\mathcal{I}_s$ der Zuweisungen $s \in \mathcal{S}$ des Algorithmus. Dadurch verändern sich ebenfalls die Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ des Algorithmus.

Mit der Reindizierung soll der Algorithmus nun so verändert werden, dass der reindizierte Algorithmus effizient auf die Zielarchitektur abgebildet werden kann. Zwei Punkte sollen bei der Reindizierung näher betrachtet werden:

(1) Die Veränderung der Größe, der Form und der Lage des gemeinsamen Iterationsraums $\mathcal{I}$, sodass die Anzahl der durch die MMC entstehenden Partitionen $\widehat{\boldsymbol{\kappa}}^{P_p} \in \widehat{\mathcal{K}}$ möglichst gering ist. Durch die Verringerung der Partitionen reduziert sich im Allgemeinen auch die Abarbeitungszeit für den Algorithmus.

(2) Die Veränderung der Abhängigkeitsvektoren $\mathbf{d}(e)$ des Algorithmus, sodass die veränderten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ zu einer effizienten Abbildung des Algorithmus auf die Zielarchitektur führen und die Ablaufplanung erleichtern. Zielstellungen für die veränderten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ sind:

- einheitliche Ausrichtung,
- geringe Länge und
- geringe Anzahl an Packoperationen bei nachfolgender Teilwortparallelisierung.

Die Reindizierungsfunktion $\rho_s(\mathbf{i})$, siehe Definition 4.1 auf Seite 38, bietet hierfür die folgenden Möglichkeiten:

(1) die zuweisungsbezogenen Reindizierungsvektoren $\mathbf{c}_s$

(2) die gemeinsame Reindizierungsmatrix $\mathbf{C}$.

Mit den zuweisungsbezogenen Reindizierungsvektoren $\mathbf{c}_s$ werden die Iterationsräume $\mathcal{I}_s$ zueinander und absolut (gemeinsam) verschoben und die gemeinsame Reindizierungsmatrix $\mathbf{C}$ verändert die Form des gemeinsamen Iterationsraums $\mathcal{I}$.

In den nächsten beiden Abschnitt werden die Optimierungsziele, welche durch die Anpassung der zuweisungsbezogenen Reindizierungsvektoren und der gemeinsamen Reindizierungsmatrix erreicht werden können, vorgestellt. Anschließend wird eine Strategie zur Durchführung der Reindizierung präsentiert.

Verschieben der Iterationsräume

Mit den Reindizierungsvektoren $\mathbf{c}_s$ der Reindizierungsfunktionen $\widetilde{\mathbf{i}} = \rho_s(\mathbf{i})$ können die Iterationen $\mathbf{i} \in \mathcal{I}_s$ der Zuweisungen $s \in \mathcal{S}$ zueinander oder gemeinsam verschoben werden. Mit dem Verschieben der Iterationsräume können mehrere Ziele verfolgt werden. Nachfolgend werden diese einzeln vorgestellt, wobei ohne Beschränkung der Allgemeinheit die Reindizierungsmatrix $\mathbf{C} = \mathbf{E}$ gesetzt wurde. Die ersten vier Ziele *Ausrichtung der Abhängigkeitsvektoren*, *minimale Anzahl an Packoperationen*, *minimale absolute Länge der Abhängigkeitsvektoren* und *Minimierung der Größe des gemeinsamen Iterationsraums* können mittels ganzzahliger linear Programmierung (kurz: GLP) verfolgt werden, wobei diese Ziele je nach Bedeutung nacheinander betrachtet werden, wie in Anhang D gezeigt wird. Das fünfte Ziel, die *Positionierung des gemeinsamen Iterationsraums im Koordinatensystem* kann unabhängig von der Lösung des GLP verfolgt werden, denn die ersten vier Ziele werden durch das relative Verschieben der Iterationsräume $\mathcal{I}_s$ zueinander erreicht, während das fünfte Ziel durch das gemeinsame Verschieben $\mathbf{c}_s = \mathbf{c}$ der Iterationsräume $\mathcal{I}_s$ erlangt wird.

Durch das Verschieben der Iterationsräume $\mathcal{I}_s$, verändern sich die Abhängigkeitsvektoren $\mathbf{d}(e)$ des Algorithmus. Für die reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ gilt:

$$\widetilde{\mathbf{d}}(e) = \widetilde{\mathbf{L}}^{-1}_{\sigma(e)}\big(\widetilde{\mathbf{l}}_{\sigma(e)} - \widetilde{\mathbf{r}}^{k}_{\sigma(e),\delta(e)}\big) \tag{7.1}$$

$$\widetilde{\mathbf{d}}(e) = \mathbf{L}^{-1}_{\sigma(e)}\mathbf{C}_{\sigma(e)}\big(\mathbf{l}_{\sigma(e)} - \mathbf{L}_{\sigma(e)}\mathbf{C}^{-1}_{\sigma(e)}\mathbf{c}_{\sigma(e)} - \mathbf{r}^{k}_{\sigma(e),\delta(e)} + \mathbf{R}^{k}_{\sigma(e),\delta(e)}\mathbf{C}^{-1}_{\delta(e)}\mathbf{c}_{\delta(e)}\big). \tag{7.2}$$

Da für die Reindizierungsmatrizen $\mathbf{C}_{\sigma(e)} = \mathbf{C}_{\delta(e)} = \mathbf{E}$ festgelegt wurde, folgt:

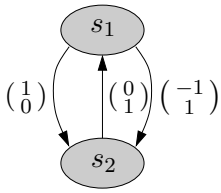
$$\widetilde{\mathbf{d}}(e) = \underbrace{\mathbf{L}^{-1}_{\sigma(e)}\big(\mathbf{l}_{\sigma(e)} - \mathbf{r}^{k}_{\sigma(e),\delta(e)}\big)}_{\mathbf{d}(e)} - \underbrace{\mathbf{L}^{-1}_{\sigma(e)}\mathbf{L}_{\sigma(e)}}_{\mathbf{E}}\mathbf{c}_{\sigma(e)} + \underbrace{\mathbf{L}^{-1}_{\sigma(e)}\mathbf{R}^{k}_{\sigma(e),\delta(e)}}_{\mathbf{E}}\mathbf{c}_{\delta(e)} \tag{7.3}$$

$$\widetilde{\mathbf{d}}(e) = \mathbf{d}(e) - \mathbf{c}_{\sigma(e)} + \mathbf{c}_{\delta(e)}. \tag{7.4}$$

Ausrichtung der Abhängigkeitsvektoren Für einen vereinfachten Ablaufplan sollen die reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ ausgerichtet sein.

s_1 :	$y_1 \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] = F_1(y_2 \left[\begin{smallmatrix} x \\ y-1 \end{smallmatrix}\right])$,	$\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] \in \mathcal{I}_1$
s_2 :	$y_2 \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] = F_2(y_1 \left[\begin{smallmatrix} x-1 \\ y \end{smallmatrix}\right],\ y_1 \left[\begin{smallmatrix} x+1 \\ y-1 \end{smallmatrix}\right])$,	$\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] \in \mathcal{I}_2$

(a) Algorithmus



(b) RAG

Abbildung 7.2: Algorithmus mit ungerichteten Abhängigkeitsvektoren

Definition 7.1 (Ausgerichtete Abhängigkeitsvektoren)
Die Abhängigkeitsvektoren $\mathbf{d}(e)$ *eines Algorithmus sind in der Raumrichtung* j *ausgerichtet, wenn für alle Datenabhängigkeiten* $e \in \mathcal{E}$ *gilt:*

$$\forall_{e \in \mathcal{E}} : d_j(e) \geq 0 \qquad \textit{oder} \qquad \forall_{e \in \mathcal{E}} : d_j(e) \leq 0. \tag{7.5}$$

Ziel der Ausrichtung der Abhängigkeitsvektoren ist es, die Iterationsräume $\mathcal{I}_s$ so anzuordnen, dass die reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ in so viel wie möglich Raumrichtungen ausgerichtet sind.

Ist eine Ausrichtung der Abhängigkeitsvektoren in **keiner** Raumrichtung möglich, so kann **kein** Abplaufplan bei der Copartitionierung des Algorithmus gefunden werden, denn in allen Raumrichtungen kommt es aufgrund der unausgerichteten Abhängigkeitsvektoren zu Konflikten bei der sequenziellen Abarbeitung. Somit folgt: In mindestens **einer** Raumrichtung müssen die Abhängigkeitsvektoren ausgerichtet sein.

Leider können nicht alle Abhängigkeitsvektoren für alle Raumrichtungen mit dem relativen Verschieben der Iterationsräume ausgerichtet werden, wie nachfolgendes Beispiel zeigt.

Beispiel 7.1 (Unausgerichtete Abhängigkeitsvektoren)
In Abbildung 7.2(a) ist ein Algorithmus dargestellt, dessen in x-Richtung entgegengesetzt gerichteten Abhängigkeitsvektoren $\mathbf{d}(e_1) = \binom{0}{1}$, $\mathbf{d}(e_2) = \binom{1}{0}$ *und* $\mathbf{d}(e_3) = \binom{-1}{1}$ *nicht durch das Verschieben der Iterationsräume zueinander beseitigt werden können. Verschieben wir Zuweisung* s_2 *mit* $\mathbf{c}_{s_2} = \binom{1}{0}$, *so wird der Konflikt zwischen* $\mathbf{d}(e_2)$ *und* $\mathbf{d}(e_3)$ *beseitigt, denn die reindizierten Abhängigkeitsvektoren lauten nun:* $\widetilde{\mathbf{d}}(e_2) = \binom{2}{0}$ *und* $\widetilde{\mathbf{d}}(e_3) = \binom{0}{1}$. *Doch nun zeigt die Datenabhängigkeit* e_1 *mit* $\widetilde{\mathbf{d}}(e_1) = \binom{-1}{1}$ *in die negative x-Richtung.*

Ein ähnliches Szenario ergibt sich, wenn wir Zuweisung s_2 *mit* $\mathbf{c}_{s_2} = \binom{-1}{0}$ *in die negative x-Richtung verschieben. Die Abhängigkeitsvektoren* $\widetilde{\mathbf{d}}(e_2) = \binom{0}{0}$ *und* $\widetilde{\mathbf{d}}(e_3) = \binom{-2}{1}$ *sind nun konfliktfrei, doch mit* $\widetilde{\mathbf{d}}(e_1) = \binom{1}{1}$ *weist die Datenabhängigkeit* e_1 *nun in die positive x-Richtung.*

Gibt es Raumrichtungen, in denen die Abhängigkeitsvektoren in verschiedene Richtungen weisen, so muss sichergestellt werden, dass keine konträren Abhängigkeitsvektoren im Algorithmus existieren:

$$\forall e_1, e_2 \in \mathcal{E} : \ \widetilde{\mathbf{d}}(e_1) \neq -k \cdot \widetilde{\mathbf{d}}(e_2) \neq \mathbf{0}, \quad k \in \mathbb{Q}_+. \tag{7.6}$$

Besitzt ein Algorithmus konträre Abhängigkeitsvektoren, so kann kein Ablaufplan für diesen Algorithmus unabhängig von der Copartitionierung gefunden werden, denn es gibt keine konfliktfreie Reihenfolge für eine sequenzielle Abarbeitung der Copartitionen.

Die Überprüfung der Ausrichtung der Abhängigkeitsvektoren kann für jede Raumrichtung einzeln erfolgen. Hierfür wird mittels eines GLP überprüft, ob alle Abhängigkeitsvektoren positiv oder negativ in einer Raumrichtung ausgerichtet werden können. Wenn eine der beiden Bedingungen erfüllt ist, so ist eine Ausrichtung der Abhängigkeitsvektoren für diese Raumrichtung möglich. Diese einzelne Überprüfung je Raumrichtung hat sich speziell für größere UItA als effektiver erwiesen, als die ganzheitliche Maximierung der Raumrichtungen in denen eine Ausrichtung möglich ist. Weiterhin können die ermittelten Resultate als Vorgaben für weitere Optimierungen verwendet werden.

Minimale Anzahl an Packoperationen Vor der Reindizierung ist meist schon bekannt, ob eine Teilwortparallelisierung des Algorithmus stattfinden soll. Auch lässt sich die Anzahl der Teilwörter B je DvB für die Teilwortparallelisierung zu diesem Zeitpunkt ermitteln.

Der Algorithmus soll nun so reindiziert werden, dass wir bei der nachfolgenden Teilwortparallelisierung eine geringe Anzahl an Packoperationen erhalten. Packoperationen können durch die Partitionierung der reindizierten Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$, durch die Partitionierung von Zuweisungen mit mehreren Teilzuweisungen und durch die Transformation der Ein- und Ausgabedatenzuweisungen entstehen, wie in Kapitel 6 gezeigt wurde. Die Anzahl der Packoperationen aufgrund der Partitionierung der Abhängigkeitsvektoren können durch die Reindizierung beeinflusst werden. Die Anzahl an Packoperationen aufgrund der Partitionierung von Zuweisungen mit mehreren Teilzuweisungen und aufgrund der Transformation der Ein- und Ausgabedatenzuweisungen lassen sich zu diesem Zeitpunkt noch nicht oder sehr schwer ermitteln. Doch zumindest für die Ein- und Ausgabedaten ist eine Abschätzung möglich, welche wiederum von der Raumrichtung j, in der die Teilwörter zu DvBs zusammengefasst werden sollen, abhängt. Diese Abschätzung ist sehr wichtig, denn die Anzahl an Packoperationen kann je nach Raumrichtung j sehr unterschiedlich sein. Diese Abschätzung hat somit entscheidenden Einfluss auf die Auswahl der Raumrichtung, in der die Teilwörter zu DvBs zusammengefasst werden sollen.

Die Bestimmung der Anzahl an Packoperationen, welche aufgrund der Partitionierung der Abhängigkeitsvektoren und aufgrund der Transformation der Ein- und Ausgabedatenzuweisungen entstehen, sollen nachfolgend vorgestellt werden.

Durch die Partitionierung der Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ entsteht eine Packoperation, wenn in der Raumrichtung j, in der die teilwortparallele Abarbeitung stattfinden soll, gilt: $d_j(e) \mod B \neq 0$. Die Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ lassen sich, wie (7.4) zeigt, durch das relative Verschieben der Iterationsräume verändern. Die Anzahl an Packoperationen p_j, welche aufgrund der Partitionierung der reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ bei Teilwortparallelisierung in Raumrichtung j entstehen, können wir

somit wie folgt bestimmen:

$$p_j^d = \sum_{e \in \mathcal{E}} |\text{sign}(\widetilde{d}_j(e) \mod B)|. \tag{7.7}$$

Für die Abschätzung der Anzahl der Packoperationen, welche aufgrund der Transformation der Ein- und Ausgabedatenzuweisungen entstehen, untersuchen wir für jede globale Variable $y_s[.], y_r[.] \in \mathcal{Y}_G$,

- ob die Variable eine Eingabe- oder Ausgabevariable ist, siehe Abschnitt 3.3.1 ab Seite 28,
- wie auf die globalen Daten zugegriffen werden kann (als Teilwörter oder als DvBs und wenn als DvBs, ob ausgerichtet oder unausgerichtet, siehe Abschnitt 2.4 ab Seite 16),
- ob bei einer Teilwortparallelisierung in Raumrichtung j die internen und globalen Daten gleichartig angeordnet sind, siehe Abschnitt *„Gleichartig angeordnete und ausgerichtete Daten“* auf Seite 115 ff. und
- ob je Iteration $\widehat{\boldsymbol{\kappa}}^P$ nur ein Teilwort gelesen oder geschrieben werden muss, siehe Abschnitt *„Haben die DvBs jeweils nur ein gültiges Teilwort?“* auf Seite 88 ff.

Abhängig von diesen Eigenschaften kann nun jeder globalen Variablen $y[.] \in \mathcal{Y}_G$ je Ein- oder Ausgabeteilzuweisung s_t und je Raumrichtung j ein Packaufwand $p_j^{e/a}(y[.], s_t)$ zugeordnet werden, der für jede Raumrichtung j in dem Parameter

$$p_j^{e/a} = \sum_{y[.] \in \mathcal{Y}_G} \sum_{s \in \mathcal{S}} \sum_{1 \le t \le T_s} p_j^{e/a}(y[.], s_t) \tag{7.8}$$

zusammengefasst wird.

Beispiel 7.2 (Bewertung der globalen Daten des FIR-Filters)
Das FIR-Filter besitzt drei globale Variablen $x[.]$, $b[.]$ und $y[.]$, deren Eigenschaften bezüglich Datentyp und Speicherzugriff in Tabelle 7.1(a) angegeben sind.

In den Beispielen 6.16 und 6.17 wurde die Teilwortparallelisierung der Ein- und Ausgabezuweisungen des FIR-Filters bereits vorgestellt, sodass wir bei der Bewertung der Ein- und Ausgabedaten auf diese Beschreibungen Bezug nehmen können.

Sind die Speicher- und Berechnungsdaten gleichartig angordnet, so können die Daten als DvBs effektiv gelesen und geschrieben werden. Werden je Iteration mehrere Teilwörter benötigt, so findet auch eine Parallelisierung bei der Dateneinlesen und -ausgabe statt ($p_{i_1}^{e/a}(y[.], s_t) = 0$). Werden je Iteration mehrere Teilwörter benötigt und sind die Speicher- und Berechnungsdaten nicht gleichartig angordnet, so müssen die Teilwörter einzeln gelesen werden und anschließend muss das DvB mittels Packoperation zusammengebaut werden ($p_{i_1}^{e/a}(y[.], s_t) = 4$). Wird nur ein Teilwort je Iteration gelesen, so muss

Tabelle 7.1: Bewertung der globalen Daten des FIR-Filters

(a) Datentyp und Speicherzugriff

	E/A	TW/DvB	Zugriff
$b[.]$	E	DvB	unausgerichtet
$x[.]$	E	DvB	unausgerichtet
$y[.]$	A	DvB	unausgerichtet

(b) Bei TWP in Raumrichtung i_1

		Anordnung	TWs je Iter.	$p_{i_1}^{e/a}$
$b[\begin{pmatrix}0&1\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{1,1}}$	nicht gleichart.	1	1
$x[\begin{pmatrix}1&-1\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{2,1}}$	gleichartig	1	1
$x[\begin{pmatrix}1&-1\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{2,2}}$	gleichartig	B	0
$y[\begin{pmatrix}1&0\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{4,1}}$	gleichartig	B	0
				2

(c) Bei TWP in Raumrichtung i_2

		Anordnung	TWs je Iter.	$p_{i_1}^{e/a}$
$b[\begin{pmatrix}0&1\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{1,1}}$	gleichartig	B	0
$x[\begin{pmatrix}1&-1\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{2,1}}$	nicht gleichart.	B	4
$x[\begin{pmatrix}1&-1\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{2,2}}$	nicht gleichart.	1	1
$y[\begin{pmatrix}1&0\end{pmatrix}\mathbf{i}]$	$\mathbf{i} \in \mathcal{I}_{s_{4,1}}$	nicht gleichart.	1	8
				13

das TW mit einer Packoperation in ein anderes DvB integriert werden ($p_{i_1}^{e/a}(y[.], s_t) = 1$). *Das Schreiben der y-Daten ist bei Teilwortparallelisierung in Raumrichtung* i_1 *sehr aufwändig* ($p_{i_1}^{e/a}(y[.], s_t) = 8$), *wie in Beispiel 6.17 gezeigt wurde.*

In den Tabellen 7.1(b) und 7.1(c) sind die Bewertungen der Ein- und Ausgabedaten zusammengefasst. In der Summe ergibt sich für das Beispiel des FIR-Filters, dass eine Teilwortparallelisierung bezüglich der Ein- und Ausgabedaten in Raumrichtung i_1 *am günstigsten ist.*

Die Gesamtzahl an Packoperationen p_j bei Teilwortparallelisierung in Raumrichtung j können wir somit wie folgt abschätzen:

$$p_j = p_j^d + p_j^{e/a}. \tag{7.9}$$

Soll eine Reindizierung bestimmt werden, die zu einer mininalen Anzahl an Packoperationen führt, so bestimmen wir das Minimum p an Packoperationen bezüglich aller Raumrichtungen j, $1 \leq j \leq n$:

$$p = \min_{1 \leq j \leq n} p_j. \tag{7.10}$$

Auf diese Weise wird sowohl die minimale Anzahl p an Packoperationen bestimmt, als auch die Raumrichtung j, in der diese Anzahl erreicht wird, ermittelt. Diese Raumrichtung wird der mehrstufigen modifizierten Copartitionierung für die ersten Stufe der Copartitionierung mitgeteilt.

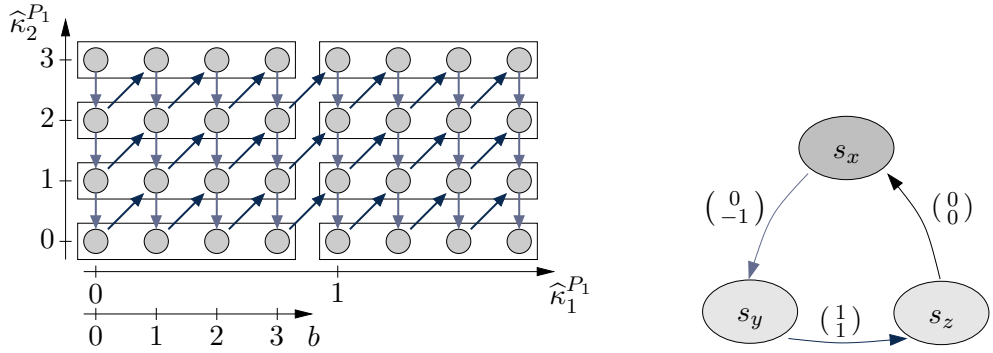
Bei der Auswahl der Raumrichtung j, in der Teilwortparallelität genutzt werden soll, muss noch eine Einschränkung beachtet werden.

Satz 7.1 (Kreise im reduzierten Abhängigkeitsgraph)
Besitzt der reduzierte Abhängigkeitsgraph eines Algorithmus einen Kreis $\mathcal{W}^\circ$, *siehe Anhang B.1, mit den beiden Eigenschaften*

- *Die Summe der Abhängigkeitsvektoren* $\mathbf{d}^\Sigma = \sum_{e \in \mathcal{W}^\circ} \mathbf{d}(e)$ *auf diesem Kreis ist ein Vektor mit* ***einem*** *Nicht-Null-Element, wobei das Nicht-Null-Element* d_j^Σ *ist und für dieses gilt:* $|d_j^\Sigma| < B$,
- *Ein Knoten* v *auf dem Kreis repräsentiert eine Berechnungszuweisung,*

so ist eine teilwortparallele Abarbeitung in der Raumrichtung j *nicht möglich.*

Beweis 7.1 (Kreise im reduzierten Abhängigkeitsgraph)
Würde man einen Iterationsraum $\mathcal{I}$ *in der Raumrichtung* j *teilwortparallelisieren, in der ein Kreis* $\mathcal{W}^\circ$ *mit den oben genannten Eigenschaften existiert, so entsteht eine teilwortparallele Zuweisung* $\mathbf{x}[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{F}_x(\ldots)$, *mit den linksseitigen DvBs* $\mathbf{x}[\widehat{\boldsymbol{\kappa}}^{P_1}]$. *Die DvBs* $\mathbf{x}[\widehat{\boldsymbol{\kappa}}^{P_1}]$ *dieser Zuweisung besitzen mindestens ein Teilwort* $x[b, \widehat{\boldsymbol{\kappa}}^{P_1}]$, *welches für seine Bestimmung das Teilwort* $x[b - d_j^\Sigma, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *aus dem selben DvB benötigt. Somit können*



(a) partitionierter Iterationsraum (b) RDG

Abbildung 7.3: Kreise im RDG

die beide Teilwörter $x[b, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *und* $x[b - d_j^{\Sigma}, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *nicht gleichzeitig ermittelt werden. Die gleichzeitige Berechnung der Teilwörter eines DvBs ist aber eine zentrale Eigenschaft der Teilwortparallelität, sie resultiert aus der Bedingung* $\boldsymbol{\tau}^{offs,1} = \mathbf{0}$.

Beispiel 7.3 (Kreise im reduzierten Abhängigkeitsgraph)
Abbildung 7.3 illustriert das Problem anhand eines Kreises mit drei Zuweisungen, wobei Zuweisung s_x *eine Berechnungszuweisung ist:*

$$s_x: \quad x[\mathbf{i}] = F_x(\ldots, z[\mathbf{i}], \ldots), \qquad \mathbf{i} \in \mathcal{I}_x, \quad (7.11)$$

$$s_y: \quad y[\mathbf{i}] = F_y(\ldots, x[\mathbf{i} - \tbinom{0}{-1}], \ldots), \qquad \mathbf{i} \in \mathcal{I}_y, \quad (7.12)$$

$$s_z: \quad z[\mathbf{i}] = F_z(\ldots, y[\mathbf{i} - \tbinom{1}{1}], \ldots), \qquad \mathbf{i} \in \mathcal{I}_z. \quad (7.13)$$

Die Summe der Abhängigkeitsvektoren auf dem Kreis beträgt $\mathbf{d}^{\Sigma} = \binom{1}{0}$. *Bei einer Partitionierung in Raumrichtung* $j = 1$ *erhalten wir den partitionierten Iterationsraum, wie in Abbildung 7.3(a) dargestellt.*

Durch das Ersetzen der rechtsseitigen Variable $z[\mathbf{i}]$ *in (7.11) durch (7.13) und dem anschließenden Ersetzen der Variable* $y[\mathbf{i} - \binom{1}{1}]$ *durch (7.12) erhalten wir:*

$$x[\mathbf{i}] = F_x(\ldots, F_z(\ldots, F_y(\ldots, x[\mathbf{i} - \tbinom{1}{0}], \ldots), \ldots), \ldots). \quad (7.14)$$

Aus dieser Gleichung wird offensichtlich, dass Variable $x[\mathbf{i}]$ *von Variable* $x[\mathbf{i} - \binom{1}{0}]$ *bzw. nach der Teilwortpartitionierung in Raumrichtung* j *Teilwort* $x[b, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *von Teilwort* $x[b-1, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *abhängig ist. Da Zuweisung* s_x *eine Berechnungszuweisung mit Latenz* $l_{m(s_x)} > 0$ *ist, können die beiden Teilwörter* $x[b, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *und* $x[b-1, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *nicht gleichzeitig ausgeführt werden, denn die Berechnung des Teilworts* $x[b, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *kann erst beginnen, wenn das Teilwort* $x[b-1, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *zur Verfügung steht. Die gleichzeitige Berechnung der Teilwörter* $x[b, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *und* $x[b-1, \widehat{\boldsymbol{\kappa}}^{P_1}]$ *ist aber eine notwendige Bedingung für die Teilwortparallelität.*

Ist das Nicht-Null-Element $d_j^{\Sigma} \geq B$, so befinden sich die voneinander abhängigen Variablen $x[\mathbf{i}]$ und $x[\mathbf{i} - \mathbf{d}^{\Sigma}]$ nach der Teilwortpartitionierung immer in unterschiedlichen Partitionen (DvBs) und können damit, wie notwendig, nacheinander abgearbeitet werden.

Die Teilwörter eines DvBs sind somit unabhängig voneinander und lassen sich parallel ermitteln.

Ist keine der Zuweisungen in einem Kreis eine Berechnungszuweisung, so kann der Kreis durch Elimination von Variablen in eine Schlinge $\mathcal{W}^s = \{e_0\}$ umgewandelt werden, bei dem der Abhängigkeitsvektor $\mathbf{d}(e_0) = \mathbf{d}^\Sigma$ ist. Die Zuweisung $\sigma(e_0) = \delta(e_0)$ ist dann eine Propagierungszuweisung. Wie in den Abschnitten 6.4.2 und 6.5.4 gezeigt wurde, ist die parallele Bestimmung von Variablen einer Propagierungszuweisung möglich.

Satz 7.2
Die Summe $\widetilde{\mathbf{d}}^\Sigma = \sum_{e\in\mathcal{W}^\circ} \widetilde{\mathbf{d}}(e)$ *der reduzierten Abhängigkeitsvektoren* $\widetilde{\mathbf{d}}(e) = \mathbf{d}(e) + \mathbf{c}_{\delta(e)} - \mathbf{c}_{\sigma(e)}$ *auf einem Kreis* $\mathcal{W}^\circ = \{e_0, e_1, \ldots, e_{m-1}\}$ *des RDG des Algorithmus ist unabhängig von der zuweisungsabhängigen Verschiebung* $\mathbf{c}_s$.

Beweis 7.2
Eine zuweisungsabhängigen Verschiebung $\mathbf{c}_v$ *der Knoten (Zuweisungen)* $v = \sigma(e)$, $e \in \mathcal{W}^\circ$ *führt zu den reindizierten Abhängigkeitsvektoren:*

$$\widetilde{\mathbf{d}}(e) = \mathbf{d}(e) + \mathbf{c}_{\delta(e)} - \mathbf{c}_{\sigma(e)}. \tag{7.15}$$

Für die Summe der Abhängigkeitsvektoren $\widetilde{\mathbf{d}}^\Sigma$ *gilt damit:*

$$\widetilde{\mathbf{d}}^\Sigma = \sum_{e\in\mathcal{W}^\circ} \widetilde{\mathbf{d}}(e) = \sum_{e\in\mathcal{W}^\circ} \mathbf{d}(e) + \mathbf{c}_{\delta(e)} - \mathbf{c}_{\sigma(e)}. \tag{7.16}$$

Jede Kante e *des Kreises* $\mathcal{W}^\circ$ *verursacht eine Addition von* $\mathbf{c}_{\delta(e)}$ *und eine Subtraktion von* $\mathbf{c}_{\sigma(e)}$. *Da jeder Knoten* v *des Kreises* $\mathcal{W}^\circ$ *genau eine zulaufende Kante* $e_1 \in \mathcal{W}^\circ$ $(\delta(e_1) = v)$ *und genau eine weglaufende Kante* $e_2 \in \mathcal{W}^\circ$ $(\sigma(e_2) = v)$ *besitzt, wird die zuweisungsabhängige Verschiebung* $\mathbf{c}_v$ *bei der Bestimmung von* $\widetilde{\mathbf{d}}^\Sigma$ *genau einmal hinzugefügt* $(\mathbf{c}_{\delta(e_1)} = \mathbf{c}_v)$ *und genau einmal abgezogen* $(\mathbf{c}_{\sigma(e_2)} = \mathbf{c}_v)$. *Somit folgt:*

$$\widetilde{\mathbf{d}}^\Sigma = \sum_{e\in\mathcal{W}^\circ} \widetilde{\mathbf{d}}(e) = \sum_{e\in\mathcal{W}^\circ} \mathbf{d}(e) = \mathbf{d}^\Sigma. \tag{7.17}$$

Somit kann im Ausgangsalgorithmus untersucht werden, ob es Kreise $\mathcal{W}^\circ$ mit den in Satz 7.1 genannten Bedingungen gibt. Ist das der Fall, so kann in diesen Raumrichtungen der Algorithmus nicht teilwortparallelisiert werden.

Minimale absolute Länge der Abhängigkeitsvektoren Die Länge der reindizierten Abhängigkeitsvektoren soll minimiert werden, um Datentransfers im Rechenfeld zu verringern bzw. lokale Datenwiederverwendung zu erhöhen [SM06]. Als Maß für die Länge der Abhängigkeitsvektoren wählen wir die Manhattan-Distanz (MH-Distanz) m:

$$m = \sum_{e\in\mathcal{E}} \sum_{j=1}^{n} |\widetilde{d}_j(e)|. \tag{7.18}$$

Tabelle 7.2: Manhattan-Distanzanteile $m_j(e)$ der Abhängigkeitsvektoren $\widetilde{d}_j(e)$ bei Teilwortparallelisierung

$m_j(e)$	mit Packoperation aufgrund $\widetilde{\mathbf{d}}(e)$	ohne Packoperation aufgrund $\widetilde{\mathbf{d}}(e)$
TWP in j	$2\lfloor\frac{\lvert\widetilde{d}_j(e)\rvert}{B}\rfloor + 1$	$\lfloor\frac{\lvert\widetilde{d}_j(e)\rvert}{B}\rfloor$
keine TWP in j	$2\lvert\widetilde{d}_j(e)\rvert$	$\lvert\widetilde{d}_j(e)\rvert$

Soll der Algorithmus später teilwortparallelisiert werden, so bestimmen wir die Manhattan-Distanz der äußeren Teilvektoren, welche durch die Partitionierung der Abhängigkeitsvektoren entstehen. Die äußeren Teilvektoren beschreiben den Transfer der DvBs im transformierten Iterationsraum, wie in Abschnitt 6.2 gezeigt wurde und sind damit das relevante Maß für einen teilwortparallelen Algorithmus.

Die Manhattan-Distanzanteile $m_j(e)$ der reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ sind abhängig von der Raumrichtung j, in der die Teilwortparallelisierung stattfinden soll, und davon, ob der reindizierte Abhängigkeitsvektor eine Packoperation verursacht. Die Manhattan-Distanzanteile $m_j(e)$ abhängig von diesen beiden Bedingungen sind in Tabelle 7.2 angegeben.

Die Manhattan-Distanz bei Teilwortparallelisierung erhalten wir somit mit

$$m = \sum_{e\in\mathcal{E}}\sum_{j=1}^{n} m_j(e). \tag{7.19}$$

Minimierung der Größe des gemeinsamen Iterationsraums Durch das Verschieben der Iterationsräume $\mathcal{I}_s$ zueinander kann der gemeinsame Iterationsraum $\mathcal{I}$ minimiert werden. Für die weiteren Abbildungsschritte ist es vor allem von Bedeutung, dass der den gemeinsamen Iterationsraum $\mathcal{I}$ umschließende Hyperquarder $\mathcal{I}^{\square}$ minimiert wird.

Das Volumen $V(\widetilde{\mathcal{I}}^{\square})$ des den reindizierten Iterationsraum $\widetilde{\mathcal{I}}$ umschließenden Hyperquarders $\widetilde{\mathcal{I}}^{\square}$ kann über die Ecken $\mathbf{v} \in \mathcal{V}$ der Iterationsräume $\mathcal{I}_s$ des Ausgangsalgorithmus leicht ermittelt werden, wie in Anhang D.4 gezeigt wird.

Für die Minimierung des Volumens in einem GLP wurden zwei Strategien entwickelt, da eine Multiplikation $V(\widetilde{\mathcal{I}}^{\square}) = \prod_{j=1}^{n} \widetilde{v}_{\mathrm{diff},j}$ der Ausdehnungen $\widetilde{v}_{\mathrm{diff},j}$ des Hyperquarders $\widetilde{\mathcal{I}}^{\square}$ in jeder Raumrichtung mit einer linearen Beschreibung nicht möglich ist:

- Die Ausdehnungen $\widetilde{v}_{\mathrm{diff},j}$ des Hyperquarders $\widetilde{\mathcal{I}}^{\square}$ in jeder Raumrichtung werden aufsummiert.
- Die Ausdehnungen $\widetilde{v}_{\mathrm{diff},j}$ des Hyperquarders $\widetilde{\mathcal{I}}^{\square}$ in jeder Raumrichtung werden mit dem Inversen der Ausdehnung $v_{\mathrm{diff},j}$ des Hyperquarder $\mathcal{I}^{\square}$ gewichtet, bevor die Werte aufsummiert werden.

Soll der Algorithmus auf eine Architektur mit Teilwortparallelität abgebildet werden, so wird das Volumen $V(\widehat{\mathcal{K}}^{\square})$ des den Partitionsraum $\widehat{\mathcal{K}}$ umschließenden Hyperquaders $\widehat{\mathcal{K}}^{\square}$ minimiert. Für diesen Fall beträgt $\widetilde{v}_{\text{diff},j}$ für die Raumrichtung j in der die Teilwörter zu DvBs zusammengefasst werden sollen $\widetilde{v}_{\text{diff},j} = \left\lceil \frac{\widetilde{v}_{\max,j} - \widetilde{v}_{\min,j} + 1}{B} \right\rceil$. Der Partitionsraum $\widehat{\mathcal{K}}$ ist der Iterationsraum der DvBs des teilwortparallelen Algorithmus.

Position des gemeinsamen Iterationsraums im Koordinatensystem Der gemeinsame Iterationsraum $\mathcal{I} \in \mathbb{Z}^n$ eines Algorithmus kann beliebig im Raum $\mathbb{Z}^n$ platziert werden. Ein Verschieben des gemeinsamen Iterationsraums ohne Veränderung der vorherigen Optimierungsziele wird durch eine gemeinsame Verschiebung mit $\mathbf{c}_s = \mathbf{c}$ erreicht.

Für eine effektive Partitionierung ist es wünscheswert, wenn je Raumrichtung die des zeitlichen Ablaufplanes entsprechend ersten Partitionen möglichst vollständig sind. Zum einen können dadurch teilbesetzte Partitionen am Anfang **und** am Ende einer Raumrichtung vermieden werden, die möglicherweise zu zusätzlichen Partitionen führen würden. Zum anderen kann durch die vollständigen, zeitlich ersten Partitionen die Dateneingabe vereinfacht werden.

Zum Zeitpunkt der Reindizierung stehen aber noch keine Informationen über die Abarbeitungsreihenfolge und Richtung zur Verfügung. Aus den Abhängigkeitsvektoren $\mathbf{d}(e)$ lassen sich aber zulässige Abarbeitungsrichtungen für jede Raumrichtung j ermitteln. Die Reihenfolge der Abarbeitung der Raumrichtungen ist für die Positionierung des Iterationsraums im Koordinatensystem uninteressant.

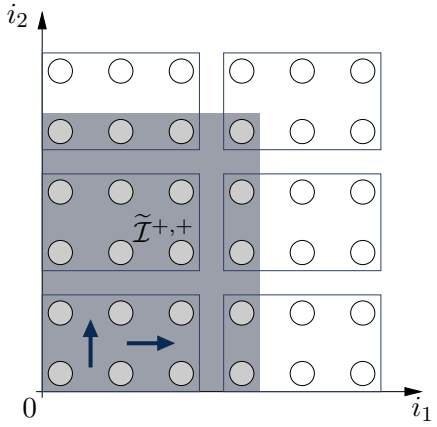
Zur Bestimmung der zulässigen Abarbeitungsrichtungen je Raumrichtung j ermitteln wir die Anzahl der Abhängigkeitsvektoren, die in dieser Raumrichtung positiv b_j^+ oder negativ b_j^- sind:

$$b_j^+ = \sum_{e \in \mathcal{E}} b_j^+(e), \quad b_j^+(e) = \begin{cases} 1, & \text{wenn } d_j(e) > 0, \\ 0, & \text{sonst}, \end{cases} \tag{7.20}$$

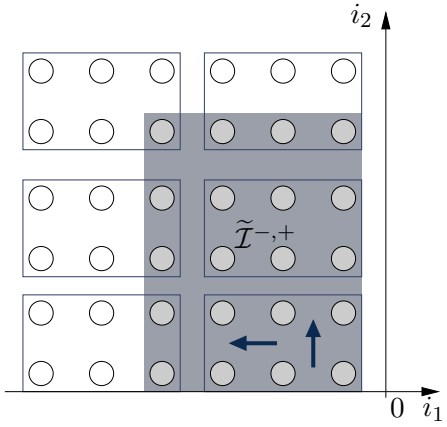
$$b_j^- = \sum_{e \in \mathcal{E}} b_j^-(e), \quad b_j^-(e) = \begin{cases} 1, & \text{wenn } d_j(e) < 0, \\ 0, & \text{sonst}. \end{cases} \tag{7.21}$$

Gilt $b_j^+ \geq b_j^-$, so gehen wir von einer positive Abarbeitungsrichtung aus. Die untere Begrenzung des den Iterationsraum $\mathcal{I}$ umschließenden Hyperquaders $\mathcal{I}^{\square}$ wird auf die Null-Linie ($i_j = 0$) des Koordinatensystems geschoben ($\widetilde{a}_{0,j}^{\sqcup} = 0$). Im anderen Fall, der negativen Abarbeitungsrichtung ($b_j^+ < b_j^-$), wird die obere Begrenzung des Hyperquaders $\mathcal{I}^{\square}$ auf die Minus-Eins-Linie ($i_j = -1$) des Koordinatensystems gelegt ($\widetilde{a}_{0,j}^{\sqcap} = -1$), um eine vollständige erste Partition zu erhalten. Somit ergibt sich für den gemeinsamen Reindizierungsvektor $\mathbf{c}_s = \mathbf{c}$:

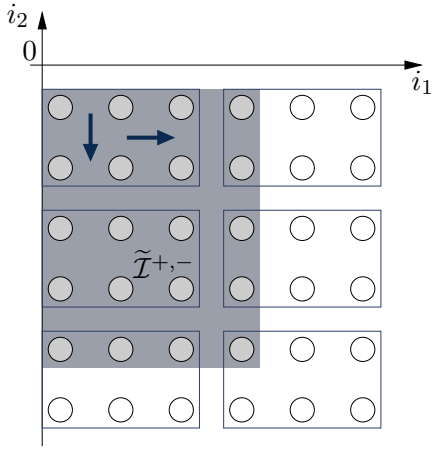
$$c_j = \begin{cases} -a_{0,j}^{\sqcup} & \text{wenn} \quad b_j^+ \geq b_j^- \\ -a_{0,j}^{\sqcap} - 1 & \text{sonst}. \end{cases} \tag{7.22}$$



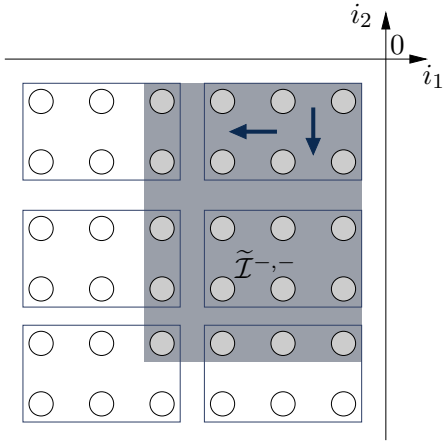
(a) positive Abarbeitung in i_1 und positive Abarbeitung i_2



(b) negative Abarbeitung in i_1 und positive Abarbeitung in i_2



(c) positive Abarbeitung in i_1 und negative Abarbeitung in i_2



(d) negative Abarbeitung in i_1 und negative Abarbeitung i_2

Abbildung 7.4: Lage des Iterationsraums im Koordinatensystem bei verschiedenen Abarbeitungsrichtungen

Beispiel 7.4 (Positionierung des Iterationsraums im Koordinatensystem) *Betrachten wir den Iterationsraum* $\mathcal{I} = \{\mathbf{i} = \binom{i_1}{i_2} \mid 0 \leq i_1 < 4 \wedge 0 \leq i_2 < 5\}$. *Dieser Iterationsraum soll abhängig von der Abarbeitungsrichtung (positiv oder negativ) je Raumrichtung im Koordinatensystem positioniert werden. Abbildung 7.4 zeigt die vier Möglichkeiten der Positionierung abhängig von der Abarbeitungsrichtung je Raumrichtung, wobei die Abarbeitungsrichtungen je verschobenen Iterationsraum* $\widetilde{\mathcal{I}}^{+,+}$, $\widetilde{\mathcal{I}}^{+,-}$, $\widetilde{\mathcal{I}}^{-,+}$ *und* $\widetilde{\mathcal{I}}^{-,-}$ *durch Pfeile im jeweiligen Iterationsraum dargestellt sind.*

Durch die Verschiebung des Iterationsraums wurde erreicht, dass die zeitlich erste Partition vollständig ist.

Ein in dieser Weise angeordneter gemeinsamer Iterationsraum $\mathcal{I}$ führt bei der nachfolgenden Partitionierung im Allgemeinen zu einer minimalen Anzahl an äußeren Partitionen $|\widehat{\mathcal{K}}|$. Es existieren aber auch Algorithmen, bei denen aufgrund der Form des

Iterationsraums $\mathcal{I} \neq \mathcal{I}^{\square}$ und der Partitionsparameter $\mathbf{\Theta}^{S_i}$ und $\mathbf{\Theta}^{P_i}$ ein derart verschobener gemeinsamer Iterationsraum nicht zur besten Lösung führt. Ein Beispiel hierfür ist der dreieckige Iterationsraum $\mathcal{I}$ aus Beispiel 4.2 von Seite 41 mit der Partitionierung $\mathbf{\Theta} = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 2 \end{smallmatrix}\right)$. Würde dieser Iterationsraum so verschoben, dass die linke untere Ecke im Koordinatenursprung wäre, so benötigte man vier äußere Partitionen zur Überdeckung des Iterationsraums. Bei einer Positionierung der linken unteren Ecke in Iteration $\mathbf{i} = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ erhielten wir hingegen nur drei äußeren Partitionen für die Überdeckung.

Da zum Zeitpunkt der Reindizierung die Partitionierungsparameter $\mathbf{\Theta}^{S_i}$ und $\mathbf{\Theta}^{P_i}$ meist nicht bekannt sind, wenden wir die für den allgemeinen Fall günstigte Lösung an.

Zusammenfassung Wie in den vorherigen Abschnitten gezeigt wurde, können mehrere Ziele mit der Festlegung der zuweisungsabhängigen Reindizierungsvektoren $\mathbf{c}_s$ verfolgt werden. Für die Verbesserung des Algorithmus durch zuweisungsabhängige Reindizierungsvektoren $\mathbf{c}_s$ wurde ein Programm entwickelt, welches mittels vier ganzzahliger linearer Programm (GLPs) die vier Ziele:

(1) *Ausrichtung der Abhängigkeitsvektoren*,

(2) *Minimale Anzahl an Packoperationen*,

(3) *Minimale absolute Länge der Abhängigkeitsvektoren* und

(4) *Minimierung der Größe des gemeinsamen Iterationsraums*

in der angegebenen Reihenfolge verfolgt. Das erste Optimierungsziel maximiert die Möglichkeiten der Parallelisierung. Mit dem zweiten Ziel wird die Anzahl an zusätzlichen Packoperationen minimiert und damit der Rechenaufwand in einer Iteration minimiert. Anschließend wird die minimale Manhattan-Distanz der partitionierten Abhängigkeitsvektoren ermittelt, welche zu kurzen Datentransfers und maximaler lokaler Datenwiederverwendung führt. Zum Abschluss versucht des Programm die Größe des gemeinsamen Iterationsraums zu reduzieren, womit die spätere Abarbeitungszeit minimiert werden soll. Die gefundenen Resultate eines GLPs sind dabei immer Beschränkungen für die nachfolgenden GLPs. In Anhang D ist das gesamte Programm mit den vier GLPs beschrieben.

Tabelle 7.3 zeigt die Ergebnisse und Lösungszeiten[1] der Optimierung für ausgewählte Algorithmen. Für die Untersuchung wurden die folgenden Parameter gewählt:

- Teilwortparallelität mit vier Teilwörtern,
- keine Vorgaben für die Auswahl der Raumrichtung für die TWP,
- keine Bewertung der Ein- und Ausgabedaten bezüglich TWP,
- keine Vorgaben für die Datentransferrichtung und

[1] Intel Core2 Duo E6750 @ 2.66GHz

Tabelle 7.3: Lösung des GLPs zur Optimierung der Reindizierungsvektoren $\mathbf{c}_s$ für verschiedene Algorithmen

Name	Alg.	n	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{E}\rvert$	Lösungszeit	r	Packop.	MH-Dist.	$V(\widehat{\mathcal{K}}^{\square})$
FIR-Filter	3.1	2	3	6	0,2 s	2	2	4	128
IIR-Filter	A.1	2	7	13	1,9 s	1	4	9	128
STAF	A.2	2	7	12	0,3 s	2	2	4	240
KEA-1	A.3	2	12	14	0,4 s	2	0	5	38400
KEA-2	8.1	2	11	13	3,4 s	2	2	5	38400
WDF	A.4	3	11	32	6013,7 $s*$	1	6	30	528

*Die 3. und die 4. Teiloptimierung wurde nach 3300 s bzw. 2000 s unterbrochen. Mit dem bisher gefundenen Ergebnis wurde die Untersuchung jeweils fortgeführt.

- gewichtete Summation der Ausdehnung des Iterationsraums je Raumrichtung.

Die zweite Spalte der Tabelle gibt die Referenz zu dem Algorithmus als UItA an. Anschließend wird die Dimension n des Iterationsraum $\mathcal{I}$, die Anzahl $|\mathcal{S}|$ an Zuweisungen und die Anzahl $|\mathcal{E}|$ an Datenabhängigkeiten für den Algorithmus angegeben.

Für jeden Algorithmus wurde weiterhin die Lösungszeit und die ermittelten Parameter: (1) die Anzahl r an Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren, (2) die Anzahl an Packoperationen, (3) die Manhattan-Distanz (MH-Dist.) und (4) das Volumen $V(\widehat{\mathcal{K}}^{\square})$ des den Partitionsraum $\widehat{\mathcal{K}}$ umschließenden Hyperquaders $\widehat{\mathcal{K}}^{\square}$ des reindizierten und teilwortparallelisierten Algorithmus angegeben.

Die Lösungszeiten für die ersten fünf Algorithmen sind relativ gering (max. 3,4 s). Die Optimierung für den Wellendigitalfilter (WDF) benötigt hingegen sehr viele Zeit. Die Ursachen liegen in der geringen Ausrichtung der Abhängigkeitsvektoren. Dadurch erhalten wir für die Minimierung der Manhatten-Distanz ein GLP mit 43 997 Zeilen, 25 302 Spalten und 151 100 Nicht-Null-Elementen[2], welches nur sehr mühsam durchsucht werden kann. Ähnlich verhält es sich mit dem GLP zur Minimierung des Volumens.

Die Ursache für die große Größe des GLP ist die große Anzahl an Nebenbedingungen zur Verhinderung von konträren Abhängigkeitsvektoren.

Aufgrund der langen Lösungszeiten werden dem dritten und vierten GLP die Lösungen des zweiten bzw. dritten GLPs als Startwerte vorgegeben. Somit hat die Optimierung immer eine Lösung und kann somit jederzeit unterbrochen werden, um kurzfristig suboptimale Lösungen zu erhalten.

Die Lösungszeiten für die verschiedenen Algorithmen können vorher leider nicht abgeschätzt werden. So besitzt der Kantenerkennungsalgorithmus KEA-1, siehe Alg. A.3 auf Seite 231, sowohl mehr Zuweisungen als auch mehr Abhängigkeitsvektoren als der Kantenerkennungsalgorithmus KEA-2, siehe Alg. 8.1 auf Seite 203, doch die Bestimmung der Lösung für KEA-2 dauert 8,5 mal länger, als die Bestimmung der Lösung für KEA-1.

[2]nach der Voroptimierung durch ILOG-CPLEX.

Veränderung der Form des Iterationsraums $\mathcal{I}$

Die Partitionierung, wie in Kapitel 4.2 beschrieben, erlaubt nur die Kachelung in Hyperquader (siehe Definition 3.3 auf Seite 24). Somit sollte der gemeinsame Iterationsraum $\mathcal{I}$ des Algorithmus möglichst ebenfalls ein Hyperquader sein oder der den Iterationsraum $\mathcal{I}$ umschließende Hyperquader $\mathcal{I}^{\square}$ sollte möglichst klein sein, denn in diesem Fall führt die nachfolgende Partitionierung zu einer minimalen Anzahl an Kacheln, was wiederrum zu einer kurzen Abarbeitungszeit für den Algorithmus führt.

Ein weiterer Aspekt bei der Veränderung der Form des Iterationsraums ist die Ausrichtung der Abhängigkeitsvektoren $\mathbf{d} \in \mathcal{D}$ des Algorithmus, denn diese können durch die Veränderung der Form des Iterationsraums möglicherweise ausgerichtet werden.

Im Allgemeinen ist der Iterationsraum $\mathcal{I} = \{\mathbf{i} \,|\, \mathbf{A} \cdot \mathbf{i} \geq \mathbf{a}_0\}$ ein Polyeder. Jede Ungleichung $\mathbf{a}_i \cdot \mathbf{i} \geq a_{0,i}$ des Ungleichungssystems $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{a}_0$ beschreibt dabei eine Hyperebene. Es sollen nun Reindizierungsmatrizen $\mathbf{C}$ gefunden werden, die den Iterationsraum $\mathcal{I}$ derart verändern, sodass die Ungleichung $\widetilde{\mathbf{a}}_i \cdot \widetilde{\mathbf{i}} \geq \widetilde{a}_{0,i}$ im transformierten Iterationsraum $\widetilde{\mathcal{I}}$ eine obere oder untere Begrenzung des umschließenden Hyperquaders $\widetilde{\mathcal{I}}^{\square}$ beschreibt. Die obere bzw. untere Begrenzung des Hyperquaders wird mit

$$-\mathbf{e}_j \cdot \widetilde{\mathbf{i}} \geq -\widetilde{a}^{\sqcap}_{0,j} \qquad \text{bzw.} \qquad \mathbf{e}_j \cdot \widetilde{\mathbf{i}} \geq \widetilde{a}^{\sqcup}_{0,j}, \tag{7.23}$$

beschrieben, wobei $\mathbf{e}_j$ die j-te Zeile bzw. Spalte der Einheitsmatrix $\mathbf{E}$ ist, siehe Definition 3.3 auf Seite 24.

Satz 7.3
Die transformierte Hyperebene $\widetilde{\mathbf{a}}_i \cdot \widetilde{\mathbf{i}} \geq \widetilde{a}_{0,i}$ ist genau dann eine obere oder untere Begrenzung, wenn für eine Zeile der Reindizierungsmatrix $\mathbf{C}$ gilt:

$$\mathbf{c}_j = \pm \mathbf{a}_i. \tag{7.24}$$

Beweis 7.3
Die Matrix $\widetilde{\mathbf{A}}$ des transformierten Iterationsraums $\widetilde{\mathcal{I}}$ wird bestimmt mit $\widetilde{\mathbf{A}} = \mathbf{A} \cdot \mathbf{C}^{-1}$, siehe (4.2) auf Seite 38. Somit gilt für jede Zeile i der Matrix $\widetilde{\mathbf{A}}$: $\widetilde{\mathbf{a}}_i = \mathbf{a}_i \cdot \mathbf{C}^{-1}$. Ist die Zeile $\mathbf{a}_i$ eine Zeile $\mathbf{c}_j$ der Reindizierungsmatrix $\mathbf{C}$ so folgt: $\mathbf{e}_j = \mathbf{c}_j \cdot \mathbf{C}^{-1}$, denn $\mathbf{E} = \mathbf{C} \cdot \mathbf{C}^{-1}$. Für $\mathbf{c}_j = -\mathbf{a}_i$ folgt entsprechend $-\mathbf{e}_j = -\mathbf{c}_j \cdot \mathbf{C}^{-1}$.

Durch Kombination der Zeilenvektoren $\mathbf{a}_i$ aus der Matrix $\mathbf{A}$ sollen nun mögliche Reindizierungsmatrizen $\mathbf{C}$ ermittelt werden. Dabei ist die Beschränkung $|\det(\mathbf{C})| = 1$ aus Abschnitt 4.1 einzuhalten. Folgendes Vorgehen wurde gewählt:

(1) Bestimmung aller paarweise linear unabhängigen Zeilenvektoren $\mathbf{a}_i = \begin{pmatrix} a_{i\,1} & \dots & a_{i\,n} \end{pmatrix}$ aus der Polyedermatrix $\mathbf{A} = (a_{i\,j})_{i=1\dots m,\, j=1\dots n}$. Für paarweise linear unabhängige Vektoren gilt: $\operatorname{Rg}\begin{pmatrix} \mathbf{a}_{i_1} \\ \mathbf{a}_{i_2} \end{pmatrix} = 2$. Die paarweise linear unabhängigen Vektoren werden in der Menge $\mathcal{A}$ zusammengefasst:

$$\mathcal{A} = \{\mathbf{a} \,|\, \forall_{i_1 \neq i_2} \operatorname{Rg}\begin{pmatrix} \mathbf{a}_{i_1} \\ \mathbf{a}_{i_2} \end{pmatrix} = 2\}. \tag{7.25}$$

(2) Ermittlung aller Kombinationen $C^n_{|\mathcal{A}|}$ von Vektoren $\mathbf{a}$ aus der Menge $\mathcal{A}$. Jede Kombination ist eine Reindizierungsmatrix $\mathbf{C} = \left(\mathbf{a}_1, \ldots, \mathbf{a}_j, \ldots, \mathbf{a}_n\right)^t$, wenn gilt $|\det(\mathbf{C})| = 1$.

(3) Die Reihenfolge der Vektoren der Reindizierungsmatrix $\mathbf{C}$ wird so gewählt, dass der Absolutwert des Produkts der Elemente der Hauptdiagonale maximal ist. Anschließend werden alle Zeilen der Matrix negiert, deren Element in der Hauptdiagonale negativ ist.

Die Menge $\mathcal{A}$ kann um weitere Vektoren erweitert werden, wenn für diese Vektoren die Bedingung aus (7.25) erfüllt ist. Beispielsweise könnten alle Zeilenvektoren $\mathbf{e}_j$ der Einheitsmatrix $\mathbf{E}$ zu der Menge $\mathcal{A}$ hinzugefügt werden. Durch Vergrößerung der Menge $\mathcal{A}$, erhöht sich die Anzahl $C^n_{|\mathcal{A}|}$ an Kombinationen und somit die Anzahl der möglichen Reindizierungsmatrizen $\mathbf{C}$ oder es entstehen erstmals verwendbare Reindizierungsmatrizen $\mathbf{C}$.

Beispiel 7.5
Wir haben einen zweidimensionalen Iterationsraum $\mathcal{I} = \{\mathbf{i} = \binom{i_1}{i_2} \mid 0 \leq i_1 + i_2 \leq 3 \wedge 0 \leq i_1 - i_2 \leq 5\}$ *in der Form eines Rechtecks, das um 45° gedreht wurde. Dieser Iterationsraum ist in Abbildung 7.5(a) dargestellt. Die implizite Definition des Iterationsraums lautet:*

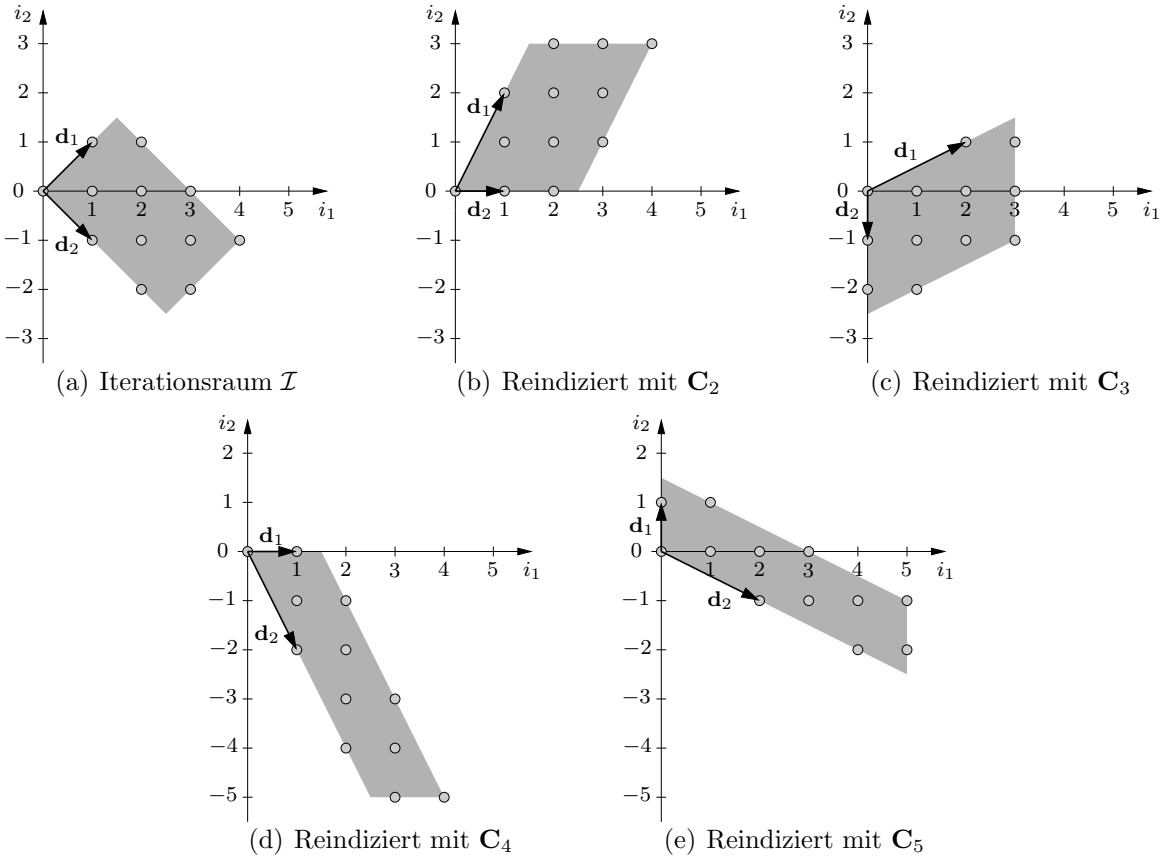
$$\mathcal{I} = \left\{\mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix} \cdot \mathbf{i} \geq \begin{pmatrix} 0 \\ -3 \\ 0 \\ -5 \end{pmatrix}\right\}. \tag{7.26}$$

Für die Menge $\mathcal{A} = \{\begin{pmatrix} 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & -1 \end{pmatrix}\}$ *haben wir zwei paarweise linear unabhängige Vektoren aus den Zeilen der Polyedermatrix* $\mathbf{A}$ *ermittelt. Eine Matrix* $\mathbf{C} = \left(\begin{smallmatrix} 1 & 1 \\ -1 & 1 \end{smallmatrix}\right)$ *kann aus der Menge* $\mathcal{A}$ *bestimmt werden. Diese Matrix ist aber keine Reindizierungsmatrix, denn für die Determinante dieser Matrix gilt:* $\det(\mathbf{C}) = 2$.

Durch Erweiterung der Menge $\mathcal{A}$ *um die beiden Zeilenvektoren der Einheitsmatrix* $\mathbf{E}$, *können* $C^2_4 = \frac{4!}{2!(4-2)!} = 6$ *Kombinationen und damit 6 Matrixen ermittelt werden.*

$$\mathbf{C}_1 = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \qquad \mathbf{C}_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad \mathbf{C}_3 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$
$$\mathbf{C}_4 = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \qquad \mathbf{C}_5 = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \qquad \mathbf{C}_6 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Nur Matrix $\mathbf{C}_1$ *ist keine Reindizierungsmatrix, denn für sie gilt wieder:* $\det(\mathbf{C}_1) = 2$. *Die Reindizierungsmatix* $\mathbf{C}_6$ *ist die Einheitsmatrix, weshalb wir eine identische Abbildung* $\widetilde{\mathcal{I}}_6 = \mathcal{I}$ *erhalten. In den Abbildungen 7.5(b)–(e) sind die durch die Reindizierungsmatrizen* $\mathbf{C}_2$ *bis* $\mathbf{C}_5$ *entstandenen Iterationsräume* $\widetilde{\mathcal{I}}_2$ *bis* $\widetilde{\mathcal{I}}_5$ *dargestellt. Die Beschreibung der*



(a) Iterationsraum $\mathcal{I}$ (b) Reindiziert mit $\mathbf{C}_2$ (c) Reindiziert mit $\mathbf{C}_3$

(d) Reindiziert mit $\mathbf{C}_4$ (e) Reindiziert mit $\mathbf{C}_5$

Abbildung 7.5: Versionen der Reindizierung des Iterationsraum $\mathcal{I}$

reindizierten Iterationsräume als impliziete Definition lautet:

$$\widetilde{\mathcal{I}}_2 = \left\{ \widetilde{\mathbf{i}} = \begin{pmatrix} \widetilde{i}_1 \\ \widetilde{i}_2 \end{pmatrix} \mid \begin{pmatrix} 0 & 1 \\ 0 & -1 \\ 2 & -1 \\ -2 & 1 \end{pmatrix} \cdot \widetilde{\mathbf{i}} \geq \begin{pmatrix} 0 \\ -3 \\ 0 \\ -5 \end{pmatrix} \right\} \tag{7.27}$$

$$\widetilde{\mathcal{I}}_3 = \left\{ \widetilde{\mathbf{i}} = \begin{pmatrix} \widetilde{i}_1 \\ \widetilde{i}_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 1 & -2 \\ -1 & 2 \end{pmatrix} \cdot \widetilde{\mathbf{i}} \geq \begin{pmatrix} 0 \\ -3 \\ 0 \\ -5 \end{pmatrix} \right\} \tag{7.28}$$

$$\widetilde{\mathcal{I}}_4 = \left\{ \widetilde{\mathbf{i}} = \begin{pmatrix} \widetilde{i}_1 \\ \widetilde{i}_2 \end{pmatrix} \mid \begin{pmatrix} 2 & -1 \\ -2 & 1 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \cdot \widetilde{\mathbf{i}} \geq \begin{pmatrix} 0 \\ -3 \\ 0 \\ -5 \end{pmatrix} \right\} \tag{7.29}$$

$$\widetilde{\mathcal{I}}_5 = \left\{ \widetilde{\mathbf{i}} = \begin{pmatrix} \widetilde{i}_1 \\ \widetilde{i}_2 \end{pmatrix} \mid \begin{pmatrix} 1 & 2 \\ -1 & -2 \\ 1 & 0 \\ -1 & 0 \end{pmatrix} \cdot \widetilde{\mathbf{i}} \geq \begin{pmatrix} 0 \\ -3 \\ 0 \\ -5 \end{pmatrix} \right\} \tag{7.30}$$

$$\widetilde{\mathcal{I}}_6 = \mathcal{I} \tag{7.31}$$

Im Folgenden sollen zwei Strategien für die Auswahl einer Reindizierungsmatrix vorgestellt werden.

Minimierung des umschließenden Hyperquaders Es sollen alle Reindizierungsmatrizen $\mathbf{C} \in \mathcal{C}_{V_{\min}}$ ermittelt werden, die zu einem minimalen Hyperquader $\widetilde{\mathcal{I}}^{\square}$ führen. Hierzu wird der reindizierte Iterationsraum $\widetilde{\mathcal{I}}$ des Iterationsraums $\mathcal{I}$ mit Hilfe der Reindizierungsfunktion $\rho_s(\mathbf{i}) = \mathbf{C} \cdot \mathbf{i} + \mathbf{0}$ bestimmt. Anschließend ermitteln wir den umschließende Hyperquader $\widetilde{\mathcal{I}}^{\square}$ und berechnen deren Volumen $V(\widetilde{\mathcal{I}}^{\square})$. Alle Reindizierungsmatrizen $\mathbf{C}$, die zu einem minimalen Volumen $V(\widetilde{\mathcal{I}}^{\square})$ führen, sind Elemente der Menge $\mathcal{C}_{V_{\min}}$.

Alternativ kann direkt über die Ecken $\mathbf{v}_i$ des Ausgangsiterationsraums $\mathcal{I}$ das Volumen $V(\widetilde{\mathcal{I}}^{\square})$ des umschließenden Hyperquaders $\widetilde{\mathcal{I}}^{\square}$ des reindizierten Iterationsraums $\widetilde{\mathcal{I}}$ bestimmt werden:

$$V(\widetilde{\mathcal{I}}^{\square}) = \prod_{j=1}^{n} (\lfloor \max_{i=1}^{n_\nu} \mathbf{c}_j \mathbf{v}_i \rfloor - \lceil \min_{i=1}^{n_\nu} \mathbf{c}_j \mathbf{v}_i \rceil + 1), \tag{7.32}$$

wobei $\mathbf{c}_j = \mathbb{Z}^{1 \times n}$ die Zeile j der Reindizierungsmatrix $\mathbf{C}$ und $\mathbf{v}_i \in Q^n$ die Spalte i der Eckenmatrix $\mathbf{V}$ des Iterationsraums $\mathcal{I}$ ist.

Ist das Volumen $V(\widetilde{\mathcal{I}}^{\square})$ minimal, so enthält der Hyperquarder $\widetilde{\mathcal{I}}^{\square}$ die wenigsten Iterationen $\widetilde{\mathbf{i}} \in \widetilde{\mathcal{I}}^{\square} \backslash \widetilde{\mathcal{I}}$, welche Element des Hyperquarders $\widetilde{\mathcal{I}}^{\square}$ aber nicht des reindizierten Iterationsraums $\widetilde{\mathcal{I}}$ sind. Das Volumen $V(\widetilde{\mathcal{I}})$ des reindizierten Iterationsraums $\widetilde{\mathcal{I}}$ ist für alle Reindizierungen konstant, wenn gilt: $\det(\mathbf{C}) = \pm 1$.

Beispiel 7.6
Für die in Beispiel 7.5 gefundenen reindizierten Iterationsräume $\widetilde{\mathcal{I}}_j$, $j = 2, \ldots 5$ sind die ermittelten Volumen $V(\widetilde{\mathcal{I}}_j^{\square})$ in Tabelle 7.4 angegeben,

Diese Werte lassen sich auch anschaulich aus Abbildung 7.5 bestimmen. Die Reindizierungsmatrix $\mathbf{C}_3$ führt zu dem geringsten Volumen $V(\widetilde{\mathcal{I}}_3^{\square}) = 16$.

Tabelle 7.4: Volumen des den Iterationsraum $\widetilde{\mathcal{I}}$ umschließenden Hyperquaders $\widetilde{\mathcal{I}}^{\square}$ abhängig von der Reindizierungsmatrix $\mathbf{C}$

	$\mathbf{E}$	$\mathbf{C}_2$	$\mathbf{C}_3$	$\mathbf{C}_4$	$\mathbf{C}_5$
$V(\widetilde{\mathcal{I}}^{\square})$	20	20	16	30	24

Tabelle 7.5: Veränderung der Abhängigkeitsvektoren durch Reindizierungsmatrix $\mathbf{C}$

	$\mathbf{E}$	$\mathbf{C}_2$	$\mathbf{C}_3$	$\mathbf{C}_4$	$\mathbf{C}_5$
$\mathbf{d}_1 = \binom{1}{1}$	$\binom{1}{1}$	$\binom{1}{2}$	$\binom{2}{1}$	$\binom{1}{0}$	$\binom{0}{1}$
$\mathbf{d}_2 = \binom{1}{-1}$	$\binom{1}{-1}$	$\binom{1}{0}$	$\binom{0}{-1}$	$\binom{1}{-2}$	$\binom{2}{-1}$

Ausrichtung der Abhängigkeitsvektoren Durch die Veränderung der Form des Iterationsraum mittels der Reindizierungsmatrix $\mathbf{C}$ ändern sich auch die Abhängigkeitsvektoren $\mathbf{d}(e)$ der Datenabhängigkeiten $e \in \mathcal{E}$. Die reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ erhalten wird durch

$$\widetilde{\mathbf{d}}(e) = \widetilde{\mathbf{L}}^{-1}_{\sigma(e)}(\widetilde{\mathbf{l}}_{\sigma(e)} - \widetilde{\mathbf{r}}_{\sigma(e)\delta(e)}) \tag{7.33}$$

$$\widetilde{\mathbf{d}}(e) = (\mathbf{L}_{\sigma(e)}\mathbf{C}^{-1})^{-1}\big((\mathbf{l}_{\sigma(e)} - \mathbf{L}_{\sigma(e)}\mathbf{C}^{-1}\mathbf{c}_{\sigma(e)}) - (\mathbf{r}^k_{\sigma(e)\delta(e)} - \mathbf{R}^k_{\sigma(e)\delta(e)}\mathbf{C}^{-1}\mathbf{c}_{\delta(e)})\big), \tag{7.34}$$

da für die Reindizierungsvektoren $\mathbf{c}_{\sigma(e)} = \mathbf{c}_{\delta(e)} = \mathbf{0}$ gilt, folgt:

$$\widetilde{\mathbf{d}}(e) = \mathbf{C} \cdot \mathbf{L}^{-1}_{\sigma(e)}(\mathbf{l}_{\sigma(e)} - \mathbf{r}^k_{\sigma(e)\delta(e)}) \tag{7.35}$$

$$\widetilde{\mathbf{d}}(e) = \mathbf{C} \cdot \mathbf{d}(e). \tag{7.36}$$

Beispiel 7.7
Der Algorithmus mit dem Iterationsraum $\mathcal{I}$ aus Beispiel 7.5 hat zwei Abhängigkeitsvektoren $\mathbf{d}(e_1) = \mathbf{d}_1 = \binom{1}{1}$ und $\mathbf{d}(e_2) = \mathbf{d}_2 = \binom{1}{-1}$. Diese sind in Abbildung 7.5(a) je einmal exemplarisch dargestellt. Beide Abhängigkeitsvektoren lassen sich durch eine Verschiebung der Iterationsräume zueinander nicht verändern, denn $\sigma(e_1) = \delta(e_1)$ und $\sigma(e_2) = \delta(e_2)$. In Raumrichtung i_2 sind diese beiden Abhängigkeitsvektoren unterschiedlich gerichtet. Wird der Algorithmus durch die in Beispiel 7.5 ermittelten Reindizierungsmatrizen $\mathbf{C}_2$ bis $\mathbf{C}_5$ reindiziert, so ergeben sich die in Tabelle 7.5 zusammengefassten und in Abbildung 7.5 dargestellten Abhängigkeitsvektoren.

Die Reindizierung mit Matrix $\mathbf{C}_2$ bzw. $\mathbf{C}_4$ führt zu gerichteten Abhängigkeitsvektoren. Das Volumen $V(\widetilde{\mathcal{I}}^{\square}_4 = 30)$ durch die Reindizierung mit Matrix $\mathbf{C}_4$ ist aber größer als durch die Reindizierung mit Matrix $\mathbf{C}_2$ $(V(\widetilde{\mathcal{I}}^{\square}_2 = 20))$, siehe Beispiel 7.6 bzw. Tabelle 7.4. Somit würde Matrix $\mathbf{C}_2$ als Reindizierungsmatrix bevorzugt werden, wenn sichergestellt werden soll, dass die Abhängigkeitsvektoren in allen Raumrichtungen ausgerichtet sind.

Da die Ausrichtung der Abhängigkeitsvektoren ebenfalls durch die Verschiebung der Iterationsräume zueinander verändert werden kann, überprüfen wir mit Hilfe eines ganzzahligen, linearen Programms die maximale Ausrichtung der Abhängigkeitsvektoren. Das GLP ergibt sich aus den Optimierungsvorgaben für die Verschiebung der Iterationsräume zueinander wie auf Seite 141 ff. und im Anhang D angegeben.

Auswahl der Reindizierungsmatrix Mit der zu Beginn dieses Abschnitts vorgestellten Methode lassen sich mehrere Reindizierungsmatrizen finden. Durch die Bewertung der Reindizierungsmatrizen bezüglich des Volumens $V(\widetilde{\mathcal{I}}^{\square})$ des resultierenden Hyperquarders $\widetilde{\mathcal{I}}^{\square}$ und der Anzahl an Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren, ist nun eine Bewertung der Reindizierungsmatrizen möglich.

Im Allgemeinen hat das Volumen eine größere Bedeutung als die Anzahl an Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren. Die paretooptimalen Lösungen eigener Untersuchungen zeigen, dass die Erhöhung der Anzahl an Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren zu einer deutlichen Vergrößerung des Volumens führt. Da die Abarbeitungszeit des Algorithmus im Allgemeinen direkt proportional mit dem Volumen des Iterationsraums des Algorithmus steigt, bedeutet diese Vergrößerung des Volumens eine deutliche Verlängerung der Abarbeitungszeit.

Ist die Anzahl r an Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren kleiner als die Dimension n, so kann der Algorithmus maximal auf ein $n^{\kappa} = r$ dimensionales Rechenfeld abgebildet werden.

Beispiel 7.8 (Ermittlung der Reindizierungsmatrizen für Wellendigitalfilter) *Ausgangsalgorithmus ist der Wellendigitalfilter [FN91] mit elf Zuweisungen und 32 Datenabhängigkeiten, wie in Algorithmus A.4 auf Seite 232 angegeben. Der Iterationsraum $\mathcal{I}$ dieses Algorithmus ist kein Hyperquarder.*

Das Volumen $V(\mathcal{I}^{\square})$ des den Iterationsraum $\mathcal{I}$ umschließenden Hyperquarders $\mathcal{I}^{\square}$ des Wellendigitalfilters (WDF) beträgt $V(\mathcal{I}^{\square}) = 1400$. Durch Verschieben der Iterationsräume zueinander lassen sich die Abhängigkeitsvektoren nur in einer Raumrichtung ausrichten, wie in Tabelle 7.3 auf Seite 153 zu sehen ist.

Mit der oben beschriebenen Strategie wurden innerhalb von 19 Sekunden[3] 73 Reindizierungsmatrizen $\mathbf{C}$ für den Wellendigitalfilter (WDF) gefunden und bewertet. Die große Anzahl an Reindizierungsmatrizen resultiert aus dem Hinzufügen weiterer unabhängiger Zeilenvektoren zu der Menge $\mathcal{A}$, welche durch Veränderung der vorhandenen Zeilenvektoren[4] ermittelt wurden. Die Bewertung für alle Reindizierungsmatrizen erfolgte bezüglich des Volumens $V(\widetilde{\mathcal{I}}^{\square})$ und der Ausrichtung r der Abhängigkeitsvektoren.

Zwölf paretooptimale Lösungen wurden gefunden. Die vier Reindizierungsmatrizen

$$\mathbf{C}_0 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{C}_1 = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{C}_2 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \qquad \mathbf{C}_3 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

führen zu einem Volumen von $V(\widetilde{\mathcal{I}}^{\square}) = 1960$ und einer Ausrichtung der Abhängigkeitsvektoren in $r = 3 = n$ Raumrichtungen. Bei einer Reindizierung mit den Reindizierungsmatrizen

$$\mathbf{C}_4 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{C}_5 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix}, \qquad \mathbf{C}_6 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \qquad \mathbf{C}_7 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix},$$
$$\mathbf{C}_8 = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{C}_9 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 2 \end{pmatrix}, \qquad \mathbf{C}_{10} = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \qquad \mathbf{C}_{11} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 2 \end{pmatrix}$$

[3] Intel Core2 Duo E6750 @ 2.66GHz

[4] Einzelne Elemente der Vektoren $\mathbf{a}_i$ wurden um „1“ erhöht ($a_{i,j} < 0$) oder reduziert ($a_{i,j} > 0$) bis $a_{i,j} = 0$ ist.

erhalten wir ein geringeres Volumen von $V(\widetilde{\mathcal{I}}^{\square}) = 1400$. *Die Abhängigkeitsvektoren sind in diesem Fall jedoch nur in* $r = 2 < n$ *Raumrichtungen ausgerichtet.*

Strategie für die Durchführung der Reindizierung

In den vorherigen beiden Absätzen wurde beschrieben, wie eine Reindizierung zum einen durch das Verschieben der Iterationsräume oder zum anderen durch die Veränderung der Form des gemeinsamen Itertionsraums erfolgen kann. In diesem Absatz sollen nun beide Methoden miteinander kombiniert werden.

Zuerst wird die Form des gemeinsamen Iterationsraums verändert. Durch diese Optimierung wird erreicht, dass die Ausrichtung der Abhängigkeitsvektoren maximal oder das Volumen $V(\widetilde{\mathcal{I}}^{\square})$ des den Iterationsraum $\widetilde{\mathcal{I}}$ umschließenden Hyperquaders $\widetilde{\mathcal{I}}^{\square}$ minimal ist. Bei dieser Optimierung können mehrere paretooptimale Lösungen entstehen. Anschließend erfolgt das Verschieben der Iterationsräume $\mathcal{I}_s$ zueinander. Hierbei minimieren wir die Anzahl an Packoperationen, die absolute Länge der Abhängigkeitsvektoren (Manhattan-Distanz) und das Volumen $V(\widehat{\mathcal{K}}^{\square})$ des den Iterationsraum $\widehat{\mathcal{K}}$ umschließenden Hyperquaders $\mathcal{K}^{\square}$. Abschließend wird der gemeinsame Iterationsraum $\widetilde{\mathcal{I}}$ so im Koordinatensystem verschoben, dass je Raumrichtung die entsprechend dem später zu ermittelnden Ablaufplan zeitlich erste Copartition vollständig ist, siehe Seite 150 ff.

Nach der Optimierung der Form des Iterationsraums können wir mehrere Reindizierungsmatrizen erhalten, die paretooptimal sind. Bei kleineren Algorithmen können alle paretooptimalen Lösungen mittels des zweiten Optimierungsschritts weiter optimiert werden. Anschließend wird die beste Gesamtlösung ausgewählt. Wie nachfolgendes Beispiel zeigt, kann der Rechenaufwand hierfür aber sehr hoch sein.

Beispiel 7.9 (Ermittlung der Reindizierung für Wellendigitalfilter)
In Beispiel 7.8 wurde der erste Optimierungsschritt, die Optimierung des Wellendigitalfilters bezüglich der Reindizierungsmatrix $\mathbf{C}$ *vorgestellt. Als Lösung der Optimierung erhielten wir zwölf Reindizierungsmatrizen nach einer Lösungszeit von 18,9 s. Für alle Matrizen wurden im zweiten Optimierungsschritt die Reindizierungsvektoren* $\mathbf{c}_s$ *bestimmt. Die gefundenen Lösungen und die Lösungszeiten sind in Tabelle 7.6 zusammengefasst. Für den zweiten Optimierungsschritt wurden die folgenden Parameter gewählt:*

- *Teilwortparallelität mit vier Teilwörtern,*
- *keine Vorgaben für die Auswahl der Raumrichtung für die TWP,*
- *keine Bewertung der Ein- und Ausgabedaten bezüglich TWP,*
- *keine Vorgaben für die Datentransferrichtung und*
- *gewichtete Summation der Ausdehnung des Iterationsraums je Raumrichtung.*

Die Lösungszeit für die Gesamtanalyse beträgt ca. 29 Minuten und 30,4 Sekunden (1751,5 s + 18,9 s = 1770,4 s). Man beachte die vergleichsweise kurzen Lösungszeiten für

Tabelle 7.6: Ermittlung der Reindizierungsparameter für Wellendigitalfilter

		Optimierung bezüglich $\mathbf{C}$			Optimierung bezüglich $\mathbf{c}_s$			
Nr.	$\mathbf{C}$	Lös.-zeit	$V(\mathcal{I}^{\square})$	Ausricht.	Lös.-zeit	Packop.	MH-Dist.	$V(\widehat{\mathcal{K}}^{\square})$
0	$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$	18,9 s	1960	3	2,7 s	10	50	$4 \cdot 11 \cdot 16 = 704$
1	$\begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$		1960	3	15,7 s	10	50	$11 \cdot 4 \cdot 16 = 704$
2	$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$		1960	3	1,1 s	8	50	$16 \cdot 4 \cdot 11 = 704$
3	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$		1960	3	1,1 s	8	50	$4 \cdot 16 \cdot 11 = 704$
4	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$		1400	2	159,2 s	6	46	$3 \cdot 11 \cdot 16 = 528$
5	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix}$		1400	2	84,5 s	6	48	$3 \cdot 11 \cdot 16 = 528$
6	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$		1400	2	83,5 s	6	48	$3 \cdot 16 \cdot 11 = 528$
7	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$		1400	2	82,7 s	6	50	$3 \cdot 15 \cdot 11 = 495$
8	$\begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$		1400	2	1071,8 s	6	46	$11 \cdot 3 \cdot 16 = 528$
9	$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 2 \end{pmatrix}$		1400	2	97,0 s	6	50	$16 \cdot 3 \cdot 11 = 528$
10	$\begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$		1400	2	87,1 s	6	48	$11 \cdot 3 \cdot 16 = 528$
11	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 2 \end{pmatrix}$		1400	2	65,1 s	6	50	$15 \cdot 3 \cdot 11 = 495$
				$\sum$	1770,4 s			

die Optimierung bezüglich der Reindizierungsvektoren $\mathbf{c}_s$ *im Vergleich zu der Lösungszeit 6013,7 s (mit Abbruch) für diesen Wellendigitalfilter mit* $\mathbf{C} = \mathbf{E}$ *aus Tabelle 7.3. Dieser Unterschied resultiert aus der besseren Ausrichtung der Abhängigkeitsvektoren, wodurch die GLPs des zweiten Optimierungsschritts merklich kleiner werden. Dieser Unterschied in der Lösungszeit ist auch für die ersten vier Reindizierungsmatrizen, welche zu einer Ausrichtung in drei Raumrichtungen führen, im Vergleich zu den anderen acht Reindizierungsmatrizen, welche zu einer Ausrichtung in zwei Raumrichtungen führen, gut zu erkennen.*

Weiterhin wird anhand der acht Reindizierungsmatrizen $\mathbf{C}_4$, $\mathbf{C}_5$ *bis* $\mathbf{C}_{11}$ *deutlich, dass der Lösungszeit für eine Optimierungsaufgabe nicht oder nur schwer abschätzbar ist. Für diese acht Versionen haben wir einen Geschwindigkeitsunterschied von 1:16,5 bei ähnlichen Randbedingungen.*

Betrachten wir nun die gefundenen Lösungen. Von den vier Reindizierungsmatrizen, die zu einer Ausrichtung der Abhängigkeitsvektoren in drei Raumrichtung führen, erhalten wir bei zwei Matrizen acht Packoperationen für die Teilwortparallelität und bei den anderen beiden 10 Packoperationen. Die beiden Matrizen $\mathbf{C}_2$ *und* $\mathbf{C}_3$ *würden in diesem Fall bevorzugt werden.*

Die Reindizierungsmatrizen $\mathbf{C}_4$ *bis* $\mathbf{C}_{11}$ *führen immer zu sechs Packoperationen für die Teilwortparallelität. Die Optimierungen bezüglich der Reindizierungsvektoren* $\mathbf{c}_s$ *unterscheiden sich hier nur bezüglich der Manhattan-Distanz (MH-Dist.) und des Volumens* $V(\widehat{\mathcal{K}}^{\square})$*. Je zwei Reindizierungsmatrizen führen zu einer minimalen Manhattan-Distanz und zwei zu einem minimalen Volumen.*

Für eine effektive Abbildung des Algorithmus auf das Rechenfeld sollte eine Reindizierungsmatrix ausgewählt werden, die zu einem minimalen Volumen führt, denn dadurch verringert sich im Allgemeinen die Abarbeitungsdauer des Algorithmus auf der Zielarchitektur.

7.1.2 Anpassung an die Algorithmenkonvention

Im Kapitel 4 wurde eine Algorithmenkonvention vorgestellt, welche Ausgangspunkt für den Abbildungsprozess ist. Hierbei besteht der Algorithmus nur aus Eingabezuweisungen, internen Berechnungszuweisungen und Ausgabezuweisungen. Wie bereits in diesem Abschnitt erwähnt wurde, kann jeder UItA in eine solche Form überführt werden.

Hat der Algorithmus diese Form, so ist eine Transformation des Algorithmus mit den in den Kapiteln 4 bis 6 vorgestellten Methoden möglich. Bei der Partitionierung zerfallen die Abhängigkeitsvektoren $\mathbf{d}(e) \neq \mathbf{0}$ des Algorithmus in mehrere partitionierte Abhängigkeitsvektoren $\mathbf{d}^{\kappa}_{i_1,i_2,\ldots,i_p}(e)$. Diese partitionierten Abhängigkeitsvektoren führen zu mehreren Teilzuweisungen in der Zuweisung $s = \delta(e)$ des partitionierten Algorithmus, wie in Abschnitt 4.2.3 gezeigt wurde. Mehrere Teilzuweisungen in einer Zuweisung haben wiederum einen erhöhten Steueraufwand bei der Abarbeitung des Algorithmus zur Folge. Weiterhin verursachen Abhängigkeitsvektoren, welche keine Nullvektoren sind,

im Allgemeinen einen Datentransfer im Rechenfeld und zwischen Rechenfeld und Speicherarchitektur, was wiederum einen erhöhten Energiebedarf zur Folge hat. Aus diesem Grund sollen alle internen Berechnungszuweisungen eines Algorithmus in lokale Berechnungszuweisungen, wie in Abschnitt 3.3.3 beschrieben wurde, umgewandelt werden.

Anschließend untersuchen wir, ob einzelne Interndatenzuweisungen des Algorithmus eliminiert werden können. Eine Interndatenzuweisung kann entfernt werden, wenn eine andere Interndatenzuweisung den Datentransfer bereits teilweise oder vollständig realisiert. Gegebenenfalls wird hierzu der Iterationsraum der genutzten Interndatenzuweisung erweitert. Auf diese Weise minimieren wir die Anzahl an Abhängigkeitsvektoren $\mathbf{d}(e) \neq \mathbf{0}$.

Betrachten wir dazu die vier Zuweisungen aus Algorithmus 7.1 mit $\mathbf{d}^1_{s\,t_1} = \mathbf{d}^1_{s\,t_2}$. Die Zuweisung s stellt die Variable $y_s[\mathbf{i}]$ bereit. Diese Variable wird mittels der Zuweisungen t_1 und t_2 im Iterationsraum transferiert. Zuweisung c verwendet die transferierte Variable $y_{t_2}[\mathbf{i}]$.

Algorithmus 7.1 Algorithmus mit eliminierbarer Interndatenzuweisung

$s:$	$y_s[\mathbf{i}] = \ldots,$	$\mathbf{i} = \mathcal{I}_s,$
$t_1:$	$y_{t_1}[\mathbf{i}] = y_s[\mathbf{i} - \mathbf{d}^1_{s\,t_1}],$	$\mathbf{i} = \mathcal{I}_{t_1},$
$t_2:$	$y_{t_2}[\mathbf{i}] = y_s[\mathbf{i} - \mathbf{d}^1_{s\,t_2}],$	$\mathbf{i} = \mathcal{I}_{t_2},$
$c:$	$y_c[\mathbf{i}] = F_c(\ldots, y_{t_2}[\mathbf{i}], \ldots),$	$\mathbf{i} = \mathcal{I}_c,$

Sind die Abhängigkeitsvektoren $\mathbf{d}^1_{s\,t_1} = \mathbf{d}(e_1)$ und $\mathbf{d}^1_{s\,t_2} = \mathbf{d}(e_2)$ der Interndatenzuweisungen t_1 und t_2 gleich und haben beide Datenabhängigkeiten die gleiche Quelle $\sigma(e_1) = \sigma(e_2)$, so kann die Interndatenzuweisung t_2 eliminiert werden. Hierfür muss der Iterationsraum $\mathcal{I}_{t_1}$ der Interndatenzuweisung t_1 erweitert werden und alle Variablen $y_{t_2}[.]$ sind durch Variablen $y_{t_1}[.]$ zu ersetzen wie in Algorithmus 7.2 zu sehen ist.

Algorithmus 7.2 Algorithmus mit eliminierter Interndatenzuweisung

$s:$	$y_s[\mathbf{i}] = \ldots,$	$\mathbf{i} = \mathcal{I}_s,$
$s':$	$y_{t_1}[\mathbf{i}] = y_s[\mathbf{i} - \mathbf{d}^1_{s\,t_1}],$	$\mathbf{i} = \mathrm{conv}(\mathcal{I}_{t_1} \cup \mathcal{I}_{t_2}),$
$c:$	$y_c[\mathbf{i}] = F_c(\ldots, y_{t_1}[\mathbf{i}], \ldots),$	$\mathbf{i} = \mathcal{I}_c.$

7.1.3 Zusammenfassung

Die Vorverarbeitung dient im Wesentlichen der Veränderung des Algorithmus, sodass bei der nachfolgenden mehrstufigen Copartitionierung eine gute Abbildung des Algorithmus auf eine Zielarchitektur realisiert werden kann.

Durch die Beschränkung der Partitionen auf die Form von Hyperquarder, wird eine Vorverarbeitung unumgänglich. Anderenfalls erhielten wir in vielen Fällen nur suboptimale Lösungen.

Die Reindizierung verschiebt die Iterationsräume $\mathcal{I}_s$ der Zuweisungen s des Algorithmus zueinander und verändert die Form und Größe des gemeinsamen Iterationsraum $\mathcal{I}$. Das ermöglicht es uns, die Abhängigkeitsvektoren auszurichten, die Anzahl an entstehenden Packoperationen bei nachfolgender Teilwortparallelisierung, die Manhattan-Distanz und die Größe des gemeinsamen Iterationsraums zu minimieren. Weiterhin kann für eine effektive Copartitionierung der gemeinsame Iterationsraum günstig im Koordinatensystem positioniert werden.

Mit der Anpassung an die Algorithmenkonvention findet eine Umformulierung des Algorithmus statt. Die Ziele der Umformulierung sind ausschließlich lokale Berechnungszuweisungen und möglichst wenige Interndatenzuweisungen, bei denen die Abhängigkeitsvektoren ungleich Null sind. Dadurch vereinfacht sich die nachfolgende (Teilwort)-Partitionierung des Algorithmus und mehrfache Datentransfers gleicher Daten werden vermieden.

7.2 Mehrstufige Modifizierte Copartitionierung

In Kapitel 5 wurde die mehrstufige modifizierte Copartitionierung (MMC) vorgestellt. Wie an dieser Stelle bereits erwähnt wurde, ist die MMC der entscheidende Abbildungsschritt zur Abbildung des Algorithmus auf unsere Zielarchitektur. Die MMC besitzt abhängig von der Anzahl p an Copartitionierungsstufen vielfältige Möglichkeiten der Abbildung. In Abbildung 7.1 wurde illustriert, mit welcher Copartitionierungsstufe welche Ziele verfolgt werden sollen.

Die erste Copartitionierungsstufe dient der Ausnutzung der Teilwortparallelität. Hierfür muss ermittelt werden, wie viele Daten des Algorithmus für die teilwortparallele Berechnung in den Funktionseinheiten zusammengefasst werden können. Diese Anzahl B an Teilwörtern je DvB ist abhängig vom Befehlssatz der Zielarchitektur und dem Datenformat der Variablen des Algorithmus. Für DSPs, wie dem TMS320C64x [Tex06] von Texas Instruments, kann eine Umwandlung des Algorithmus von Fließkommazahlberechnungen zu Ganzzahlberechnungen, wie in [MCCS02] vorgestellt, notwendig sein. Durch eine, ebenfalls in [MCCS02] präsentierte Reduzierung der Wortbreite der Daten des Algorithmus auf das, bezüglich der Genauigkeit, notwendige Maß, kann die Anzahl B an Teilwörtern je DvB erhöht werden, wie in [MGQS03] gezeigt wurde.

Mit der zweiten Copartitionierungsstufe soll eine Anpassung an das Rechenfeld erfolgen, das heißt, an die Größe $\boldsymbol{\vartheta}^{\square}$ des Rechenfeldes, die verfügbaren Verbindungskanäle $\mathbf{q} \in \mathcal{Q}$ und an den lokalen Speicher L_R in jedem Prozessorelement. Diese Copartitionierungsstufe entspricht dem klassischen Rechenfeldentwurf, wie er bereits in anderen Arbeiten [FM99][Eck01][Dut04][SM04] vorgestellt wurde. In [FM99] und [SM06] lag der Schwerpunkt der Abbildung auf der Minimierung der Kosten für die Kanäle $\mathbf{q}$ und für die Register (lokaler Speicher). Die Ausnutzung des vorhandenen lokalen Speichers L_R durch die Wahl der entsprechenden LS-2-Partition $\boldsymbol{\Theta}^{S_2}$ wurde unter anderem in [Eck01] und [Dut04] gezeigt. In dieser Arbeit wird in Abschnitt *Datenwiederverwendung zwischen*

Partitionen ab Seite 175 eine neue Methode basierend auf der Partitionierung der Abhängigkeitsvektoren, siehe Abschnitt 4.2.2 ab Seite 44, vorgestellt.

Die ersten beiden Copartitionierungsstufen dienen vorrangig der Ausnutzung der Parallelität auf der Ebene der Funktionseinheiten und auf der Ebene des Rechenfeldes. Der Grad der Parallelverarbeitung wurde hierbei mittels der Partitionierungsmatrizen $\mathbf{\Theta}^{P_1}$ und $\mathbf{\Theta}^{P_2}$ festgelegt. Die nachfolgenden Copartitionierungsstufen $3 \ldots n = 2 + L$ dienen der Anpassung an die hierarchisch aufgebaute Speicherarchitektur, welche durch die Größen M_{L_l} für den l-ten Zwischenspeicher $1 \leq l \leq L$ und M_M für den Hauptspeicher gegeben sind. Für jeden Zwischenspeicher L_l wird eine Copartitionierungsstufe $2+l$ eingeführt. Da mit diesen Copartitionierungsstufen nur die Reihenfolge der Abarbeitung der Iterationen $\widehat{\boldsymbol{\kappa}}^{P_{2+l-1}}$ verändert werden soll, muss für die Partitionsmatrix $\mathbf{\Theta}^{P_{2+l}} = \mathbf{E}$ gewählt werden. Durch die geeignete Wahl der LS-Partition $\mathbf{\Theta}^{S_{2+l}}$ und der Abfolgevektoren $\boldsymbol{\alpha}^{S_{2+l}}$ und $\widehat{\boldsymbol{\alpha}}^{P_{2+l}}$ kann die Abarbeitung der Iterationen $\widehat{\boldsymbol{\kappa}}^{P_{2+l-1}}$ so beeinflusst werden, dass die Daten in dem Zwischenspeicher L_l maximal wiederverwendet werden. In den Arbeiten [Eck01] und [Dut04] wurden solche Ansätze zur Bestimmung der Partitionsgröße $\boldsymbol{\vartheta}^{S_{2+l}}$ vorgestellt. Ziel der Datenwiederverwendung in den Zwischenspeicher L_l ist die Reduzierung des Energiebedarfs für die Abarbeitung des Algorithmus auf dieser Architektur, denn Speicherzugriffe auf kleine Speicher verbrauchen weniger Energie als Speicherzugriffe auf große Speicher. Die Ideen zur Reduzierung des Energiebedarfs bassieren auf der Datentransfer- und Speicheruntersuchung (DTSE - Data Transfer and Storage Exploration) [CWD+98], welche am IMEC Leuven entwickelt wurde. Am Beispiel der Bewegungsabschätzung wurde in [SCM00] das Zusammenspiel von DTSE und klassischem Rechenfeldentwurf gezeigt.

Der Schwerpunkt der Datenübertragungs- und -speicheruntersuchung liegt bei der Optimierung der Speicherzugriffe und der effizienten Zwischenspeicherung von Daten in lokalen Zwischenspeichern mit dem Ziel, den Energieverbrauch bei der Abarbeitung des Algorithmus zu senken. Wie in [SCM00] gezeigt wurde, ist eine Kombination der Copartitionierungsmethode mit dem DTSE-Ansatz möglich. Dabei findet mittels DTSE eine Optimierung der Speicherzugriffe zum Hauptspeicher und den größeren Zwischenspeichern statt. Mit der Copartitionierung wird der Algorithmus auf das Rechenfeld abgebildet. Weiterhin erfolgt eine Anpassung an die Register in Rechenfeld und die kleineren Zwischenspeicher.

Nach der p-fachen Copartitionierung des Iterationsraum muss abschließend der Ablaufplan ermittelt werden. Hierfür werden die bei der p-fachen Copartitionierung bestimmten Abfolgevektoren $\boldsymbol{\alpha}^{S_p}$ und $\widehat{\boldsymbol{\alpha}}^{P_n}$ verwendet. Mit der Ablaufplanung findet auch die Zuordnung der Funktionseinheiten $m \in \mathcal{M}$ der Prozessorelemente zu den Teilzuweisungen s_t des Algorithmus statt. Besitzen die Prozessorelemente mehrere Funktionseinheiten, so soll mit der Ablaufplanung die Parallelverarbeitung auf dieser Ebene maximiert werden. Mehrere Arbeiten haben sich bereits diesem Thema gewidmet. In [Fim02] und [MFM04] liegt der Schwerpunkt auf der Parallelverarbeitung im Prozessorelement. Mit der Abarbeitung der Partitionen und der Parallelverarbeitung im Prozessorelement beschäftigte sich [HT04].

Nachfolgend werden die verschiedenen Copartitionierungsstufen etwas ausführlicher vor-

gestellt. Hierbei soll vor allem ein Gefühl vermittelt werden, was mit jeder Copartitionierungsstufe erreicht werden kann.

7.2.1 Stufe 1: Teilwortparallelisierung

In Kapitel 6 wurde die Teilwortparallelisierung, das heißt die Transformation eines Algorithmus mit Einzeldatenoperationen zu einem ein-/ausgabeäquivalenten Algorithmus mit TWP-Operationen beschrieben. Die Möglichkeiten für die teilwortparallele Copartitionierung des gemeinsamen Iterationsraums $\mathcal{I} \in \mathbb{Z}^n$ des Algorithmus sind aufgrund der Restriktionen $\boldsymbol{\Theta}^{S_1} = \mathbf{E}$, $\exists!\ \vartheta_j^{P_1} > 1$, $0 \leq j < n$ und $\boldsymbol{\tau}^{\text{offs}_1} = \mathbf{0}$ sehr eingeschränkt. Maximal n Varianten existieren für die Teilwortparallelisierung, wobei n die Dimension des gemeinsamen Iterationsraums $\mathcal{I} \subset \mathbb{Z}^n$ des Algorithmus ist. Hierbei gehen wir davon aus, dass die Anzahl B an Teilwörtern je DvB bekannt ist.

Bei der Auswahl der Raumrichtung j, in der die Iterationen zu DvBs zusammengepackt werden sollen, haben wir das Ziel, die Anzahl an zusätzlichen Packbefehlen zu minimieren. Wurde in der Vorverarbeitung die Anzahl an entstehenden Packoperationen bereits minimiert, siehe Seite 143 ff, so kann die gefundene Raumrichtung j nun für die Bestimmung der Partitionsmatrix genutzt werden. Alternativ kann die Raumrichtung j mit Hilfe (7.7), (7.8) und (7.9) von Seite 144 ff. ermittelt werden. Es ist aber zu beachten, dass dies eine Abschätzung der Anzahl an Packoperationen ist und dass hierbei die Packoperationen aufgrund mehrerer Teilzuweisungen in einer Zuweisung, siehe Abschnitt 6.3, nicht mit einbezogen wurden.

Beispiel 7.10 (Teilwortparallelisierung des FIR-Filters)
Das FIR-Filter, siehe Beispiel 3.1 auf Seite 23, wurde bei der Vorstellung der Teilwortparallelisierung bereits an mehreren Stellen als Beispielalgorithmus gewählt. Der Filter soll auf Funktionseinheiten mit $B = 4$ Teilwörtern im DvB abgebildet werden. Aus der Bewertung der globalen Daten in Beispiel 7.2 auf Seite 144 wird deutlich, dass bei dieser Speicherarchitektur, die Raumrichtung $j = i_1$ für die Teilwortparallelisierung zu bevorzugen ist. In Algorithmus 7.3 ist der transformierte Algorithmus dargestellt.

Nur die Transformation der Zuweisung s_2 des Algorithmus 3.1 auf Seite 23 wurde noch nicht vorgestellt. Die aus dieser Zuweisung resultierenden teilwortparallelen Zuweisungen sind die Zuweisungen s_{x_1}, s_{x_2}, s_p, s_{x_3} und s_2 des Algorithmus 7.3. Mit den Zuweisungen s_{x_1} und s_{x_2} werden die x-Daten eingelesen, wobei die Zuweisung s_{x_1} die Daten als DvBs liest und die Zuweisung s_{x_2} die Daten als Teilwörter einliest. Die Zuweisung s_p packt die Daten aufgrund des Abhängigkeitsvektors $\mathbf{d}^1_{s_2\, s_{2_3}} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ unter Verwendung der Packoperation F_p. Die DvBs $\mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^{P_1}]$ mit $\widehat{\kappa}_1^{P_1} = 0$ sind nicht vollständig. Das benötigte niederwertigste Teilwort wird mit Zuweisung s_{x_3} dem DvB hinzugefügt. Abschließend wird mit Zuweisung s_2 das jeweils gültige DvB $\mathbf{y}_{x,1}[\widehat{\boldsymbol{\kappa}}^{P_1}]$, $\mathbf{y}_{x,3}[\widehat{\boldsymbol{\kappa}}^{P_1}]$ oder $\mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^{P_1}]$ dem DvB $\mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^{P_1}]$ zugewiesen.

Insgesamt besteht das teilwortparallele FIR-Filter aus acht Zuweisungen mit insgesamt dreizehn Teilzuweisungen. Drei zusätzliche Packoperationen mussten zu dem Algorith-

Algorithmus 7.3 FIR-Filter nach der Teilwortparallelisierung

$$
\begin{aligned}
s_1: \quad & \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^{P_1}] = \begin{cases} \text{verteileTW}\big(a[(\,0\;1\,)\,\widehat{\boldsymbol{\kappa}}^{P_1}]\big), & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{1_1}, \\ \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{1_2}, \end{cases} \\
s_{x_1}: \quad & \mathbf{y}_{x,1}[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{x}[(\,1\;-1\,) \cdot \widehat{\boldsymbol{\kappa}}^{P_1}], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{x_1}, \\
s_{x_2}: \quad & \mathbf{y}_{x,2}[\widehat{\boldsymbol{\kappa}}^{P_1}] = \text{leseTW}\big(x[(\,4\;-1\,) \cdot \widehat{\boldsymbol{\kappa}}^{P_1}]\big), & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{x_2}, \\
s_p: \quad & \mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^{P_1}] = \begin{cases} F_p\big(\mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)], \mathbf{0}, 1\big), & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{p_1}, \\ F_p\big(\mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)], \mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)], 1\big), & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{p_2}, \end{cases} \\
s_{x_3}: \quad & \mathbf{y}_{x,3}[\widehat{\boldsymbol{\kappa}}^{P_1}] = \text{ersetzeTW}\big(0, \mathbf{y}_{x,2}[\widehat{\boldsymbol{\kappa}}^{P_1}], \mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^{P_1}]\big), & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{x_3}, \\
s_2: \quad & \mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^{P_1}] = \begin{cases} \mathbf{y}_{x,1}[\widehat{\boldsymbol{\kappa}}^{P_1}], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{2_1}, \\ \mathbf{y}_{x,3}[\widehat{\boldsymbol{\kappa}}^{P_1}], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{2_2}, \\ \mathbf{y}_p[\widehat{\boldsymbol{\kappa}}^{P_1}], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{2_3}, \end{cases} \\
s_3: \quad & \mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^{P_1}] = \begin{cases} \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^{P_1}] \cdot \mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^{P_1}], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{3_1}, \\ \mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)] + \mathbf{y}_a[\widehat{\boldsymbol{\kappa}}^{P_1}] \cdot \mathbf{y}_x[\widehat{\boldsymbol{\kappa}}^{P_1}], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{3_2}, \end{cases} \\
s_4: \quad & \mathbf{y}[(\,1\;0\,) \cdot \widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{y}_y[\widehat{\boldsymbol{\kappa}}^{P_1}], & \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{4_1},
\end{aligned}
$$

mit $\widehat{\mathcal{K}}_{1_1} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid \widehat{\kappa}_1^{P_1} = 0 \wedge 0 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{1_2} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 1 \leq \widehat{\kappa}_1^{P_1} \leq I-1 \wedge 0 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{x_1} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 0 \leq \widehat{\kappa}_1^{P_1} \leq \lceil \frac{I-1}{4} \rceil \wedge \widehat{\kappa}_2^{P_1} = 0\}$,
$\widehat{\mathcal{K}}_{x_2} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid \widehat{\kappa}_1^{P_1} = 0 \wedge 1 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{p_1} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid \widehat{\kappa}_1^{P_1} = 0 \wedge 1 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{p_2} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 1 \leq \widehat{\kappa}_1^{P_1} \leq \lceil \frac{I-1}{4} \rceil \wedge 1 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{x_3} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid \widehat{\kappa}_1^{P_1} = 0 \wedge 1 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{2_1} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 0 \leq \widehat{\kappa}_1^{P_1} \leq \lceil \frac{I-1}{4} \rceil \wedge \widehat{\kappa}_2^{P_1} = 0\}$,
$\widehat{\mathcal{K}}_{2_2} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid \widehat{\kappa}_1^{P_1} = 0 \wedge 1 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{3_2} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 1 \leq \widehat{\kappa}_1^{P_1} \leq \lceil \frac{I-1}{4} \rceil \wedge 1 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{3_1} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 0 \leq \widehat{\kappa}_1^{P_1} \leq \lceil \frac{I-1}{4} \rceil \wedge \widehat{\kappa}_2^{P_1} = 0\}$,
$\widehat{\mathcal{K}}_{3_2} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 0 \leq \widehat{\kappa}_1^{P_1} \leq \lceil \frac{I-1}{4} \rceil \wedge 1 \leq \widehat{\kappa}_2^{P_1} \leq J-1\}$,
$\widehat{\mathcal{K}}_{4_1} = \{\widehat{\boldsymbol{\kappa}}^{P_1} \mid 0 \leq \widehat{\kappa}_1^{P_1} \leq I-1 \wedge \widehat{\kappa}_2^{P_1} = J-1\}$,
sowie $\widehat{\boldsymbol{\kappa}}^{P_1} = (\,\widehat{\kappa}_1^{P_1}\;\widehat{\kappa}_2^{P_1}\,)^t$, $I = 128$ und $J = 20$.

mus hinzugefügt werden, wobei die beiden Packoperationen in der Zuweisung s_p disjunkt sind.

7.2.2 Stufe 2: Anpassung an das Rechenfeld

Mit der zweiten Copartitionierungsstufe soll die Parallelität des Rechenfeldes $\boldsymbol{\vartheta}^{\square}$ und der lokale Speicher L_R mit der Größe M_R in den PEs ausgenutzt werden. Die Anpassung an die Größe des Rechenfeldes erfolgt mit der Partitionsmatrix $\boldsymbol{\Theta}^{P_2}$ und die effiziente Ausnutzung des lokalen Speichers kann mit der Partitionsmatrix $\boldsymbol{\Theta}^{S_2}$ erreicht werden. Bei der Abbildung des Algorithmus auf das Rechenfeld werden auch die benötigten Datenkanäle ermittelt.

Anpassung an die Größe des Rechenfeldes

Haben wir ein n^{κ}-dimensionales Rechenfeld ($\boldsymbol{\vartheta}^{\square} \in \mathbb{N}_{+}^{n^{\kappa}}$), auf das ein Algorithmus mit einem n-dimensionalen Iterationsraum ($\mathcal{I} \subset \mathbb{Z}^n$) abgebildet werden soll, so ergeben sich maximal $(n)_{n^{\kappa}} = \frac{n!}{(n-n^{\kappa})!}$ Möglichkeiten der Abbildung. Ist die Größe $\boldsymbol{\vartheta}^{\square}$ des Rechenfeldes in zwei oder mehr Raumrichtungen gleich groß, so verringert sich dieser Wert ein wenig.

Die Bestimmung der möglichen Partitionierungsmatrizen $\boldsymbol{\Theta}^{P_2}$ erfolgt, indem wir alle Variationen $(n)_{n^{\kappa}}$ von n^{κ} Elementen aus dem Vektor $\boldsymbol{\vartheta}^{P_2} = (\vartheta_1^{P_2}, \vartheta_2^{P_2}, \ldots, \vartheta_n^{P_2})^t$ ermitteln. Den Elementen einer Variation ordnen wir anschließend die Größen $\vartheta_j^{\square}$ des Rechenfeldes zu.

Beispiel 7.11 (Varianten der Abbildung auf das Rechenfeld)
Wir haben ein $n^{\kappa} = 2$-dimensionales Rechenfeld der Größe $\boldsymbol{\vartheta}^{\square} = \left(\begin{smallmatrix}2\\4\end{smallmatrix}\right)$. Auf dieses Rechenfeld soll ein Algorithmus mit einem $n = 3$-dimensionalen Iterationsraum abgebildet werden. Damit ergeben sich $\frac{3!}{(3-2)!} = 6$ Möglichkeiten der Abbildung:

$$\boldsymbol{\Theta}_1^{P_2} = \left(\begin{smallmatrix}2&0&0\\0&4&0\\0&0&1\end{smallmatrix}\right), \quad \boldsymbol{\Theta}_2^{P_2} = \left(\begin{smallmatrix}2&0&0\\0&1&0\\0&0&4\end{smallmatrix}\right), \quad \boldsymbol{\Theta}_3^{P_2} = \left(\begin{smallmatrix}4&0&0\\0&2&0\\0&0&1\end{smallmatrix}\right),$$
$$\boldsymbol{\Theta}_4^{P_2} = \left(\begin{smallmatrix}4&0&0\\0&1&0\\0&0&2\end{smallmatrix}\right), \quad \boldsymbol{\Theta}_5^{P_2} = \left(\begin{smallmatrix}1&0&0\\0&2&0\\0&0&4\end{smallmatrix}\right), \quad \boldsymbol{\Theta}_6^{P_2} = \left(\begin{smallmatrix}1&0&0\\0&4&0\\0&0&2\end{smallmatrix}\right).$$

Die zugehörigen Allokationsmatrizen $\boldsymbol{\Gamma}_i^2$, $1 \leq i \leq 6$ lauten:

$$\boldsymbol{\Gamma}_1^2 = \left(\begin{smallmatrix}1&0&0\\0&1&0\end{smallmatrix}\right), \quad \boldsymbol{\Gamma}_2^2 = \left(\begin{smallmatrix}1&0&0\\0&0&1\end{smallmatrix}\right), \quad \boldsymbol{\Gamma}_3^2 = \left(\begin{smallmatrix}0&1&0\\1&0&0\end{smallmatrix}\right),$$
$$\boldsymbol{\Gamma}_4^2 = \left(\begin{smallmatrix}0&0&1\\1&0&0\end{smallmatrix}\right), \quad \boldsymbol{\Gamma}_5^2 = \left(\begin{smallmatrix}0&1&0\\0&0&1\end{smallmatrix}\right), \quad \boldsymbol{\Gamma}_6^2 = \left(\begin{smallmatrix}0&0&1\\0&1&0\end{smallmatrix}\right).$$

Die Auswahl einer Partitionsmatrix $\boldsymbol{\Theta}_i^{P_2}$ aus den gegebenen Möglichkeiten erfolgt nach Parametern, wie z. B. Effizienz des Ablaufplans und des Datentransfer im Rechenfeld. Zuvor soll aber das Thema *Dimension des Rechenfeldes* im nächsten Abschnitt ausführlicher diskutiert werden, denn ist bei einem Rechenfeld die Dimension des Rechenfeldes noch nicht fixiert, so hat die Dimension des Rechenfeldes entscheidenden Einfluss auf die Abbildung des Algorithmus.

Dimension des Rechenfeldes

Prinzipiell kann die Dimension des Rechenfeldes n^κ durch die Wahl der Partitionsmatrix $\mathbf{\Theta}^{P_2}$ von 1 bis n frei gewählt werden, wobei n die Dimension des gemeinsamen Iterationsraums $\mathcal{I}$ des Algorithmus ist. Da aber das Rechenfeld physisch vorhanden sein muss, sind derzeit nur Rechenfelder der Dimension $n^\kappa = \{1, 2\}$ praktikabel. In der nahen Zukunft scheinen auch dreidimensionale Rechenfelder möglich, denn durch das Stapeln von Chips und das Durchkontaktieren durch die Siliziumscheiben können 3D-Chips [BKM+05] und damit 3D-Rechenfelder gebaut werden.

Für Algorithmen, deren gemeinsamer Iterationsraum eine geringe Dimension hat (z. B. $n = 2$), stellt sich die Frage, ob die Dimension des Rechenfeldes n^κ auch gleich der des gemeinsamen Iterationsraums des Algorithmus sein kann bzw. sollte?

Hierzu soll zu Beginn ein kleiner Exkurs in die Geschichte des Rechenfeldentwurfs gemacht werden [Rao85]. Der klassischen Entwurf von parallelen Rechenfeldern mittels der Ort-Zeit-Transformation führte bei einem n-dimensionalen Iterationsraum des Algorithmus zu einem $n-1$-dimensionalen Rechenfeld voller Größe und einer eindimensionalen Zeitachse. Später wurden Ort-Zeit-Transformationen entwickelt, die zu einem $n-m$-dimensionalen Rechenfeld voller Größe und einem m-dimensionalen Zeitraum führten, wobei m immer größer Null ist [Eck01]. Ziel dieser Ort-Zeit-Transformationen war es Rechenfelder voller Größe der Dimension $n^\kappa \leq n-1$ zu erhalten. Auf diese Rechenfelder voller Größe kann der Algorithmus immer abgebildet werden, denn die Ort-Zeit-Transformation musste die Kausalitätsbedingung erfüllen.

Der neue Rechenfeldentwurf [Sie03] mit der mehrstufigen modifizierten Copartitionierung erlaubt es nun prinzipiell mit Hilfe der Partitionierung, die Iterationen eines Algorithmus mit einem n-dimensionalen gemeinsamen Iterationsraum auf ein n-dimensionales Rechenfeld abzubilden. Doch bei der Bestimmung der Partitionsmatrix $\mathbf{\Theta}^{P_2}$ werden bisher die Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ des Algorithmus vernachlässigt. Besitzt der Algorithmus ungünstige Abhängigkeitsvektoren, so kann eine zeitliche Abarbeitung des Algorithmus auf einem durch $\mathbf{\Theta}^{P_2}$ gegebenen Rechenfeld unmöglich oder zumindestens zeitlich aufwändig sein.

Beispiel 7.12 (Realisierung eines 2D-Algorithmus auf 2D-Rechenfeld)
Am Beispiel des STAF-Algorithmus, dessen Kern in Algorithmus 7.4 dargestellt ist, soll die Problematik der Abbildung eines Algorithmus mit zweidimensionalen Iterationsraum auf ein zweidimensionales Rechenfeld illustriert werden.

Algorithmus 7.4 STAF-Algorithmus

$$
\begin{aligned}
s_1 &: \quad r[\mathbf{i}] = r[\mathbf{i} - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)] && \mathbf{i} \in \mathcal{I}_1 \\
s_2 &: \quad d[\mathbf{i}] = d[\mathbf{i} - \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)] + r[\mathbf{i}] \cdot u[\mathbf{i} - \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)], && \mathbf{i} \in \mathcal{I}_2 \\
s_3 &: \quad u[\mathbf{i}] = u[\mathbf{i} - \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)] + r[\mathbf{i}] \cdot d[\mathbf{i} - \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)], && \mathbf{i} \in \mathcal{I}_3
\end{aligned}
$$

mit $\mathcal{I}_1 = \mathcal{I}_2 = \mathcal{I}_3 = \{\mathbf{i} = \left(\begin{smallmatrix}i\\k\end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \leq i < 8 \wedge 0 \leq k < k_n\}$ und $k_n \in \{13, 14, 120\}$.

Der STAF-Algorithmus soll auf ein Rechenfeld der Größe $\boldsymbol{\vartheta}^{\square} = \left(\begin{smallmatrix}2\\2\end{smallmatrix}\right)$ *realisiert werden. Hierfür partitionieren wir den Algorithmus mit Partitionsmatrix* $\boldsymbol{\Theta}^{P_2} = \left(\begin{smallmatrix}2 & 0\\0 & 2\end{smallmatrix}\right)$ *(*$\boldsymbol{\Theta}^{S_1} = \boldsymbol{\Theta}^{P_1} = \boldsymbol{\Theta}^{S_2} = \mathbf{E}$*). Die Allokationsmatrix hat die Größe* $\boldsymbol{\Gamma}^2 = \mathbf{E}$*. Für die Abarbeitung der Partitionen sind zwei Abfolgevektoren* $\widehat{\boldsymbol{\alpha}}_1^2 = \begin{pmatrix}1 & 2\end{pmatrix}$ *und* $\widehat{\boldsymbol{\alpha}}_2^2 = \begin{pmatrix}2 & 1\end{pmatrix}$ *möglich. In Abbildungen 7.6(a) und (b) ist diese Partitionierung des Iterationsraums für* $0 \leq i < 4$ *und* $K = 8$ *dargestellt. In den Iterationen der Abbildungen sind die Abarbeitungszeitpunkte, welche sich aus den beiden Abfolgevektoren* $\widehat{\boldsymbol{\alpha}}_1^2$ *und* $\widehat{\boldsymbol{\alpha}}_2^2$ *ergeben, dargestellt, wobei das Iterationsintervall* $\lambda = 1$ *ist.*

Betrachten wir die beiden Lösungen etwas genauer:

- ***Abfolgevektor*** $\widehat{\boldsymbol{\alpha}}_1^2 = \begin{pmatrix}1 & 2\end{pmatrix}$***:*** *Aufgrund der Abhängigkeitsvektoren mit* $\mathbf{d}^1_{s_2 s_2} = \mathbf{d}^1_{s_2 s_3} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\mathbf{d}^1_{s_3 s_2} = \mathbf{d}^1_{s_3 s_3} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *muss die Berechnung in den Prozessorelementen* $\mathbf{p}^2 = \boldsymbol{\Gamma}^2 \cdot \boldsymbol{\kappa}^{P_2} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\mathbf{p}^2 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *gegenüber den PEs* $\mathbf{p}^2 = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *und* $\mathbf{p}^2 = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *verzögert begonnen werden. Hierzu wird der Verzögerungsvektor* $\boldsymbol{\tau}^{\text{offs},2} = \begin{pmatrix}1 & 0\end{pmatrix}$ *gesetzt. Eine Verzögerung in Raumrichtung* k *ist nicht notwendig, da der Abhängigkeitsvektor* $\mathbf{d}^1_{s_1 s_1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *ein Propagierungsvektor ist und somit ein verzögerungsfreier Datentransfer möglich ist.*

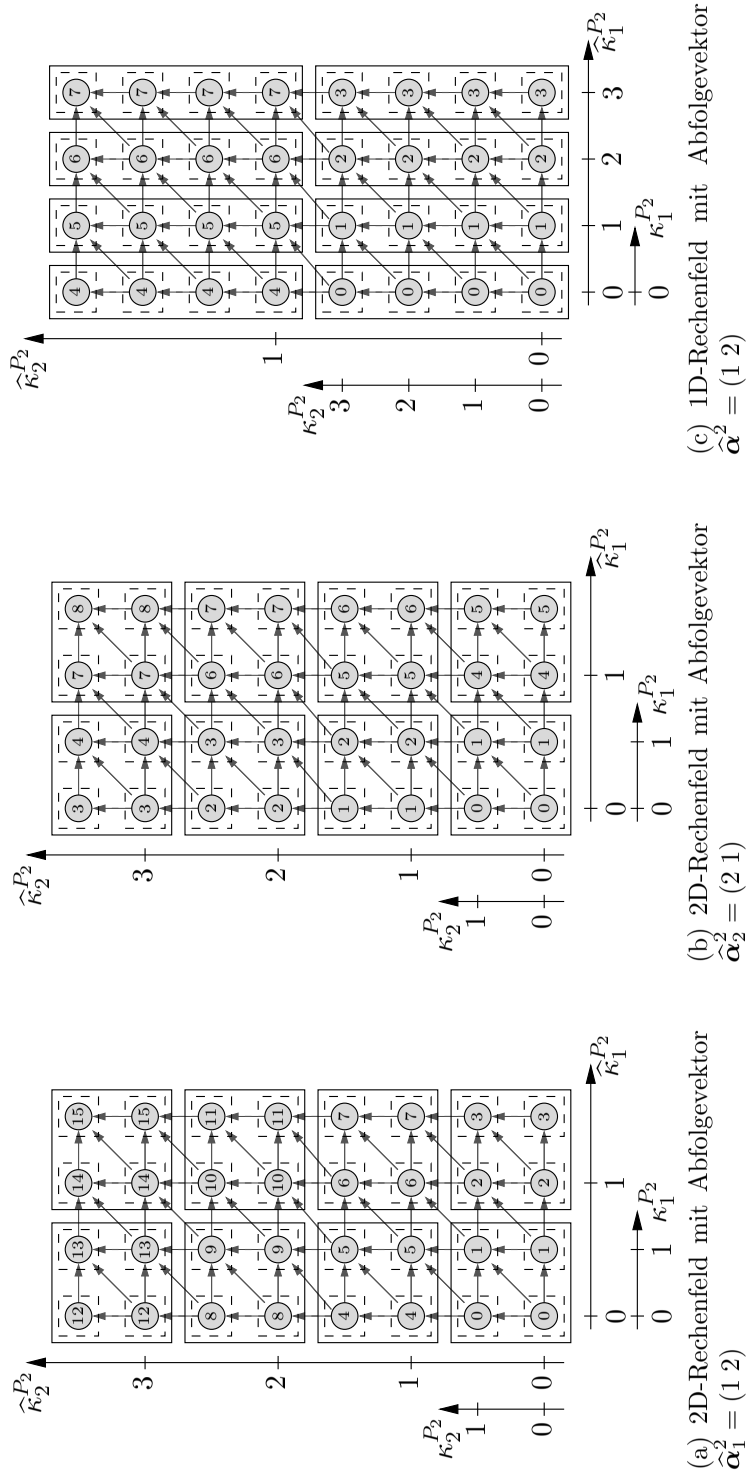
  Entsprechend Abfolgevektor $\widehat{\boldsymbol{\alpha}}^2 = \begin{pmatrix}1 & 2\end{pmatrix}$ *soll nach der Abarbeitung der Copartition* $\widehat{\boldsymbol{\kappa}}^{P_2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *die Copartition* $\widehat{\boldsymbol{\kappa}}^{P_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *abgearbeitet werden. Die Abarbeitung kann aber nicht bereits zum Zeitpunkt* $t = 1$ *begonnen werden, da zu diesem Zeitpunkt erst die für die Iterationen* $\mathbf{i} = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$ *und* $\mathbf{i} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *benötigten Daten in den PEs* $\mathbf{p}^2 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\mathbf{p}^2 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *berechnet werden. Durch die Leerlaufverzögerung* $\widehat{\boldsymbol{\iota}}^{P_2} = \begin{pmatrix}1 & 0\end{pmatrix}$ *erreichen wir die notwendige Verzögerung. Diese Leerlaufverzögerung führt aber dazu, dass in jedem zweiten Iterationsintervall* λ *das Prozessorelement inaktiv ist.*

- ***Abfolgevektor*** $\widehat{\boldsymbol{\alpha}}_2^2 = \begin{pmatrix}2 & 1\end{pmatrix}$***:*** *Die Berechnungen in der zweiten Spalte des 2x2 Rechenfeldes müssen wieder verzögert gegenüber der ersten Spalte erfolgen, was durch den Verzögerungsvektor* $\boldsymbol{\tau}^{\text{offs},2} = \begin{pmatrix}1 & 0\end{pmatrix}$ *erreicht wird.*

  Durch den Abfolgevektor $\widehat{\boldsymbol{\alpha}}_2^{P_2} = \begin{pmatrix}2 & 1\end{pmatrix}$ *erfolgt nun nach der Abarbeitung der Copartition* $\widehat{\boldsymbol{\kappa}}^{P_2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *die Copartition* $\widehat{\boldsymbol{\kappa}}^{P_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$*. Der zwischen beiden Copartitionen existierende Abhängigkeitsvektor* $\mathbf{d}^1_{s_1 s_1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *ist ein Propagierungsvektor. Somit kann Copartition* $\widehat{\boldsymbol{\kappa}}^{P_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *verzögerungsfrei direkt nach Copartition* $\widehat{\boldsymbol{\kappa}}^{P_2} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *abgearbeitet werden. Für die Leerlaufverzögerung kann* $\widehat{\boldsymbol{\iota}}^{P_2} = \begin{pmatrix}0 & 0\end{pmatrix}$ *gewählt werden. Somit benötigt diese Realisierung ca. 50 % der Zeit gegenüber der Version mit* $\widehat{\boldsymbol{\alpha}}_1^2$*.*

  Der Nachteil dieser Realisierung ist aber die schlechte Wiederverwendung der d- und u-Daten, denn erst nach $\lceil K/2 \rceil - 1$ *bzw.* $\lceil K/2 \rceil$ *Takten werden die in den PEs* $\mathbf{p}^2 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\mathbf{p}^2 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *ermittelten d- und u-Daten wiederverwendet.*

Wie sich zeigt, ist eine Abbildung des STAF-Algorithmus auf das zweidimensionale Rechenfeld prinzipiell möglich. Doch beide Realisierungen haben ihre Nachteile.



(a) 2D-Rechenfeld mit Abfolgevektor $\widehat{\boldsymbol{\alpha}}_1^2 = (1\,2)$

(b) 2D-Rechenfeld mit Abfolgevektor $\widehat{\boldsymbol{\alpha}}_2^2 = (2\,1)$

(c) 1D-Rechenfeld mit Abfolgevektor $\widehat{\boldsymbol{\alpha}}^2 = (1\,2)$

Abbildung 7.6: Realisierung des STAF-Algorithmus auf 2D- und 1D-Rechenfeld

Alternativ ist in Abbildung 7.6(c) die Abbildung des STAF-Algorithmus auf ein eindimensionales Rechenfeld der Größe $\boldsymbol{\vartheta}^{\square} = \begin{pmatrix}4\end{pmatrix}$*, mit Allokationsmatrix* $\boldsymbol{\Gamma}^2 = \begin{pmatrix}0 & 1\end{pmatrix}$ *und Abfolgevektor* $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix}1 & 2\end{pmatrix}$ *dargestellt. Diese Realisierung erlaubt die Abarbeitung des Algorithmus in der kürzesten Zeit und eine kurzfristige Wiederverwendung der d- und u-Daten ist ebenfalls gegeben.*

Durch die Abbildung des STAF-Algorithmus auf ein eindimensionales Rechenfeld ergeben sich mehr Freiheitsgrade für eine effiziente zeitliche Abarbeitung des Algorithmus auf dem Rechenfeld.

Für eine effektive Abbildung eines Algorithmus auf ein Rechenfeld sollten deshalb die folgenden beiden Regeln beachtet werden.

Regel 7.1 (Dimension des Rechenfeldes)
Ist die Dimension n^{κ} *eines Rechenfeldes frei wählbar, so sollte die Dimension* n^{κ} *so gering als möglich gewählt werden. Rechenfelder mit geringer Dimension führen meist zu einer effektiveren Realisierung und der Steueraufwand für diese Realisierung ist geringer.*

Regel 7.2 (Erste Abarbeitungsrichtung)
Wird ein n-dimensionaler Algorithmus auf ein n^{κ}*-dimensionales Rechenfeld mit* $n > n^{\kappa}$ *abgebildet werden, so sollte die erste Abarbeitungsrichtung* $\widehat{\alpha}_1^{P_2} = \pm j$ *für die Abarbeitung der LP-2-Partitionen* $\widehat{\boldsymbol{\kappa}}^{P_2}$ *eine Raumrichtung* j *sein, in der die Ausdehnung* $\widehat{\vartheta}_{j,j}^{P_2}$ *der LP-2-Partition gleich eins ist.*

Vermeidung konträrer äußerer Teilvektoren

Sind die Abhängigkeitsvektoren eines Algorithmus nicht ausgerichtet, siehe Abschnitt „Ausrichtung der Abhängigkeitsvektoren“ auf Seite 141 ff., so können durch die Copartitionierung mit $\boldsymbol{\Theta}^{S_2}$ und $\boldsymbol{\Theta}^{P_2}$ konträre äußere Teilvektoren $\widehat{\mathbf{d}}_{i_1}^{P_2} = -k \cdot \widehat{\mathbf{d}}_{i_1}^{P_2} \neq \mathbf{0}$, $k \in \mathbb{Q}_+$ entstehen. Das ist durch Verringerung der Ausdehnung der Partitionen zu vermeiden. Besitzt ein partitionierter Algorithmus konträre äußere Teilvektoren so kann keine sequenzielle Abarbeitung der LP-2-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ gefunden werden.

Beispiel 7.13 (Vermeidung konträrer äußerer Teilvektoren)
Der in Beispiel 7.1 auf Seite 142 vorgestellte Algorithmus mit den unausgerichteten Abhängigkeitsvektoren $\mathbf{d}(e_1) = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, $\mathbf{d}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\mathbf{d}(e_3) = \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)$ *soll mit* $\boldsymbol{\Theta}^{S_1} = \boldsymbol{\Theta}^{P_1} = \boldsymbol{\Theta}^{S_2} = \mathbf{E}$ *und* $\boldsymbol{\Theta}_1^{P_2} = \left(\begin{smallmatrix}4&0\\0&2\end{smallmatrix}\right)$ *partitioniert werden. Durch die Copartitionierung der Abhängigkeitsvektoren erhalten wir für den Abhängigkeitsvektor* $\mathbf{d}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *den partitionierten Abhängigkeitsvektor* $\mathbf{d}^{\kappa}_{1,1,1,2}(e_2) = \left(\mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad \left(\begin{smallmatrix}-3\\0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\right)^t$ *und für den Abhängigkeitsvektor* $\mathbf{d}(e_3) = \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)$ *den partitionierten Abhängigkeitsvektor* $\mathbf{d}^{\kappa}_{1,1,1,2}(e_3) = \left(\mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad \left(\begin{smallmatrix}3\\1\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)\right)^t$*. Die äußeren Teilvektoren* $\widehat{\mathbf{d}}^{P_2}_{1,1,1,2}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\widehat{\mathbf{d}}^{P_2}_{1,1,1,2}(e_3) = \left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)$ *dieser beiden partitionierten Abhängigkeitsvektoren sind konträr.*

Durch Veränderung der Partitionierungsmatrix $\boldsymbol{\Theta}^{P_2}$ *zu* $\boldsymbol{\Theta}_2^{P_2} = \left(\begin{smallmatrix}8&0\\0&1\end{smallmatrix}\right)$ *kann die Entstehung konträrer äußerer Teilvektoren vermieden werden. Die beiden partitionierten Abhängig-*

keitsvektoren, welche sich nun aus dem Abhängigkeitsvektor $\mathbf{d}(e_3) = \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)$ *ergeben, lauten* $\mathbf{d}^{\kappa}_{1,1,1,1}(e_3) = \left(\mathbf{0}\ \ \mathbf{0}\ \ \mathbf{0}\ \ \left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)\ \ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right)^t$ *und* $\mathbf{d}^{\kappa}_{1,1,1,2}(e_3) = \left(\mathbf{0}\ \ \mathbf{0}\ \ \mathbf{0}\ \ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\ \ \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)\right)^t$*. Die beiden äußeren Teilvektoren* $\widehat{\mathbf{d}}^{P_2}_{1,1,1,1}(e_3) = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *und* $\widehat{\mathbf{d}}^{P_2}_{1,1,1,2}(e_3) = \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)$ *sind nicht konträr zu den äußeren Teilvektoren* $\widehat{\mathbf{d}}^{P_2}_{1,1,1,1}(e_1) = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, $\widehat{\mathbf{d}}^{P_2}_{1,1,1,1}(e_2) = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *und* $\widehat{\mathbf{d}}^{P_2}_{1,1,1,2}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$*, welche wir aufgrund der Partitionierung der Abhängigkeitsvektoren* $\mathbf{d}(e_1) = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *und* $\mathbf{d}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *erhalten haben.*

Der Grad der Parallelverarbeitung ist für beide Partitionierungsmatrizen $\mathbf{\Theta}^{P_2}_1$ *und* $\mathbf{\Theta}^{P_2}_2$ *mit* $\det\left(\mathbf{\Theta}^{P_2}\right) = 8$ *Prozessorelementen gleich.*

Datentransfer im Rechenfeld

Die in einem Iterationsraum zwischen Iterationen transferierten Daten müssen im resultierenden Rechenfeld zwischen den Prozessorelementen des Rechenfeldes oder den Rechenfeld und dem Speicher transferiert werden. Der Datentransfer im Iterationsraum wird durch die Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ beschrieben. Mit der Copartitionierung werden auch die Abhängigkeitsvektoren $\mathbf{d}(e)$ partitioniert. Dabei kann ein Abhängigkeitsvektor $\mathbf{d}(e)$ in mehrere partitionierte Abhängigkeitsvektoren $\mathbf{d}^{\kappa} \in \mathcal{D}^{\kappa}(e)$ zerfallen. Der partitionierte Abhängigkeitsvektor $\mathbf{d}^{\kappa} = \left(\mathbf{d}^{S_1}\ \mathbf{d}^{P_1}\ \mathbf{d}^{S_2}\ \mathbf{d}^{P_2}\ \widehat{\mathbf{d}}^{P_2}\right)^t$ besteht bei der zweistufigen Copartitionierung aus fünf Teilvektoren. Der Teilvektor $\mathbf{d}^{P_2}$ beschreibt hierbei den Datentransfer zwischen den Prozessorelementen des Rechenfeldes.

Beispiel 7.14 (Datentransfer im Rechenfeld)
Das teilwortparallele FIR-Filter, siehe Algorithmus 7.3, hat vier verschiedene Abhängigkeitsvektoren $\mathbf{d}^{\kappa_1}_1 = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, $\mathbf{d}^{\kappa_1}_2 = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$, $\mathbf{d}^{\kappa_1}_3 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\mathbf{d}^{\kappa_1}_4 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$*. Diese zerfallen durch die zweite Copartitionierung mit* $\mathbf{\Theta}^{S_2} = \mathbf{E}$ *und* $\mathbf{\Theta}^{P_2} = \left(\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right)$ *(*$\mathbf{\Gamma}^2 = \mathbf{E}$*) in folgende partitionierte Abhängigkeitsvektoren* $\mathbf{d}^{\kappa_2}_{i_1,i_2} = \left(\mathbf{d}^{S_2}_{i_1}\ \mathbf{d}^{P_2}_{i_1,i_2}\ \widehat{\mathbf{d}}^{P_2}_{i_1,i_2}\right)^t \in \mathcal{D}^{\kappa_2}(e)$*:*

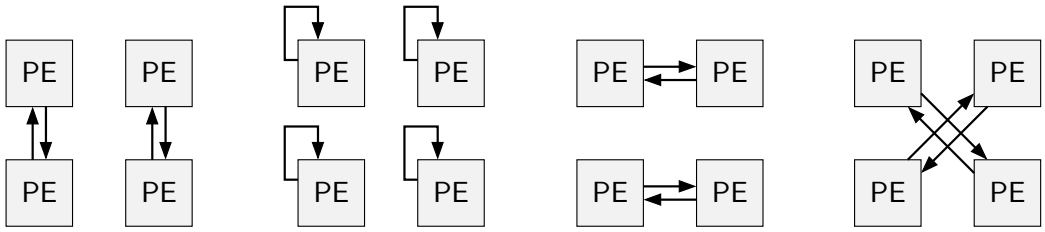
$$\mathbf{d}^{\kappa_1}_1 = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right): \quad \mathbf{d}^{\kappa_2}_{1,1} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \quad \mathbf{d}^{\kappa_2}_{1,2} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\-1\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix},$$

$$\mathbf{d}^{\kappa_1}_2 = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right): \quad \mathbf{d}^{\kappa_2}_{2,1} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix},$$

$$\mathbf{d}^{\kappa_1}_3 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right): \quad \mathbf{d}^{\kappa_2}_{3,1} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \quad \mathbf{d}^{\kappa_2}_{3,1} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}-1\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{pmatrix},$$

$$\mathbf{d}^{\kappa_1}_4 = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right): \quad \mathbf{d}^{\kappa_2}_{4,1} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \quad \mathbf{d}^{\kappa_2}_{4,2} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right)\\\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \quad \mathbf{d}^{\kappa_2}_{4,3} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)\\\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{pmatrix} \quad \mathbf{d}^{\kappa_2}_{4,4} = \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\\left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right)\\\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix}.$$

Abbildung 7.7 zeigt die Datentransfers für die verschiedenen Abhängigkeitsvektoren. Die Abhängigkeitsvektoren $\mathbf{d}^{\kappa_1}_1 = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *und* $\mathbf{d}^{\kappa_1}_3 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *führen zu vertikalen bzw. horizontalen*



(a) $\mathbf{d}_1 = (0\ 1)^t$ (b) $\mathbf{d}_2 = (0\ 0)^t$ (c) $\mathbf{d}_3 = (1\ 0)^t$ (d) $\mathbf{d}_4 = (1\ 1)^t$

Abbildung 7.7: Datentransfer im Rechenfeld bei verschiedenen Abhängigkeitsvektoren

Datentransfers, wie auch die Abhängigkeitsteilvektoren $\mathbf{d}_i^{P_2}$ *zeigen. Für den Abhängigkeitsvektor* $\mathbf{d}_2^{\kappa_1} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ *erhalten wir lokale Datenwiederverwendung. Der Abhängigkeitsvektor* $\mathbf{d}_4^{\kappa_1} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *verursacht Datentransfers in allen Diagonalrichtungen (*$\mathbf{d}_{4,1}^{P_2} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$, $\mathbf{d}_{4,2}^{P_2} = \left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right)$, $\mathbf{d}_{4,3}^{P_2} = \left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)$, $\mathbf{d}_{4,4}^{P_2} = \left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right)$*).*

Inwieweit die Datentransfers $\mathbf{d}^{P_2}$ im Rechenfeld zu Datenkanälen $\mathbf{q} = \mathbf{\Gamma}^2 \cdot \mathbf{d}^{P_2} \in \mathcal{Q}$ und Speicherung der Daten im Rechenfeld oder zu globaler Speicherung der Daten führen, ist abhängig von dem zeitlichen Abstand $t^{\triangle}(\mathbf{d}^{\kappa}(e), e)$ zwischen Datenerzeugung und Datenverwendung und der gegebenen Architektur. Der zeitliche Abstand $t^{\triangle}(\mathbf{d}^{\kappa}(e), e)$, siehe (5.41) auf Seite 78, ist abhängig von dem partitionierten Abhängigkeitsvektoren $\mathbf{d}^{\kappa}(e)$. Diese Entscheidungen zwischen Transfer im Rechenfeld und globaler Speicherung sollen im folgenden Beispiel vorgestellt werden.

Beispiel 7.15 (Datentransfer zu Kanälen)
Betrachten wir den Abhängigkeitsvektor $\mathbf{d}_4^{\kappa_1} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *aus Beispiel 7.14. Dieser Abhängigkeitsvektor zerfällt in vier partitionierte Abhängigkeitsvektoren* $\mathbf{d}_{4,1}^{\kappa_1}$, $\mathbf{d}_{4,2}^{\kappa_1}$, …, $\mathbf{d}_{4,4}^{\kappa_1}$. *Um den zeitlichen Abstand* $t^{\triangle}(\mathbf{d}^{\kappa_2}(e), e)$, *siehe (5.41) auf Seite 78, für alle partitionierten Abhängigkeitsvektoren ermitteln zu können, müssen alle Parameter der Gleichung*

$$t^{\triangle}(\mathbf{d}^{\kappa_2}(e), e) = \lambda \left(\boldsymbol{\tau}^{S_2} \cdot \mathbf{d}^{S_2} + T^S \cdot \widehat{\boldsymbol{\tau}}^{P_2} \cdot \widehat{\mathbf{d}}_2^P \right) + \boldsymbol{\tau}^{\text{offs},2} \cdot \mathbf{d}_2^P + t_{0,\delta(e)} - t_{v,\sigma(e)} \tag{7.37}$$

bestimmt werden. Das Iterationsintervall λ *soll für dieses Beispiel* $\lambda = 1$ *sein. Der Ablaufvektor ist* $\boldsymbol{\tau}^{S_2} = \mathbf{0}$ *und die Abarbeitungsdauer ist* $T^{S_2} = 1$, *da* $\mathbf{\Theta}^{S_2} = \mathbf{E}$. *Für die Bestimmung des Ablaufvektors* $\widehat{\boldsymbol{\tau}}^P$ *wählen wir für den Abfolgevektor* $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix}2 & 1\end{pmatrix}$, *damit die Leerlaufverzögerung* $\widehat{\boldsymbol{\iota}}^{P_2} = \mathbf{0}$ *ist. Somit ergibt sich für den Ablaufvektor* $\widehat{\boldsymbol{\tau}}^{P_2} = \begin{pmatrix}10 & 1\end{pmatrix}$ *bei der Größe* $J = 20$ *und* $\mathbf{\Theta}^{P_2} = \left(\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right)$. *Die Differenz* $t_{0,\delta(e)} - t_{v,\sigma(e)}$ *betrage der Einfachheit halber immer Null. Somit gilt für unser Beispiel:*

$$t^{\triangle}(\mathbf{d}^{\kappa_2}(e), e) = \begin{pmatrix}0 & 0\end{pmatrix} \cdot \mathbf{d}^{S_2} + \begin{pmatrix}10 & 1\end{pmatrix} \cdot \widehat{\mathbf{d}}^{P_2} + \begin{pmatrix}1 & 0\end{pmatrix} \cdot \mathbf{d}^{P_2}. \tag{7.38}$$

Für die vier partitionierten Abhängigkeitsvektoren $\mathbf{d}_{4,i}^{\kappa_2} = (\mathbf{d}_{4,i}^{S_2}\ \mathbf{d}_{4,i}^{P_2}\ \widehat{\mathbf{d}}_{4,i}^{P_2})^t$, $1 \leq i \leq 4$

erhalten wir somit:

$$\mathbf{d}_{4,1}^{\kappa_2} = \begin{pmatrix} \binom{0}{0} \\ \binom{1}{1} \\ \binom{0}{0} \end{pmatrix} \qquad t_{4,1}^{\triangle} = 1, \qquad \mathbf{d}_{4,2}^{\kappa_2} = \begin{pmatrix} \binom{0}{0} \\ \binom{1}{-1} \\ \binom{0}{1} \end{pmatrix} \qquad t_{4,2}^{\triangle} = 2, \tag{7.39}$$

$$\mathbf{d}_{4,3}^{\kappa_2} = \begin{pmatrix} \binom{0}{0} \\ \binom{-1}{1} \\ \binom{1}{0} \end{pmatrix} \qquad t_{4,3}^{\triangle} = 9, \qquad \mathbf{d}_{4,4}^{\kappa_2} = \begin{pmatrix} \binom{0}{0} \\ \binom{-1}{-1} \\ \binom{1}{1} \end{pmatrix}. \qquad t_{4,4}^{\triangle} = 10. \tag{7.40}$$

Bei den partitionierten Abhängigkeitsvektoren $\mathbf{d}_{4,1}^{\kappa_2}$ und $\mathbf{d}_{4,2}^{\kappa_2}$ erfolgt die Datenwiederverwendung im nächsten oder übernächsten Takt. Eine Realisierung dieser Datentransfers mittels der Datenkanäle $\mathbf{q}_1 = \mathbf{\Gamma}^2 \cdot \mathbf{d}_{4,1}^{P_2} = \binom{1}{1}$ bzw. $\mathbf{q}_2 = \mathbf{\Gamma}^2 \cdot \mathbf{d}_{4,2}^{P_2} = \binom{1}{-1}$ und lokaler Speicher wäre sinnvoll. Im Gegensatz dazu erfolgt die Datenwiederverwendung bei den partitionierten Abhängigkeitsvektoren $\mathbf{d}_{4,3}^{\kappa_2}$ und $\mathbf{d}_{4,4}^{\kappa_2}$ erst nach 9 bzw. 10 Takten. Die lokale Speicherung dieser Werte im Rechenfeld würde einen hohen Speicherbedarf in den PEs erfordern, weshalb diese Daten bis zur Wiederverwendung im globalen Speicher abgelegt werden.

Regel 7.3 (Kurze Datentransfers im Rechenfeld)
Durch die Partitionierung eines Anhängigkeitsvektors $\mathbf{d}$ entstehen meist mehrere partitionierte Abhängigkeitsvektoren $\mathbf{d}_{i_1,\ldots,i_4}^{\kappa}$. Ist der Teilvektor $\widehat{\mathbf{d}}_{i_1,\ldots,i_4}^{P_2}$ des partitionierten Abhängigkeitsvektors $\mathbf{d}_{i_1,\ldots,i_4}^{\kappa}$ ein Nullvektor, so wird der Datentransfer meist im Rechenfeld realisiert, denn die Datenwiederverwendung erfolgt in einem zeitlich geringen Abstand.

Hat der Teilvektor $\widehat{\mathbf{d}}_{i_1,\ldots,i_4}^{P_2}$ nur ein Nicht-Null-Element, ist dieses 1 oder -1 und befindet es sich an der Position $j = |\alpha_1^{P_2}|$, so handelt es sich um eine Datenwiederverwendung in der ersten Abarbeitungsrichtung der LP-2-Partitionen. Der zeitliche Abstand der Abarbeitung dieser Partitionen ist meist gering. Doch der räumliche Datentransfer in großen Rechenfeldern kann sehr lang sein. Der Grund hierfür ist, dass die Daten von einem Prozessor an einem Rand des Rechenfeldes zu einem Prozessor am gegenüberliegenden Rand des Rechenfeldes transferiert werden müssen. Statt des Transports der Daten durch das gesamte Rechenfeld kann es hier sinnvoll sein, die Daten auf der einen Seite des Rechenfeldes auszulesen und sie unmittelbar auf der anderen Seite des Rechenfeldes wieder einzuspeisen [Sie08].

Nicht immer ist der benötigte Datenkanal $\mathbf{q} = \mathbf{d}^{P_2}$ in der Zielarchitektur vorhanden. Meist ist es aber möglich, durch Linearkombination existierender Datenkanäle den gewünschten Datentransfer zu realisieren [SM06].

Datenwiederverwendung zwischen Partitionen

Ein weiterer Aspekt der Auswahl der Partitionsmatrix $\mathbf{\Theta}^{P_2}$ ist der Grad an Datenwiederverwendung in der ersten Abarbeitungsrichtung. Da die Datenwiederverwendung zwischen Partitionen ein allgemeines Thema bei der Auswahl von Partitionierungsparame-

tern ist, soll nachfolgend eine allgemeine Strategie zur Bestimmung der Datenwiederverwendung zwischen Partitionen vorgestellt werden. Für die Bestimmung der Datenwiederverwendung soll zuerst der Datentransfer bei einem einfach partitionierten Iterationsraum ermittelt werden. Anschließend wird dieses Ergebnis verallgemeinert.

Betrachten wir die Variable $y_s[\mathbf{i} - \mathbf{d}^{k_s}_{s\,s'_t}]$, deren Instanzen in Zuweisung s erzeugt und in Teilzuweisung s'_t verwendet werden. Der gemeinsame Iterationsraum $\mathcal{I}$ des Algorithmus wird einstufig mit der Partitionsmatrix $\boldsymbol{\Theta}$ partitioniert. Jede Iteration $\mathbf{i} = \vartheta(\mathbf{i}^\kappa)$ wird nun durch $\mathbf{i}^\kappa = \left(\begin{smallmatrix}\boldsymbol{\kappa}\\ \widehat{\boldsymbol{\kappa}}\end{smallmatrix}\right)$, siehe Definition 4.2 auf Seite 40, dargestellt, wobei $\boldsymbol{\kappa}$ die Iteration innerhalb der Partition angibt und $\widehat{\boldsymbol{\kappa}}$ die Partition beschreibt. Durch die Partitionierung des Abhängigkeitsvektors $\mathbf{d} = \mathbf{d}^{k_s}_{s\,s'_t}$, siehe Abschnitt 4.2.2 auf Seite 44, entstehen die partitionierten Abhängigkeitsvektoren $\mathbf{d}^\kappa_i = \left(\mathbf{d}^{\boldsymbol{\Theta}}_i\ \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i\right)^t$ mit den beiden Teilvektoren $\mathbf{d}^{\boldsymbol{\Theta}}_i$ und $\widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i$, sowie die Mengen $\mathcal{K}^{\sigma(\mathbf{d})}_i$. Die Menge $\mathcal{K}^{\sigma(\mathbf{d})}_i = \{\boldsymbol{\kappa} \mid \ldots\}$ enthält die Iterationen $\boldsymbol{\kappa}$ einer Partition $\widehat{\boldsymbol{\kappa}}_\sigma$, deren zugehörige Instanzen der Variable $y_s[\vartheta\left(\mathbf{i}^\kappa = \left(\begin{smallmatrix}\boldsymbol{\kappa}\\ \widehat{\boldsymbol{\kappa}}_\sigma\end{smallmatrix}\right)\right)]$ von Partition $\widehat{\boldsymbol{\kappa}}_\sigma$ nach Partition $\widehat{\boldsymbol{\kappa}}_\delta = \widehat{\boldsymbol{\kappa}}_\sigma + \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i$ transferiert werden. Mit anderen Worten: Der äußere Teilvektor $\widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i$ gibt an, zwischen welchen Partitionen $\widehat{\boldsymbol{\kappa}}$ Instanzen von Variablen transferiert werden und die Menge $\mathcal{K}^{\sigma(\mathbf{d})}_i$ enthält die Iterationen $\boldsymbol{\kappa}$, welche entsprechend $\widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i$ transferiert werden.

Gibt es mehrere Datenabhängigkeiten $e \in \mathcal{E}$ mit der Quellzuweisung $\sigma(e) = s$, welche zu partitionierten Datenabhängigkeiten zwischen zwei Partitionen $\widehat{\boldsymbol{\kappa}}_\sigma$ und $\widehat{\boldsymbol{\kappa}}_\delta = \widehat{\boldsymbol{\kappa}}_\sigma + \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}$ führen, so müssen die entsprechenden Instanzen der Variablen $y_{[}\vartheta(\mathbf{i}^\kappa)]$ nur einmal transferiert und danach für die Wiederverwendung gespeichert werden. Um diesen Sachverhalt zu beschreiben, definieren wir die Menge

$$\mathcal{U}(s, \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}) = \bigcup_{\substack{\forall e \in \mathcal{E},\ \sigma(e)=s,\\ \mathbf{d}(e) \to \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i = \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}}} \mathcal{K}^{\sigma(\mathbf{d}(e))}_i, \tag{7.41}$$

wobei $\mathbf{d}(e) \to \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i = \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}$ bedeutet: Durch die Partitionierung des Abhängigkeitsvektors $\mathbf{d}(e)$ entsteht der äußere Abhängigkeitsvektor $\widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_i$, welcher gleich dem gesuchten Abhängigkeitsvektor $\widehat{\mathbf{d}}^{\boldsymbol{\Theta}}$ ist.

Diese Menge $\mathcal{U}(s, \widehat{\mathbf{d}}^{\boldsymbol{\Theta}})$ enthält diejenigen Iterationen $\boldsymbol{\kappa} \in \mathcal{K}$ einer Partition $\widehat{\boldsymbol{\kappa}}_\sigma$, deren Instanzen der Variable $y_s[\vartheta\left(\mathbf{i}^\kappa = \left(\begin{smallmatrix}\boldsymbol{\kappa}\\ \widehat{\boldsymbol{\kappa}}_\sigma\end{smallmatrix}\right)\right)]$ von Partition $\widehat{\boldsymbol{\kappa}}_\sigma$ zu Partition $\widehat{\boldsymbol{\kappa}}_\delta = \widehat{\boldsymbol{\kappa}}_\sigma + \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}$ transferiert werden müssen. Diese Menge bezeichnen wir als Transfermenge.

Beispiel 7.16 (Transfermenge)
Ein Algorithmus habe zwei Abhängigkeitsvektoren $\mathbf{d}(e_1) = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *und* $\mathbf{d}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *mit* $\sigma(e_1) = \sigma(e_2) = s$. *Der Iterationsraum* $\mathcal{I}$ *des Algorithmus wird mit* $\boldsymbol{\Theta} = \left(\begin{smallmatrix}3&0\\0&2\end{smallmatrix}\right)$ *partitioniert.*

Durch die Partitionierung zerfällt der Abhängigkeitsvektor $\mathbf{d}(e_1) = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *in vier partitionierte Abhängigkeitsteilvektoren mit den äußeren Teilvektoren*

$$\widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_1(e_1) = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \qquad \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_2(e_1) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \qquad \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_3(e_1) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \qquad \widehat{\mathbf{d}}^{\boldsymbol{\Theta}}_4(e_1) = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$$

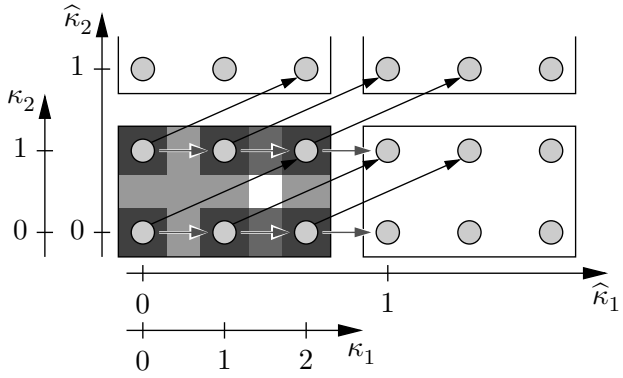


Abbildung 7.8: Datentransfer bei einer Variable mit zwei verschiedenen Datentransfers

und den Räumen

$$\mathcal{K}_1^{\sigma(d(e_1))} = \{\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\},\quad \mathcal{K}_2^{\sigma(d(e_1))} = \{\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\},\quad \mathcal{K}_3^{\sigma(d(e_1))} = \{\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\},\quad \mathcal{K}_4^{\sigma(d(e_1))} = \{\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right),\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)\}.$$

Für den Abhängigkeitsvektor $\mathbf{d}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *erhalten wir zwei partitionierte Abhängigkeitsvektoren mit den äußeren Teilvektoren*

$$\widehat{\mathbf{d}}_1^{\mathbf{\Theta}}(e_2) = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \qquad \widehat{\mathbf{d}}_2^{\mathbf{\Theta}}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$$

und den Räumen

$$\mathcal{K}_1^{\sigma(d(e_2))} = \{\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right),\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\}, \qquad \mathcal{K}_2^{\sigma(d(e_2))} = \{\left(\begin{smallmatrix}2\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)\}.$$

In Abbildung 7.8 sind die Teilräume $\mathcal{K}_i^{\sigma(\mathbf{d}(e))}$ *für die zwei Abhängigkeitsvektoren* $\mathbf{d}(e_1) = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$ *und* $\mathbf{d}(e_2) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *in verschiedenen Grautönen dargestellt.*

Da vier verschiedene äußere Teilvektoren $\widehat{\mathbf{d}}^{\mathbf{\Theta}} = \{\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right),\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\}$ *entstanden sind, können vier Transfermengen* $\mathcal{U}(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}})$ *bestimmt werden.*

Die Menge $\mathcal{U}(s, \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)) = \mathcal{K}_2^{\sigma(d(e_1))} \cup \mathcal{K}_2^{\sigma(d(e_2))} = \{\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}2\\0\end{smallmatrix}\right),\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)\}$ *enthält drei Elemente:*

- *Die Instanzen der Variablen* $\mathbf{y}_s[\vartheta(\mathbf{i}^{\kappa} = \left(\begin{smallmatrix}\boldsymbol{\kappa}\\ \widehat{\boldsymbol{\kappa}}_\sigma\end{smallmatrix}\right))]$ *mit* $\boldsymbol{\kappa} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *und* $\boldsymbol{\kappa} = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$*, welche in der Partition* $\widehat{\boldsymbol{\kappa}}_\delta = \widehat{\boldsymbol{\kappa}}_\sigma + \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *wiederverwendet werden und*
- *Die Instanzen der Variablen* $\mathbf{y}_s[\vartheta(\mathbf{i}^{\kappa} = \left(\begin{smallmatrix}\boldsymbol{\kappa}\\ \widehat{\boldsymbol{\kappa}}_\sigma\end{smallmatrix}\right))]$ *mit* $\boldsymbol{\kappa} = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$ *und* $\boldsymbol{\kappa} = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$*, welche in der Partition* $\widehat{\boldsymbol{\kappa}}_\delta = \widehat{\boldsymbol{\kappa}}_\sigma + \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *wiederverwendet werden.*

Die Instanz der Variablen $\mathbf{y}_s[\vartheta(\mathbf{i}^{\kappa} = \left(\begin{smallmatrix}\boldsymbol{\kappa}\\ \widehat{\boldsymbol{\kappa}}_\sigma\end{smallmatrix}\right))]$ *mit* $\boldsymbol{\kappa} = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$ *muss nur einmal zwischen den Partitionen transformiert werden.*

Die Anzahl an Instanzen aller Variablen $\mathbf{y}_s[\mathbf{i}] = \mathbf{y}_s[\vartheta(\mathbf{i}^{\kappa})]$, $s \in \mathcal{S}$ des Algorithmus, welche zwischen Partitionen entsprechend Teilvektor $\widehat{\mathbf{d}}^{\mathbf{\Theta}}$ transferiert werden müssen, kann mittels der Transferrate

$$U(\widehat{\mathbf{d}}^{\mathbf{\Theta}}) = \sum_{\forall s \in \mathcal{S}} |\mathcal{U}(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}})|, \tag{7.42}$$

bestimmt werden, wobei $|\mathcal{U}(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}})|$ die Anzahl an Elementen der Menge $\mathcal{U}(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}})$ ist.

Wird der Iterationsraum mehrfach partitioniert, so können die folgenden Transfermengen bestimmt werden:

$$\mathcal{U}^b(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k}) = \bigcup_{\substack{\forall e \in \mathcal{E},\ \sigma(e)=s,\\ \mathbf{d}(e) \to \widehat{\mathbf{d}}^{\mathbf{\Theta}}_{i_1,i_2,\ldots i_k} = \widehat{\mathbf{d}}^{\mathbf{\Theta}^k}}} \left(\mathcal{K}^{\sigma(\mathbf{d}(e))}_{i_1,i_2,\ldots,i_b} \oplus \mathcal{K}^{\sigma(\mathbf{d}(e))}_{i_1,i_2,\ldots,i_{b+1}} \oplus \cdots \oplus \mathcal{K}^{\sigma(\mathbf{d}(e))}_{i_1,i_2,\ldots,i_k} \right), \tag{7.43}$$

Die Elemente $\left(\boldsymbol{\kappa}^b, \boldsymbol{\kappa}^{b+1}, \ldots, \boldsymbol{\kappa}^k\right)^t$ dieser Menge sind Teilvektoren der Iterationen $\mathbf{i}^\kappa = \left(\boldsymbol{\kappa}^1, \ldots, \boldsymbol{\kappa}^b, \boldsymbol{\kappa}^{b+1}, \ldots, \boldsymbol{\kappa}^k, \ldots, \boldsymbol{\kappa}^p, \widehat{\boldsymbol{\kappa}}^p\right)^t$. Die Menge $\mathcal{U}^b(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k})$ enthält die Elemente, welche bezüglich der Iteration $\boldsymbol{\kappa}^b$ zwischen den Partitionen $\widehat{\boldsymbol{\kappa}}^k_\sigma$ und $\widehat{\boldsymbol{\kappa}}^k_\delta$, gegeben durch Abhängigkeitsvektor $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k} = \widehat{\boldsymbol{\kappa}}^k_\delta - \widehat{\boldsymbol{\kappa}}^k_\sigma$, transferiert werden.

Die Anzahl an Instanzen bezüglich der Iteration $\boldsymbol{\kappa}^b$ aller Variablen $\mathbf{y}_s[\mathbf{i}] = \mathbf{y}_s[\vartheta(\mathbf{i}^\kappa)]$, $s \in \mathcal{S}$ des Algorithmus, welche zwischen Partitionen der Partitionsstufe k entsprechend Teilvektor $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}$ transferiert werden, kann nun mittels der Transferrate

$$U^b(\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}) = \sum_{\forall s \in \mathcal{S}} |\mathcal{U}^b(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k})|, \tag{7.44}$$

ermittelt werden.

Betrachten wir als Beispiel die zweistuftige modifizierte Copartitionierung, wie wir sie für die Abbildung eines Algorithmus auf ein Rechenfeld mit TWP nutzen. Für die Partitionierungsmatrizen gelte: $\mathbf{\Theta}^{S_1} = \mathbf{E}$ sowie $\mathbf{\Theta}^{P_1} \neq \mathbf{E}$, $\mathbf{\Theta}^{S_2} \neq \mathbf{E}$ und $\mathbf{\Theta}^{P_2} \neq \mathbf{E}$. So können unter anderem die folgenden Mengen ermittelt werden:

- $\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{S_1})$: Transfer von Teilwörtern zwischen LS-1-Partitionen. Da die LS-1-Partition aufgrund $\mathbf{\Theta}^{S_1} = \mathbf{E}$ immer nur ein Element hat, wird maximal ein Teilwort von einer LS-1-Partition zu einer anderen LS-1-Partition transferiert.
- $\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{P_1})$: Transfer von Teilwörtern zwischen LP-1-Partitionen. Die LP-1-Partition entspricht unserem Datenwort voller Breite (DvB) bzw. dem Prozessorelement, welches die DvBs verarbeitet.
- $\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{S_2})$: Transfer von Teilwörtern zwischen LS-2-Partitionen, das heißt zwischen den Prozessorelementen des Rechenfeldes.
- $\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{P_2})$: Transfer von Teilwörtern zwischen LP-2-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$. Eine Iteration $\widehat{\boldsymbol{\kappa}}^{P_2}$ enthält einen Satz von Teilwörtern, welche auf dem Rechenfeld quasi parallel bestimmt werden.
- $\mathcal{U}^{S_2}(s, \widehat{\mathbf{d}}^{S_2})$: Transfer von DvBs zwischen LS-2-Partitionen, das heißt zwischen den Prozessorelementen des Rechenfeldes.
- $\mathcal{U}^{S_2}(s, \widehat{\mathbf{d}}^{P_2})$: Transfer von DvBs zwischen LP-2-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$. Eine Iteration $\widehat{\boldsymbol{\kappa}}^{P_2}$ enthält einen Satz von DvBs, welche auf dem Rechenfeld quasi parallel bestimmt werden.

$$\mathcal{K}_{1}^{S_1,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{S_1} \;\middle|\; \begin{smallmatrix} \kappa_1^{S_1}=0 \\ \kappa_2^{S_1}=0 \end{smallmatrix} \right\}$$

$$\mathcal{K}_{1,1}^{P_1,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_1} \;\middle|\; \begin{smallmatrix} 0\leq\kappa_1^{P_1}\leq 2 \\ \kappa_2^{P_1}=0 \end{smallmatrix} \right\}$$

$$\mathcal{K}_{1,2}^{P_1,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_1} \;\middle|\; \begin{smallmatrix} \kappa_1^{P_1}=3 \\ \kappa_2^{P_1}=0 \end{smallmatrix} \right\}$$

$$\mathcal{K}_{1,1,1}^{S_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{S_2} \;\middle|\; \begin{smallmatrix} 0\leq\kappa_1^{S_2}\leq 2 \\ \kappa_2^{S_2}=0 \end{smallmatrix} \right\}$$

$$\mathcal{K}_{1,2,1}^{S_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{S_2} \;\middle|\; \begin{smallmatrix} 0\leq\kappa_1^{S_2}\leq 1 \\ \kappa_2^{S_2}=0 \end{smallmatrix} \right\}$$

$$\mathcal{K}_{1,2,2}^{S_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{S_2} \;\middle|\; \begin{smallmatrix} 0\kappa_1^{S_2}=2 \\ \kappa_2^{S_2}=0 \end{smallmatrix} \right\}$$

$$\mathcal{K}_{1,1,1,1}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} 0\leq\kappa_1^{P_2}\leq 1 \\ \kappa_2^{P_2}=0 \end{smallmatrix} \right\} \quad \mathcal{K}_{1,2,1,1}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} 0\leq\kappa_1^{P_2}\leq 1 \\ \kappa_2^{P_2}=0 \end{smallmatrix} \right\} \quad \mathcal{K}_{1,2,2,1}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} \kappa_1^{P_2}=0 \\ \kappa_2^{P_2}=0 \end{smallmatrix} \right\} \quad \mathcal{K}_{1,2,2,3}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} \kappa_1^{P_2}=0 \\ \kappa_2^{P_2}=1 \end{smallmatrix} \right\}$$

$$\mathcal{K}_{1,1,1,2}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} 0\leq\kappa_1^{P_2}\leq 1 \\ \kappa_2^{P_2}=1 \end{smallmatrix} \right\} \quad \mathcal{K}_{1,2,1,2}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} 0\leq\kappa_1^{P_2}\leq 1 \\ \kappa_2^{P_2}=1 \end{smallmatrix} \right\} \quad \mathcal{K}_{1,2,2,2}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} \kappa_1^{P_2}=1 \\ \kappa_2^{P_2}=0 \end{smallmatrix} \right\} \quad \mathcal{K}_{1,2,2,4}^{P_2,\sigma(\mathbf{d})} = \left\{ \boldsymbol{\kappa}^{P_2} \;\middle|\; \begin{smallmatrix} \kappa_1^{P_2}=1 \\ \kappa_2^{P_2}=1 \end{smallmatrix} \right\}$$

Abbildung 7.9: BPA eines zweifach copartitionierten Abhängigkeitsvektors

Beispiel 7.17 (Datentransfer zwischen Partitionen)
Ein Algorithmus besitzt eine Datenabhängigkeit $e \in \mathcal{E}$ mit der Quellzuweisung $\sigma(e) = s$ und dem Abhängigkeitsvektor $\mathbf{d}(e) = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$. Diese Datenabhängigkeit sei die einzige Datenabhängigkeit e mit der Quellzuweisung $\sigma(e) = s$. Der Iterationsraum des Algorithmus und damit der Abhängigkeitsvektor $\mathbf{d}(e)$ werden mit $\mathbf{\Theta}^{S_1} = \mathbf{E}$, $\mathbf{\Theta}^{P_1} = \left(\begin{smallmatrix}4&0\\0&1\end{smallmatrix}\right)$, $\mathbf{\Theta}^{S_2} = \left(\begin{smallmatrix}3&0\\0&1\end{smallmatrix}\right)$ und $\mathbf{\Theta}^{P_2} = \left(\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right)$ partitioniert. Abbildung 7.9 zeigt den Baum der partitionierten Abhängigkeitsvektoren (BPA) des zweifach copartitionierten Abhängigkeitsvektors $\mathbf{d}(e) = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ mit den zugehörigen Teilräumen $\mathcal{K}^{k,\sigma(\mathbf{d})}_{i_1,\ldots,i_k}$. Daraus ergeben sich unter anderem die folgenden Transfermengen:

$$\mathcal{U}^{S_1}\left(s, \widehat{\mathbf{d}}^{S_1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) = \left\{\boldsymbol{\kappa}^{S_1} \middle| \ldots\right\} = \emptyset, \tag{7.45}$$

$$\mathcal{U}^{S_1}\left(s, \widehat{\mathbf{d}}^{P_1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) = \left\{\begin{pmatrix}\boldsymbol{\kappa}^{S_1}\\ \boldsymbol{\kappa}^{P_1}\end{pmatrix} \middle| \ldots\right\} = \left\{\begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\end{pmatrix}\right\}, \tag{7.46}$$

$$\begin{aligned}\mathcal{U}^{S_1}\left(s, \widehat{\mathbf{d}}^{S_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) = \left\{\begin{pmatrix}\boldsymbol{\kappa}^{S_1}\\ \boldsymbol{\kappa}^{P_1}\\ \boldsymbol{\kappa}^{S_2}\end{pmatrix} \middle| \ldots\right\} = \Bigg\{&\begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\end{pmatrix}\\ &\begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{pmatrix}\Bigg\},\end{aligned} \tag{7.47}$$

$$\begin{aligned}\mathcal{U}^{S_1}\left(s, \widehat{\mathbf{d}}^{P_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) &= \left\{\begin{pmatrix}\boldsymbol{\kappa}^{S_1}\\ \boldsymbol{\kappa}^{P_1}\\ \boldsymbol{\kappa}^{S_2}\\ \boldsymbol{\kappa}^{P_2}\end{pmatrix} \middle| \ldots\right\}\\ &= \Bigg\{\begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix}\\ &\quad \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix}\\ &\quad \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix}\\ &\quad \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{pmatrix} \begin{pmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}3\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{pmatrix}\Bigg\},\end{aligned} \tag{7.48}$$

$$\mathcal{U}^{S_2}\left(s, \widehat{\mathbf{d}}^{S_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) = \left\{\boldsymbol{\kappa}^{S_2} \middle| \ldots\right\} = \left\{\begin{pmatrix}0\\0\end{pmatrix} \begin{pmatrix}1\\0\end{pmatrix} \begin{pmatrix}2\\0\end{pmatrix}\right\}, \tag{7.49}$$

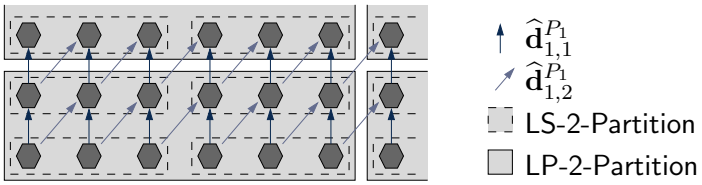


Abbildung 7.10: Zweite Stufe eines zweifach copartitionierten Iterationsraums

$$\mathcal{U}^{S_2}\left(s, \widehat{\mathbf{d}}^{P_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) = \left\{ \left(\begin{smallmatrix}\boldsymbol{\kappa}^{S_2}\\ \boldsymbol{\kappa}^{P_2}\end{smallmatrix}\right) \middle| \ldots \right\} = \left\{ \left(\begin{smallmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{smallmatrix}\right) \left(\begin{smallmatrix}\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{smallmatrix}\right) \left(\begin{smallmatrix}\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\end{smallmatrix}\right) \left(\begin{smallmatrix}\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{smallmatrix}\right) \left(\begin{smallmatrix}\left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\end{smallmatrix}\right) \left(\begin{smallmatrix}\left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)\\ \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)\end{smallmatrix}\right) \right\}. \quad (7.50)$$

Folgende Aussagen lassen sich daraus ableiten:

- $|\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{S_1})| = 0$, *das heißt, kein Teilwort* $y_s[.]$ *aus Partition* $\widehat{\boldsymbol{\kappa}}^{S_1}$ *wird in Partition* $\widehat{\boldsymbol{\kappa}}^{S_1} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *benötigt.*
- $|\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{P_1})| = 3$ *Teilwörter des DvBs* $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^{P_1}]$, *der Partition* $\widehat{\boldsymbol{\kappa}}^{P_1}$, *werden in DvB* $\mathbf{y}_s[\widehat{\boldsymbol{\kappa}}^{P_1} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)]$, *der Partition* $\widehat{\boldsymbol{\kappa}}^{P_1} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, *wiederverwendet.*
- $|\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{S_2})| = 11$ *Teilwörter der Partition* $\widehat{\boldsymbol{\kappa}}^{S_2}$ *werden in Partition* $\widehat{\boldsymbol{\kappa}}^{S_2} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *wiederverwendet.*
- $|\mathcal{U}^{S_1}(s, \widehat{\mathbf{d}}^{P_2})| = 23$ *Teilwörter der Partition* $\widehat{\boldsymbol{\kappa}}^{P_2}$ *werden in Partition* $\widehat{\boldsymbol{\kappa}}^{P_2} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *wiederverwendet.*
- $|\mathcal{U}^{S_2}(s, \widehat{\mathbf{d}}^{S_2})| = 3$ *DvBs der Partition* $\widehat{\boldsymbol{\kappa}}^{S_2}$ *werden in Partition* $\widehat{\boldsymbol{\kappa}}^{S_2} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *wiederverwendet.*
- $|\mathcal{U}^{S_2}(s, \widehat{\mathbf{d}}^{P_2})| = 6$ *DvBs der Partition* $\widehat{\boldsymbol{\kappa}}^{P_2}$ *werden in Partition* $\widehat{\boldsymbol{\kappa}}^{P_2} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *wiederverwendet.*

In Abbildung 7.10 ist die zweite Stufe des zweifach copartitionierten Iterationsraums dargestellt. Durch die erste Copartitionierungsstufe mit $\boldsymbol{\Theta}^{S_1} = \mathbf{E}$ *und* $\boldsymbol{\Theta}^{P_1} = \left(\begin{smallmatrix}4&0\\0&1\end{smallmatrix}\right)$ *ist der Abhängigkeitsvektor* $\mathbf{d} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *in zwei partitionierte Abhängigkeitsvektoren zerfallen. Die äußeren Teilvektoren* $\widehat{\mathbf{d}}^{P_1}_{1,1} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ *und* $\widehat{\mathbf{d}}^{P_1}_{1,2} = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ *beschreiben den Transfer der DvBs zwischen den Iterationen* $\widehat{\boldsymbol{\kappa}}^{P_1}$. *Betrachten wir den Transfer der DvBs zwischen zwei LS-2-Partitionen* $\widehat{\boldsymbol{\kappa}}^{S_2}_{\sigma}$ *und* $\widehat{\boldsymbol{\kappa}}^{S_2}_{\delta} = \widehat{\boldsymbol{\kappa}}^{S_2}_{\sigma} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, *so erkennt man, dass zwischen den beiden Partitionen drei DvBs hervorgerufen durch Abhängigkeitsvektor* $\widehat{\mathbf{d}}^{P_1}_{1,1}$ *und zwei DvBs hervorgerufen durch Abhängigkeitsvektor* $\widehat{\mathbf{d}}^{P_1}_{1,2}$ *transferiert werden müssen. Da der durch den Abhängigkeitsvektor* $\widehat{\mathbf{d}}^{P_1}_{1,2}$ *beschriebene Datentransfer zwischen den Partitionen bereits durch den Datentransfer gegeben durch Abhängigkeitsvektor* $\widehat{\mathbf{d}}^{P_1}_{1,1}$ *realisiert wird, müssen effektiv nur* $|\mathcal{U}^{S_2}(s, \widehat{\mathbf{d}}^{S_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right))| = 3$ *DvBs zwischen den beiden Partitionen transferiert werden. Ähnlich verhält es sich beim Datentransfer zwischen den LP-2-Partitionen* $\widehat{\boldsymbol{\kappa}}^{P_2}_{\sigma}$

und $\widehat{\boldsymbol{\kappa}}_{\delta}^{P_2} = \widehat{\boldsymbol{\kappa}}_{\sigma}^{P_2} + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$*, sodass der effektive Datentransfer nur* $|\mathcal{U}^{S_2}\left(s, \widehat{\mathbf{d}}^{P_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right)| = 6$ *beträgt.*

Dieses Beispiel illustriert, dass die durch die Teilwortparallelisierung entstehenden mehrfachen Transfers von DvBs durch die Verwendung der Transfermengen auf das tatsächliche Maß wieder reduziert werden können.

Wie in Kapitel 6 ab Seite 79 gezeigt wurde, können durch die Teilwortparallelisierung neue Zuweisungen mit entsprechenden Datentransfers entstehen. Werden die Datentransfers auf Basis des Ausgangsalgorithmus ermittelt, so werden die neuen Packzuweisungen nicht mit einbezogen.

Soll der Datentransfer zwischen Partitionen ermittelt werden und sind unausgerichtete Speicherzugriffe, siehe Abschnitt 2.4 ab Seite 16, möglich, so sind die Berechnungen auf Basis des Ausgangsalgorithmus ausreichend. In diesem Fall finden die Datentransfers aufgrund der neuen Zuweisungen innerhalb der Iterationen statt und es entstehen keine neuen Packoperationen für die Daten Ein- und Ausgabe.

Sind nur ausgerichtete Speicherzugriffe möglich, so muss der Algorithmus vor der Bestimmung der Datentransfers in einen teilwortparallelen Algorithmus umgewandelt werden. Die Ermittlung der Datentransfers erfolgt dann auf Basis der Zuweisungen und der Abhängigkeitsvektoren des teilwortparallelen Algorithmus.

Im Folgenden gehen wir von unausgerichteten Speicherzugriffen aus.

Ausnutzung des Speichers in den Prozessorelementen

Durch die Vergrößerung der LS-2-Partition mittels $\boldsymbol{\Theta}^{S_2}$ findet eine lokale, sequenzielle Abarbeitung mehrerer Iterationen $\mathbf{i}^{\kappa}$ in einem Prozessorelement $\mathbf{p}$ statt. Damit steigt der Bedarf an lokalen Speicher [Eck01] und die Datenrate für den Datentransfer zwischen den PEs sinkt.

Die Bestimmung des Speicherbedarf in den Prozessorelementen ist sehr schwierig, denn der Bedarf hängt nicht nur von der Partitionierung des Iterationsraums sondern auch vom Ablaufplan ab. Deshalb soll an dieser Stelle nur ein mögliches Vorgehen und dessen Folgen gezeigt werden.

Regel 7.4 (Dimension der LS-2-Partition)
Die Ausdehnung der LS-2-Partition ($\boldsymbol{\Theta}^{S_2}$*) sollte auf die Raumrichtungen beschränkt werden, in denen die LP-2-Partition eine Ausdehnung größer eins haben.*

Durch diese Einschränkung wird die sequenzielle Abarbeitung der Iterationen $\mathbf{i}^{\kappa}$ einer LS-2-Partition auf die Raumrichtungen eingeschränkt, welche ohne LS-2-Partition vorrangig parallel verarbeitet werden würden. Eine sequenzielle Abarbeitung in den anderen Raumrichtungen wird durch die sequenzielle Abarbeitung der LP-2-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ erreicht.

Ist die Ausdehnung der LS-2-Partition nur in einer Raumrichtung größer eins (eindimensionale LS-2-Partition), so ist der Speicherbedarf aufgrund der Datenwiederverwendung innerhalb der LS-2- und der LP-2-Partition meist sehr gering. Der zusätzliche Speicherbedarf ergibt sich vielmehr aus der Datenwiederverwendung in der ersten Abarbeitungsrichtung der LP-2-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$, welche durch das erste Element $\widehat{\alpha}_1^{P_2}$ des Abfolgevektors $\widehat{\boldsymbol{\alpha}}^{P_2}$ festgelegt wird.

Bei einer LS-2-Partition mit einer Ausdehnung größer eins in zwei Raumrichtungen (zweidimensionale LS-2-Partition) kann sich der Speicherbedarf aufgrund der Datenwiederverwendung innerhalb der LS-2-Partition zusätzlich weiter erhöhen.

Die Datenwiederverwendung zwischen den äußeren LP-2-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ kann mittels der Transferrate $U^{S_2}(\widehat{\mathbf{d}}^{P_2})$, siehe (7.44) auf Seite 178, bestimmt werden. Als Vektor $\widehat{\mathbf{d}}^{P_2}$ für die Bestimmung der Datenwiederverwendung in der zeitlich nächsten Copartition muss $\widehat{\mathbf{d}}^{P_2} = \text{sign}(\widehat{\alpha}_1^{P_2}) \cdot \mathbf{e}_{|\widehat{\alpha}_1^{P_2}|}$ gewählt werden, wobei $\mathbf{e}_i$ die i-te Spalte der Einheitsmatrix $\mathbf{E}$ ist. Der Speicherbedarf im Rechenfeld muss hierfür $r^{\text{RF}} = U^{S_2}\left(\widehat{\mathbf{d}}^{P_2}\right)$ mit $\widehat{\mathbf{d}}^{P_2} = \text{sign}(\widehat{\alpha}_1^{P_2}) \cdot \mathbf{e}_{|\widehat{\alpha}_1^{P_2}|}$ sein.

Sollen die Daten, die in der zeitlich übernächsten Copartition verwendet werden, ebenfalls lokal gespeichert werden, so ermitteln wir für jede Variable $\mathbf{y}_s[.]$ die Transfermengen $\mathcal{U}^{S_2}(s, \widehat{\mathbf{d}}^{P_2})$ und $\mathcal{U}^{S_2}(s, 2 \cdot \widehat{\mathbf{d}}^{P_2})$ mit $\widehat{\mathbf{d}}^{P_2} = \text{sign}(\widehat{\alpha}_1^{P_2}) \cdot \mathbf{e}_{|\widehat{\alpha}_1^{P_2}|}$.

Für die Daten, die in der übernächsten Copartition verwendet werden, ist ein Speicherbedarf von $2 \cdot \left|\mathcal{U}^{S_2}(s, 2 \cdot \widehat{\mathbf{d}}^{P_2})\right|$ notwendig. Da ein Teil der Daten, die in der nächsten Copartition verwendet werden, auch in der übernächsten Copartition verwendet werden, ist für diese Daten kein zusätzlicher Speicherbedarf notwendig. Der Speicherbedarf für die Daten die ausschließlich in der nächsten Copartition verwendet werden, beträgt somit $\left|\mathcal{U}^{S_2}\left(s, \widehat{\mathbf{d}}^{P_2}\right) \backslash \mathcal{U}^{S_2}\left(s, 2 \cdot \widehat{\mathbf{d}}^{P_2}\right)\right|$.

Verallgemeinern wir die Bestimmung des Speicherbedarfs im Rechenfeld, so erhalten wir:

$$r^{\text{RF}}(h^2, \widehat{\mathbf{d}}^{P_2}) = \sum_{s \in \mathcal{S}} \sum_{i=1}^{h} i \cdot \left| \mathcal{U}^{S_2}\left(s, i \cdot \widehat{\mathbf{d}}^{P_2}\right) \Big\backslash \bigcup_{j=i+1}^{h} \mathcal{U}^{S_2}\left(s, j \cdot \widehat{\mathbf{d}}^{P_2}\right) \right|,$$

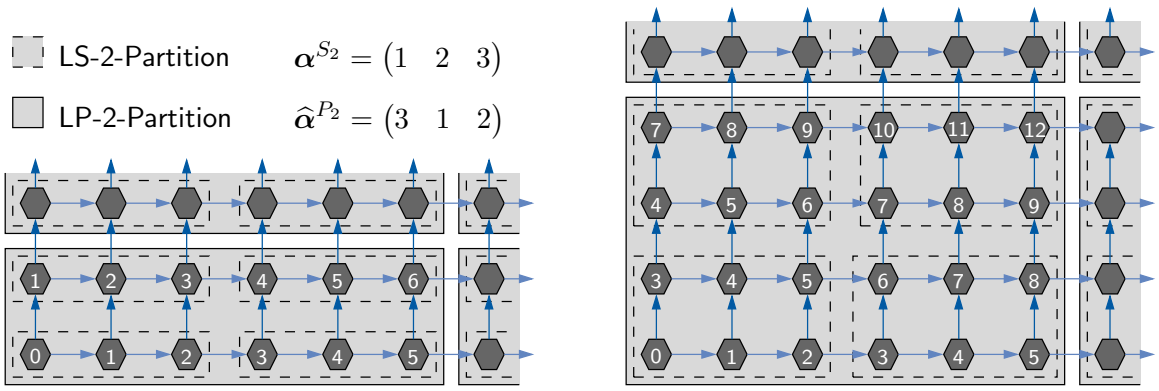
$$\widehat{\mathbf{d}}^{P_2} = \text{sign}(\widehat{\alpha}_1^{P_2}) \cdot \mathbf{e}_{|\widehat{\alpha}_1^{P_2}|}, \quad (7.51)$$

wobei $\widehat{\mathbf{d}}^{P_2}$ die erste Abarbeitungsrichtung angibt und h^2 die Anzahl der nachfolgenden Copartition ist, für die die Daten zur Datenwiederverwendung im Rechenfeld gespeichert werden sollen.

Die maximale Anzahl $h^p_{\max}(j)$ an nachfolgenden Copartitionen, welche für die Speicherung der Daten beachtet werden könnten, kann mit:

$$h^p_{\max}(j) = \left\lceil \frac{\max_{e \in \mathcal{E}} \text{sign}(j) \cdot d_{|j|}(e)}{\prod_{i=1}^{p} \vartheta^{S_i}_{|j|} \cdot \vartheta^{P_i}_{|j|}} \right\rceil, \qquad j = \widehat{\alpha}_1^{P_2} \quad (7.52)$$

ermittelt werden, wobei j die erste Abarbeitungsrichtung ist und p die Copartitionierungsstufe. Im Allgemeinen ist diese Zahl sehr klein, da $|d_i(e)|$ sehr klein ist.



(a) eindimensionale LS-2-Partition
Datenwiederverwendung in i_3-Richtung: 3

(b) zweidimensionale LS-2-Partition
Datenwiederverwendung in i_3-Richtung: 6

Abbildung 7.11: Registerbedarf bei LS-2-Partitionierung

Da sich der Speicherbedarf $r^{\mathrm{RF}}(h^2, \widehat{\mathbf{d}}^{P_2})$ auf alle Prozessorelemente des Rechenfeldes bezieht, beträgt der Registerbedarf je Prozessorelement

$$r^{\mathrm{PE}}(h^2, \widehat{\mathbf{d}}^{P_2}) = \left\lceil \frac{r^{\mathrm{RF}}(h^2, \widehat{\mathbf{d}}^{P_2})}{\det(\boldsymbol{\Theta}^{P_2})} \right\rceil. \tag{7.53}$$

Beispiel 7.18 (Datenwiederverwendung aufgrund LS-2-Partitionen)
Wir haben einen Algorithmus mit einem dreidimensionalen Iterationsraum $\mathcal{I}$ und dem Iterationsvektor $\mathbf{i} = \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix}$. Dieser wird mit $\boldsymbol{\Theta}^{P_2} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ auf ein 2x2 Rechenfeld abgebildet. Wir wählen die eindimensionale LS-2-Partition $\boldsymbol{\Theta}_1^{S_2} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ bzw. die zweidimensionale LS-2-Partition $\boldsymbol{\Theta}_2^{S_2} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. In den Abbildungen 7.11(a) und (b) sind die beiden Partitionierungen in der Ebene $i_3 = 0$ dargestellt. In den jeweils ersten Co-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2} = \mathbf{0}$ wurden die Abarbeitungszeiten für die Iterationen angegeben. Die Abarbeitungsreihenfolge der Co-Partitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ lautet $\widehat{\boldsymbol{\alpha}}^{P_2} = (3 \quad 1 \quad 2)$. Das heißt, die nächste abzuarbeitende Co-Partition ist $\widehat{\boldsymbol{\kappa}}^{P_2} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, welche nicht dargestellt ist.

Der Algorithmus habe drei Abhängigkeitsvektoren $\widehat{\mathbf{d}}_1^{P_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $\widehat{\mathbf{d}}_2^{P_1} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ und $\widehat{\mathbf{d}}_3^{P_1} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, die in jede Raumrichtung jeweils eine andere Variable $y_{s_1}[.]$, $y_{s_2}[.]$ oder $y_{s_3}[.]$ transferieren. Die Abhängigkeitsvektoren $\widehat{\mathbf{d}}_1^{P_1}$ und $\widehat{\mathbf{d}}_2^{P_1}$ sind ebenfalls in Abbildung 7.11 dargestellt.

Bei der Verwendung der LS-2-Partition $\boldsymbol{\Theta}_1^{S_2}$, siehe Abbildung 7.11(a), benötigen wir je ein Register für Datenwiederverwendung bei Variable $y_{s_1}[.]$ und $y_{s_2}[.]$, denn die in einer Iteration berechneten Instanzen der Variablen werden in der zeitlich darauf folgenden Iteration entweder im gleichen oder einem benachbarten Prozessorelement wiederverwendet. Die drei Instanzen der Variable $y_{s_3}[.]$, welche mittels $\widehat{\mathbf{d}}_3^{P_1}$ transferiert werden, müssen lokal gespeichert werden, bevor sie in der Co-Partition $\widehat{\boldsymbol{\kappa}}^{P_2} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ verwendet

werden, was sich auch aus

$$r^{PE}_{1,s_3} = \left\lceil \frac{r^{PE}(1, \widehat{\mathbf{d}}^{P_2} = (0\,0\,1)^t, s_3)}{\det(\mathbf{\Theta}^{P_2})} \right\rceil = \left\lceil \frac{12}{4} \right\rceil = 3, \quad (h^2_{\max}(3) = 1) \tag{7.54}$$

ergibt. Somit ist der lokale Registerbedarf gleich fünf.

*Verwenden wir hingegen die LS-2-Partition $\mathbf{\Theta}^{S_2}_2$, siehe Abbildung 7.11(b), so werden für die lokale Datenwiederverwendung der Instanzen der Variable $y_{s_2}[.]$ **in** der LS-2-Partition nun drei Register benötigt. Ein Register benötigen wir weiterhin für die Wiederverwendung der Variable $y_{s_1}[.]$. Für die Wiederverwendung der berechneten $y_{s_3}[.]$ sind*

$$r^{PE}_{2,s_3} = \left\lceil \frac{r^{RF}(1, \widehat{\mathbf{d}}^{P_2} = (0\,0\,1)^t, s_3)}{\det(\mathbf{\Theta}^{P_2})} \right\rceil = \left\lceil \frac{24}{4} \right\rceil = 6 \tag{7.55}$$

Register notwendig. In der Summe werden damit zehn lokale Register benötigt.

Mit der Festlegung einer LS-2-Partition wird gleichzeitig der Abfolgevektor $\boldsymbol{\alpha}^{S_2}$ und die erste Abarbeitungsrichtung $\widehat{\alpha}^{P_2}_1$ festgelegt, denn diese Parameter haben entscheidenden Einfluss auf die Datenwiederverwendung.

Abarbeitungsreihenfolge der äußeren Copartitionen

Die Abarbeitungsreihenfolge für die äußeren Copartitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ soll so gewählt werden, dass die Datenwiederverwendung zwischen den Copartitionen möglichst hoch ist. Die Abarbeitungsreihenfolge der Copartitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ wird durch den Abfolgevektor $\widehat{\boldsymbol{\alpha}}^{P_2}$ beschrieben. Durch Anpassung des Algorithmus an den lokalen Speicher in den Prozessorelementen wurde das erste Element $\widehat{\alpha}^{P_2}_1$ des Abfolgevektors bereits festgelegt.

Für die Ermittlung der zweiten Raumrichtung partitionieren wir den Algorithmus in der dritten Stufe mit $\mathbf{\Theta}^{S_3} = \mathrm{diag}(\boldsymbol{\vartheta}^{S_3})$ und $\mathbf{\Theta}^{P_3} = \mathbf{E}$, wobei $\vartheta^{S_3}_{|\widehat{\alpha}^{P_2}_1|} = \widehat{\vartheta}^{P_2}_{|\widehat{\alpha}^{P_2}_1|}$ ist und alle anderen Elemente des Vektors $\boldsymbol{\vartheta}^{S_3}$ gleich eins sind. Das Element $\widehat{\vartheta}^{P_2}_j$ ergibt sich aus der Matrix $\widehat{\mathbf{\Theta}}^{P_2}$, welche entsprechend (5.28) auf Seite 74 bestimmt wird. Für den dreifach copartitionierten Algorithmus bestimmen wir den Speicherbedarf

$$r^{\mathrm{M}}(h^3, \widehat{\mathbf{d}}^{P_3}) = \sum_{s \in \mathcal{S}} \sum_{i=1}^{k} i \cdot \left| \mathcal{U}^{S_3}(s, i \cdot \widehat{\mathbf{d}}^{P_3}) \setminus \bigcup_{j=i+1}^{k} \mathcal{U}^{S_3}(s, j \cdot \widehat{\mathbf{d}}^{P_3}) \right| \tag{7.56}$$

für alle $\widehat{\mathbf{d}}^{P_3} = \pm\mathbf{e}_j$, $j \in \{1, \ldots, n\} \backslash \{|\widehat{\alpha}^{P_2}_1|\}$. Dann wählen wir den Vektor $\widehat{\mathbf{d}}^{P_3}$ aus, der zum höchsten Speicherbedarf führt. Mit Hilfe dieses Vektors ermitteln wir nun das zweite Element des Abfolgevektors $\widehat{\boldsymbol{\alpha}}^{P_2}$: $\widehat{\alpha}^{P_2}_2 = \widehat{d}^{P_3}_j \cdot j$, wobei $\widehat{d}^{P_3}_j = \pm 1$ das einzigste Nicht-Null-Element des Vektors $\widehat{\mathbf{d}}^{P_3}$ ist.

Entsprechend dem Vorgehen für $\widehat{\alpha}^{P_2}_2$ können nun die weiteren Vektorelemente $\widehat{\alpha}^{P_2}_k$, $3 \leq k \leq n$ bestimmt werden, wobei die Partitionsmatrix $\mathbf{\Theta}^{S_3}$ $k-1$ Elemente $\vartheta^{S_3}_{|\widehat{\alpha}^{P_2}_i|} = \widehat{\vartheta}^{P_2}_{|\widehat{\alpha}^{P_2}_i|}$

$1 \leq i \leq k-1$, größer Eins in der Hauptdiagonalen hat und der Transfervektor $\widehat{\mathbf{d}}^{P_3} = \pm\mathbf{e}_j$, $j \in \{1,\ldots,n\}\backslash\{|\widehat{\alpha}_1^{P_2}|,\ldots,|\widehat{\alpha}_{k-1}^{P_2}|\}$ sein kann. Ist der Speicherbedarf für alle Transfervektoren gleich, so wählen wir die Raumrichtung mit dem geringsten Betrag.

Dieses Vorgehen kann aber nur gewählt werden, wenn der Ausgangsalgorithmus ausgerichtete Abhängigkeitsvektoren, siehe Definition 7.1 auf Seite 141, besitzt. Bei unausgerichteten Abhängigkeitsvektoren ist die Abarbeitungsreihenfolge nicht frei wählbar.

Zusammenfassung

Die zweite Copartitionierung wird für die Anpassung an das Rechenfeld ($\boldsymbol{\Theta}^{P_2}$) und die Ausnutzung der lokalen Register in den Prozessorelementen ($\boldsymbol{\Theta}^{S_2}$) genutzt. Mit der Festlegung der Partitionierungsparameter muss ebenfalls der Abfolgevektor $\boldsymbol{\alpha}^{S_2}$ und die erste Abarbeitungsrichtung $\widehat{\alpha}_1^{P_2}$ festgelegt werden. Hierzu wurden Strategien vorgestellt, wie die Parameter der zweiten Copartitionierungsstufe ermittelt werden können. Ein Schwerpunkt bildete dabei die Datenwiederverwendung innerhalb und zwischen Partitionen.

Beispiel 7.19 (Abbildung des FIR-Filters auf ein lineares Rechenfeld)
In Beispiel 7.10 auf Seite 166 wurde die Teilwortparallelisierung des FIR-Filters vorgestellt. Der transformierte Algorithmus soll nun auf ein lineares Rechenfeld der Größe $\boldsymbol{\vartheta}^{\square} = \begin{pmatrix}4\end{pmatrix}$ *abgebildet werden.*

Für die Abbildung des Algorithmus auf das Rechenfeld sind zwei Partitionierungsmatrizen $\boldsymbol{\Theta}_1^{P_2} = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ *oder* $\boldsymbol{\Theta}_2^{P_2} = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ *möglich. Für den Abfolgevektor erhalten wir entsprechend Regel 7.2* $\widehat{\boldsymbol{\alpha}}_1^{P_2} = \begin{pmatrix}2 & 1\end{pmatrix}$ *bzw.* $\widehat{\boldsymbol{\alpha}}_2^{P_2} = \begin{pmatrix}1 & 2\end{pmatrix}$.

Die Transferrate in der ersten Abarbeitungsrichtung beträgt bezüglich der DvBs

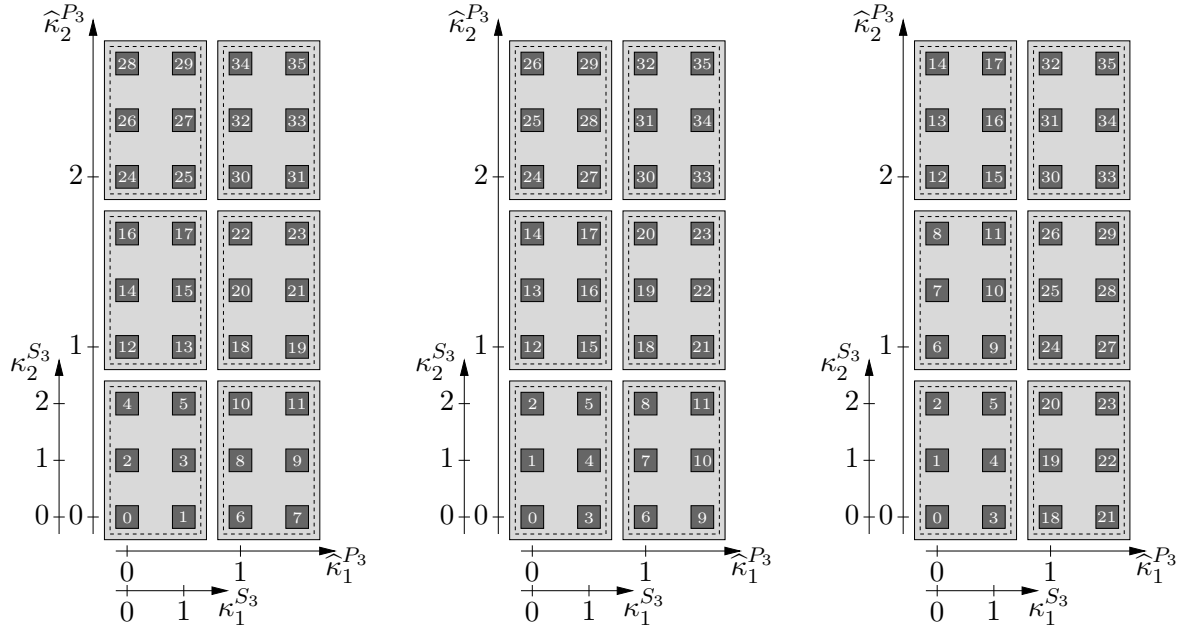
$$r_1^{RF}\left(1, \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) = 8 \quad \textit{für} \quad \boldsymbol{\Theta}_1^{P_2} \qquad \textit{und} \qquad r_2^{RF}\left(1, \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\right) = 4 \quad \textit{für} \quad \boldsymbol{\Theta}_2^{P_2},$$

wobei $\boldsymbol{\Theta}^{S_2} = \mathbf{E}$ *und* $h_{\max}^2(1) = h_{\max}^2(2) = 1$ *ist. Somit wählen wir* $\boldsymbol{\Theta}_1^{P_2}$ *für die Abbildung auf das Rechenfeld aus.*

In jedem Prozessorelement stehen uns vier lokale Register für lokale Datenwiederverwendung zur Verfügung, welche mittels geeigneter Wahl der Partitionsmatrix $\boldsymbol{\Theta}^{S_2}$ *ausgenutzt werden sollen. Entsprechend Regel 7.4 hat diese Partition nur eine Ausdehnung in Raumrichtung* i_1. *Damit wählen wir* $\boldsymbol{\Theta}^{S_2} = \left(\begin{smallmatrix}2 & 0\\ 0 & 1\end{smallmatrix}\right)$, *denn der Registerbedarf für die Datenwiederverwendung in der ersten Abarbeitungsrichtung beträgt nun* $r^{PE}(1, \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)) = \left\lceil \frac{r^{RF}\left(1,(0\ 1)^t\right)}{\det(\boldsymbol{\Theta}_1^{P_2})} \right\rceil = \lceil \frac{16}{4} \rceil = 4$. *Der Abfolgevektor für die LS-2-Partition lautet:* $\boldsymbol{\alpha}^{S_2} = \begin{pmatrix}1 & 2\end{pmatrix}$.

7.2.3 Stufe 3...p: Anpassung an Speicherarchitektur

Mit der dritten und den nachfolgenden Copartitionierungsstufen soll die Abbildung des Algorithmus an die Speicherarchitektur angepasst werden. Ziel dieser Anpassung ist es die Zwischenspeicher L_i der Speicherarchitektur effektiv auszunutzen, um Speicherzugriffe auf die nachfolgenden Zwischenspeicher L_{i+1} bis L_l und den Hauptspeicher zu



(a) $\boldsymbol{\alpha}^{S_3} = (1\ 2)$, $\widehat{\boldsymbol{\alpha}}^{P_3} = (1\ 2)$ (b) $\boldsymbol{\alpha}^{S_3} = (2\ 1)$, $\widehat{\boldsymbol{\alpha}}^{P_3} = (1\ 2)$ (c) $\boldsymbol{\alpha}^{S_3} = (2\ 1)$, $\widehat{\boldsymbol{\alpha}}^{P_3} = (2\ 1)$

Abbildung 7.12: Sequenzielle Abarbeitung eines copartitionierten Iterationsraums

minimieren. Durch die Verringerung der Speicherzugriffe auf die Zwischenspeicher L_{i+1} bis L_l und den Hauptspeicher kann die Verlustleistung der Speicherarchitektur bei Abarbeitung des Algorithmus merklich reduziert werden, wie unter anderem in [CWD+98] gezeigt wurde.

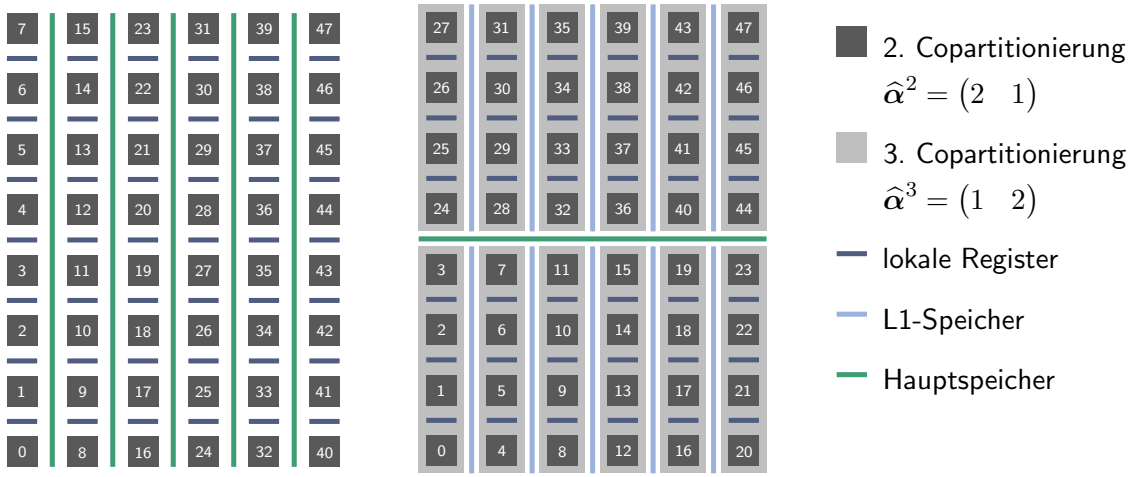
Jeweils eine Copartitionierungsstufe k wollen wir für die Anpassung an einen Zwischenspeicher L_{k-2} nutzen. Da bei diesen Copartitionierungsstufen keine Parallelverarbeitung möglich ist[5], muss für die Größe der LP-k-Partitionen $\boldsymbol{\Theta}^{P_k} = \mathbf{E}$ gelten. Durch $\boldsymbol{\Theta}^{S_k} \neq \mathbf{E}$ kann mittels der Abfolgevektoren $\boldsymbol{\alpha}^{S_k}$ und $\widehat{\boldsymbol{\alpha}}^{P_k}$ die Abarbeitungsreihenfolge der Iterationen $\widehat{\boldsymbol{\kappa}}^{P_{k-1}}$ beeinflusst werden, wie nachfolgendes Beispiel zeigt.

Beispiel 7.20 (Veränderung der Abarbeitungsreihenfolge)
Betrachten wir einen Algorithmus, der in der dritten Copartitionierungsstufe mit $\boldsymbol{\Theta}^{S_3} = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 3 \end{smallmatrix}\right)$ *und* $\boldsymbol{\Theta}^{P_3} = \mathbf{E}$ *partitioniert wird. Die Größe des zweifach copartitionierten Iterationsraums beträgt* $\widehat{\mathcal{K}}^{P_2} = \{\widehat{\boldsymbol{\kappa}}^{P_2} = \left(\widehat{\kappa}_1^{P_2} \quad \widehat{\kappa}_2^{P_2}\right)^t \mid 0 \leq \widehat{\kappa}_1^{P_2} \leq 3 \wedge 0 \leq \widehat{\kappa}_2^{P_2} \leq 8\}$. *Abbildung 7.12 zeigt die Partitionierung dieses Iterationsraums in drei Versionen. Diese Versionen unterscheiden sich nur in der Wahl der Abfolgevektoren* $\boldsymbol{\alpha}^{S_3}$ *und* $\widehat{\boldsymbol{\alpha}}^{P_3}$*, wie die Abarbeitungszeitpunkte in den Iterationen* $\widehat{\boldsymbol{\kappa}}^{P_2} = \boldsymbol{\kappa}^{S_3} + \boldsymbol{\Theta}^{S_3}\widehat{\boldsymbol{\kappa}}^{P_3}$ *zeigen.*

Die Abarbeitungsdauer wird durch den Abfolgevektor nicht beeinflusst.

Die Abarbeitungsreihenfolge der Iterationen $\widehat{\boldsymbol{\kappa}}^{P_{k-1}} = \widehat{\boldsymbol{\kappa}}^{S_k} + \boldsymbol{\Theta}^{S_k}\widehat{\boldsymbol{\kappa}}^{P_k}$ wird durch die Abfolgevektoren $\boldsymbol{\alpha}^{S_k}$ und $\widehat{\boldsymbol{\alpha}}^{P_k}$ festgelegt. Mit dem Vektor $\boldsymbol{\alpha}^{S_k}$ wird die Abarbeitung der

[5]Für eine Parallelverarbeitung in der dritten Copartitionierungsstufe müssten mehrere gleichartige Rechenfelder existieren, die gleichartig miteinander verbunden sind.



(a) Anpassung an lokale Register $\widehat{\boldsymbol{\alpha}}^{P_2} = (2\ 1)$

(b) Anpassung an Speicher L_1 $\boldsymbol{\alpha}^{S_3} = (2\ 1)$, $\widehat{\boldsymbol{\alpha}}^{P_3} = (1\ 2)$

Abbildung 7.13: Speicheranpassung durch mehrstufige Copartitionierung

Iterationen in einer Copartition $\widehat{\boldsymbol{\kappa}}^{P_k}$ festgelegt und der Vektor $\widehat{\boldsymbol{\alpha}}^{P_k}$ bestimmt die Reihenfolge der Abarbeitung der Copartition $\widehat{\boldsymbol{\kappa}}^{P_k}$.

Durch geeignete Wahl der Partitionsmatrix $\boldsymbol{\Theta}^{S_k}$ soll nun eine Anpassung an den Zwischenspeicher L_{k-2} mit der Größe $M_{L_{k-2}}$ erfolgen. Hierfür soll eine Gleichung, ähnlich (7.51) verwendet werden:

$$r^{L_{k-2}}(h^k = h^k_{\max}, \widehat{\mathbf{d}}^{P_k}) = \sum_{s\in\mathcal{S}} \sum_{i=1}^{h} i \cdot \left| \mathcal{U}^{S_2}\left(s, i \cdot \widehat{\mathbf{d}}^{P_k}\right) \setminus \bigcup_{j=i+1}^{h} \mathcal{U}^{S_2}\left(s, j \cdot \widehat{\mathbf{d}}^{P_k}\right) \right|,$$

$$\widehat{\mathbf{d}}^{P_k} = \operatorname{sign}(\widehat{\alpha}_1^{P_k}) \cdot \mathbf{e}_{|\widehat{\alpha}_1^{P_k}|}. \quad (7.57)$$

Diese Gleichung ermittelt den Speicherbedarf für die Speicherung der Daten, die in der ersten Abarbeitungsrichtung $\widehat{\alpha}_1^{P_k}$ der Partitionen $\widehat{\boldsymbol{\kappa}}^{P_k}$ der k-ten Copartitionierungsstufe wiederverwendet werden. Mit $h^k = h^k_{\max}, \widehat{\mathbf{d}}^{P_k}$ sollen alle in einer Abarbeitungsrichtung wiederverwendbaren Daten im jeweiligen Zwischenspeicher L_{k-2} gespeichert werden. Meist gilt für $h^k_{\max} = 1$. In diesem Fall ist $r^{L_{k-2}}(1, \widehat{\mathbf{d}}^{P_k}) = U^{S_2}\left(\widehat{\mathbf{d}}^{P_k}\right)$. Die Anzahl an Daten, die in der ersten Abarbeitungsrichtung wiederverwendet werden, kann durch die Partitionsmatrix $\boldsymbol{\Theta}^{S_k}$ beeinflusst werden.

Damit der in der vorherigen Copartitionierungsstufe $k-1$ ermittelte Speicherbedarf $r^{RF}(h^2, \widehat{\mathbf{d}}^{P_2})$ bzw. $r^{L_{k-3}}(h^{k-1}_{\max}, \widehat{\mathbf{d}}^{P_{k-1}})$ erhalten bleibt und der in der aktuellen Copartitionierungsstufe ermittelte Speicherbedarf $r^{L_{k-2}}(h^k_{\max}, \widehat{\mathbf{d}}^{P_k})$ effektiv ausgenutzt wird, muss für die Abfolgevektoren der Copartitionierungsstufe k, mit $k \geq 2$, gelten:

$$\boldsymbol{\alpha}^{S_k} = \widehat{\boldsymbol{\alpha}}^{P_{k-1}} \qquad \text{und} \qquad \widehat{\alpha}_1^{P_k} \neq \alpha_1^{S_k}. \quad (7.58)$$

In Abbildung 7.13 ist die Strategie der Speicheranpassung illustriert. Abbildung 7.13(a) zeigt die sequenzielle Abarbeitung der Copartitionen $\widehat{\boldsymbol{\kappa}}^{P_2}$ entsprechend Abfolgevektor

$\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix} 2 & 1 \end{pmatrix}$. Die Daten einer Copartition, die in den zeitlich nachfolgenden Copartitionen der ersten Abarbeitungsrichtung wiederverwendet werden, sind in den lokalen Registern der Prozessorelemente gespeichert. Diese Datenmenge ist mittels der kurzen, dunkelblauen Linien zwischen den Copartitionen dargestellt. Die anderen Daten, illustriert durch die lange grüne Linie, müssen im Hauptspeicher gespeichert werden, bevor sie acht Copartitionen später wiederverwendet werden.

Durch die dritte Copartitionierungsstufe mit $\boldsymbol{\Theta}^{S_3} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ und $\boldsymbol{\Theta}^{P_3} = \mathbf{E}$, sowie den Abfolgevektoren $\boldsymbol{\alpha}^{S_3} = \widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix} 2 & 1 \end{pmatrix}$ und $\widehat{\boldsymbol{\alpha}}^{P_3} = \begin{pmatrix} 1 & 2 \end{pmatrix}$ erhalten wir die in Abbildung 7.13(b) dargestellte Abarbeitungsreihenfolge. Damit wurde die in der zweiten Abarbeitungsrichtung der zweiten Copartitionierungsstufe transferierte Datenmenge soweit reduziert, dass sie im Zwischenspeicher L_1 abgelegt werden kann. Die in dem Zwischenspeicher L_1 gespeicherte Datenmenge ist in der Abbildung 7.13(b) durch eine hellblaue Linie dargestellt. Somit muss nun nur noch die zwischen den Copartitionen $\widehat{\boldsymbol{\kappa}}^{P_3}$ transferierte Datenmenge, welche durch eine grüne Linie in Abb. 7.13(b) illustriert ist, im Hauptspeicher gespeichert werden. Dieser deutlich verringerte Speicherzugriff auf den Hauptspeicher durch Ausnutzung des Zwischenspeichers L_1 führt im Allgemeinen zu einer geringeren Verlustleistung.

Wie wird nun die Größe der LS-k-Partition ermittelt?

- Ausgangspunkt ist eine LS-k-Partition der Größe $\boldsymbol{\Theta}^{S_k} = \mathbf{E}$. Diese Partition dehnen wir in die Raumrichtung $j = |\widehat{\alpha}_1^{P_{k-1}}|$ aus, in welcher die Copartitionen $\widehat{\boldsymbol{\kappa}}^{P_{k-1}}$ zuerst sequenziell abgearbeitet werden.

- Anschließend partitionieren wir alle Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ des Algorithmus k-fach unter Verwendung der „eindimensionalen“ LS-k-Partition.

- Nun ermitteln wir die Abarbeitungsrichtung $\widehat{\alpha}^{P_k}$ mit der höchsten Datenwiederverwendung in der ersten Abarbeitungsrichtung. Hierfür geben wir für $\vartheta_j^{S_k}$ einen konstanten Wert vor, z. B. $\vartheta_j^{S_k} = 4$ und bestimmen den Speicherbedarf $r^{L_k}(h_{\max}^k, \widehat{\mathbf{d}}^{P_k})$ für $\widehat{\mathbf{d}}^{P_k} = \{\pm\mathbf{e}_i \,|\, (1 \leq i \leq n) \backslash |\widehat{\alpha}_1^{P_{k_1}}|\}$, wobei $\mathbf{e}_i$ der i-te Spaltenvektor der Einheitsmatrix $\mathbf{E}$ ist. Aus den Vektoren $\widehat{\mathbf{d}}^{P_k}$ wählen wir denjenigen aus, der zum größten Speicherbedarf führt. Mittels des ausgewählten Vektor $\widehat{\mathbf{d}}^{P_k} = \pm\mathbf{e}_i$ kann weiterhin das erster Element $\widehat{\alpha}_1^{P_k}$ des Abfolgevektors $\widehat{\boldsymbol{\alpha}}^{P_k}$ bestimmt werden.

- Nachfolgend ermitteln wir den Maximalwert $\vartheta_{\max,j}^{S_k}$ für den der Speicherbedarf $r^{L_k}(h_{\max}^k, \widehat{\mathbf{d}}^{P_k})$ noch kleiner als die Größe $M_{L_{k-2}}$ des Zwischenspeichers L_{k-2} ist:

$$r^{L_k}(h_{\max}^k, \widehat{\mathbf{d}}^{P_k}) < M_{L_{k-2}}. \tag{7.59}$$

  Bei der Bestimmung von $\vartheta_{\max,j}^{S_3}$ der dritten Copartitionierungsstufe muss beachtet werden, dass nicht alle in der ersten Abarbeitungsrichtung $\widehat{\alpha}_1^{P_2} = \alpha^{S_3}$ der zweiten Copartitionierungsstufe wiederverwendbaren Daten $r^{\mathrm{RF}}(h, \widehat{\mathbf{d}}^{P_2})$ in den Registern des Rechenfeldes gespeichert worden sein können ($h^3 < h_{\max}^3$). Dementsprechend muss für die dritte Copartitionierungsstufe gelten:

$$r^{L_3}(h_{\max}^3, \widehat{\mathbf{d}}^{P_3}) + r^{\mathrm{RF}}(h^2 + 1, h_{\max}^2, \widehat{\mathbf{d}}^{P_2}) < M_{L_1}. \tag{7.60}$$

- Ist die Ausdehnung $\vartheta^{S_k}_{\max,j}$ der LS-k-Partition größer oder gleich der Anzahl $\widehat{\vartheta}^{P_2}_j$ der Partitionen in dieser Raumrichtung, so setzen wir $\vartheta^{S_k}_{\max,j} = \widehat{\vartheta}^{P_2}_j$. Die LS-$k$-Partition kann nun in die nächste Raumrichtung erweitert werden. Als Raumrichtung wählen wir die zweite Abarbeitungsrichtung $\widehat{\alpha}_2^{P_{k-1}}$ für die Copartitionen $\widehat{\boldsymbol{\kappa}}^{P_{k-1}}$ der vorherigen Copartitionierungsstufe. Die Bestimmung der Ausdehnung in dieser Raumrichtung ist ähnlich der Bestimmung in der ersten Raumrichtung. Es ist aber zu beachten, dass der Speicherbedarf für die Abarbeitung in die erste Abarbeitungsrichtung im Zwischenspeicher reserviert bleiben muss.

Für die weitere Arbeit gehen wir davon aus, dass die LS-k-Partitionen nur in einer Raumrichtung eine Ausdehnung größer Eins haben.

Mit der Bestimmung von $\vartheta^{S_k}_{\max,j}$ haben wir die bezüglich der Speicherausnutzung günstigste Partitionsgröße ermittelt. Bei einer Partitionierung mit diesem Wert können Partitionen entstehen, bei dem ein Großteil der Elemente dieser Partition nicht Elemente des Iterationsraums $\mathcal{I}^{\kappa}$ sind. Haben wir beispielsweise einen eindimensionalen Iterationsraum $\mathcal{I} = \{\mathbf{i} = (i) \,|\, 0 \leq i \leq 7\}$ und partitionieren diesen in Partitionen der Größe $\boldsymbol{\Theta} = (6)$, so ist die erste Partition $\widehat{\boldsymbol{\kappa}} = (0)$ vollständig, doch die zweite Partition $\widehat{\boldsymbol{\kappa}} = (1)$ enthält nur zwei gültige Elemente $\boldsymbol{\kappa} \in \{(0), (1)\}$.

Diese unvollständigen Partitionen können die Abarbeitungsdauer des Algorithmus unnötig verlängern, da der Ablaufvektor $\widehat{\boldsymbol{\tau}}^{P_k}$ mit Hilfe der Matrix $\widehat{\boldsymbol{\Theta}}^{P_k}$ bestimmt wird. Deshalb sollte die nachfolgende Regel beachtet werden.

Regel 7.5 (Vollständige Copartitionen)
Die Größe der LS-k-Partition sollte so gewählt werden, dass alle Copartitionen $\widehat{\boldsymbol{\kappa}}^{P_k}$ vollständig sind. Unvollständige Copartitionen führen aufgund des linearen Ablaufplans zu Leertakten bei der Abarbeitung des Algorithmus. Diese Leertakte verlängern die Abarbeitungszeit des Algorithmus unnötig.

Aus der maximalen Partitionsgröße $\vartheta^{S_k}_{\max,j}$ kann die minimale Anzahl an Partitionen in der Raumrichtung j mit $z_{\min,j} = \lceil \widehat{\vartheta}^{P_{k-1}} / \vartheta^{S_k}_{\max,j} \rceil$ ermittelt werden. Aus $z_{\min,j}$ kann die effektive Partitionsgröße $\vartheta^{S_k}_{\text{eff},j} = \lceil \widehat{\vartheta}^{P_{k-1}} / z_{\min,j} \rceil$ bei minimaler Partitionenzahl ermittelt werden.

Die Partitionsgröße $\vartheta^{S_k}_{\text{eff},j}$ und Partitionsanzahl $z_{\min,j}$ sind optimal, wenn gilt $\widehat{\vartheta}^{P_{k-1}} = \vartheta^{S_k}_{\text{eff},j} \cdot z_{\min,j}$. Im Allgemeinen ist das für diese beiden Werte nicht der Fall. Durch Vergrößern der Anzahl an Partition und Verkleinern der Partitionsgröße ermitteln wir die Parameter, für die gilt: $\widehat{\vartheta}^{P_{k-1}} \leq \vartheta^{S_k}_{\text{opt},j} \cdot z_{\text{opt},j} \leq \epsilon$, wobei ϵ die Anzahl an ungültigen Iterationen $\boldsymbol{\kappa}^{S_k}$ in der letzten Copartition der Raumrichtung j ist.

Die optimale Partitionsgröße der aktuellen Copartitionierungsstufe k hängt auch von den gewählten Partitionsgrößen der vorherigen Copartitionierungsstufen ab, denn die Matrix $\widehat{\boldsymbol{\Theta}}^{P_{k-1}}$ ist abhängig von diesen Partitionsgrößen. Die Ermittlung der optimalen Partitionsgrößen über alle Copartitionierungsstufen ist nicht Gegenstand dieser Arbeit, sondern muss in zukünftigen Arbeiten genauer untersucht werden.

Beispiel 7.21 (Ausnutzung des Speichers L_1 beim FIR-Filter)
Im Beispiel 7.19 wurde die Abbildung des teilwortparallelen FIR-Filters auf ein lineares Rechenfeld vorgestellt. Hierbei erfolgte auch eine Anpassung an die lokalen Register in den Prozessorelementen. Durch die dritte Copartitionierung soll nun der Zwischenspeicher L_1 ausgenutzt werden. Der Speicher habe eine Größe von $M_{L_1} = 32$.

Die Abarbeitungsreihenfolge der Copartitionen der zweiten Stufe wurde mit $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix} 2 & 1 \end{pmatrix}$ festgelegt. Damit erweitern wir die LS-3-Partition der dritten Copartitionierungsstufe in Raumrichtung $j = 2$. Als Abfolgevektoren für die dritte Copartitionierungsstufe erhalten wir damit $\boldsymbol{\alpha}^{S_3} = \widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix} 2 & 1 \end{pmatrix}$ und $\widehat{\boldsymbol{\alpha}}^{P_3} = \begin{pmatrix} 1 & 2 \end{pmatrix}$.

Bei einer Ausdehnung der LS-3-Partition mit $\vartheta^{S_3}_{\max,2} = 16$ erhalten wir einen Speicherbedarf von $r^{L_1}\left(1, \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)\right) = 31 < M_{L_1}$, wobei $h^3_{\max} = 1$ ist und alle Daten in der Abarbeitungsrichtung $\alpha^{S_3}_1$ im Rechenfeld gespeichert werden.

Diese Partitionsgröße würde $z_{\min} \cdot \vartheta^{S_3}_{\max,2} - \widehat{\vartheta}^{P_2}_2 = 12$ undefinierte Iterationen $\boldsymbol{\kappa}^{S_3}$ in der zweiten Copartition $\widehat{\boldsymbol{\kappa}}^{P_3}$ der Raumrichtung $j = 2$ verursachen, wobei $z_{\min} = 2$ und $\widehat{\vartheta}^{P_2}_2 = 20$ ist. Mit $\vartheta^{S_3}_{\text{eff},2} = \vartheta^{S_3}_{\text{opt},2} = \lceil \widehat{\vartheta}^{P_2}_2 / z_{\min} \rceil = 10$ erhalten wir vollständige Copartitionen, denn $z_{\min} \cdot \vartheta^{S_3}_{\text{eff},2} - \widehat{\boldsymbol{\vartheta}}^{P_2} = 0$. Somit wählen wir für die dritte Copartitionierungsstufe die Partitionierungsmatrizen $\boldsymbol{\Theta}^{S_3} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 10 \end{smallmatrix}\right)$ und $\boldsymbol{\Theta}^{P_3} = \mathbf{E}$.

Abschließend wollen wir nun die Reduzierung der Verlustleistung abschätzen. Hierfür bestimmen wir die Datenzugriffe auf den Zwischenspeicher L_1 und den Hauptspeicher, welche für die Zwischenspeicherung der Daten notwendig sind.

Ohne Anpassung an den Zwischenspeicher L_1 erhalten wir die Anzahl Z^o_M von Datenzugriffen auf den Hauptspeicher unter Zuhilfenahme einer dritten Copartitionierungsstufe mit $\boldsymbol{\Theta}^{S_3} = \begin{pmatrix} 1 & 0 \\ 0 & \widehat{\vartheta}^{P_2}_2 \end{pmatrix}$ und $\boldsymbol{\Theta}^{P_3} = \mathbf{E}$. Nach der Partitionierung bestimmen wir den Speicherbedarf $r^{L_1}\left(1, \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)\right) = 39$. Dieser Wert wird mit der Anzahl an Partitionsübergängen in Raumrichtung $j = 1$ multipliziert. Diese Anzahl erhalten wir durch $\widehat{\vartheta}^{P_3}_1 - 1 = 31$. Somit beträgt die Datenmenge, die im Hauptspeicher zwischengespeichert wird $Z^o_M = 39 \cdot 31 = 1209$.

Bei der Anpassung an den Zwischenspeicher L_1 erhalten wir die Anzahl Z^m_M von Datenzugriffen auf den Hauptspeicher durch eine vierte Copartitionierungsstufe mit $\boldsymbol{\Theta}^{S_4} = \begin{pmatrix} \widehat{\vartheta}^{P_3}_1 & 0 \\ 0 & 1 \end{pmatrix}$ und $\boldsymbol{\Theta}^{P_4} = \mathbf{E}$. Die Anzahl der Datenzugriffe ergibt sich nun aus dem Speicherbedarf $r^{L_2}(1, \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)) = 512$ und der Anzahl an Partitionsübergängen $\widehat{\vartheta}^{P_4}_2 - 1 = 1$ in Raumrichtung $j = 2$. Wir erhalten damit $Z^m_M = 512 \cdot 1$. Die Anzahl $Z^m_{L_1}$ an Datenzugriffen auf den Zwischenspeicher L_1 beträgt damit $Z^m_{L_1} = r^{L_1}\left(1, \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)\right) \cdot (\widehat{\vartheta}^{P_3}_1 - 1) \cdot \widehat{\vartheta}^{P_3}_2 = 19 \cdot 31 \cdot 2 = 1178$.

Gehen wir nun davon aus, dass die Verlustleistung beim Speicherzugriff auf den Hauptspeicher fünfmal größer als beim Zugriff auf den Zwischenspeicher L_1 ist, so erhalten wir eine Reduktion der Verlustleistung auf

$$\frac{5 \cdot Z^m_M + Z^m_{L_1}}{5 \cdot Z^o_M} \cdot 100\% = \frac{5 \cdot 512 + 1178}{5 \cdot 1209} \cdot 100\% = 62\% \tag{7.61}$$

für die Version mit Anpassung an den Zwischenspeicher L_1 gegenüber der Version ohne Anpassung.

Die Speicheranpassung mittels Partitionierung ist eng mit der Optimierung der Zwischenspeicher[6] bei der Datentransfer- und Speicheruntersuchung (engl.: Data Transfer and Storage Exploration, kurz: DTSE) [CWD+98] verbunden. Ziel der DTSE ist die Minimierung der Verlustleistung aufgrund von Speicherzugriffen durch eine Anpassung des Algorithmus an die Speicherarchitektur oder der Entwurf einer dementsprechenden Speicherarchitektur. Hierbei wird nicht nur die Speichergröße in die Untersuchung mit einbezogen, sondern es werden auch andere Eigenschaften des Speichers, wie z. B. Speicherorganisation und -adressierung, mit betrachtet. Für diese Anpassung an die Speicherarchitektur werden einzelne Schleifen des Schleifenprogramms zerteilt, was einer Teilung des Iterationsraum in diesen Raumrichtungen entspricht und auch als Partitionierung interpretiert werden kann. Die Partitionierung findet hierbei von außen nach innen statt. Dementsprechend erfolgt die Anpassung vom größten zum kleinsten Zwischenspeicher.

7.2.4 Ablaufplanung mit Anpassung an Funktionseinheiten

Das Ziel der Ablaufplanung ist die Bestimung des Ablaufvektors $\boldsymbol{\tau}^{\kappa}$ (globaler Ablaufplan) und der Startzeitpunkte t_{0,s_t} für alle Teilzuweisungen s_t des teilwortparallelen Algorithmus (lokaler Ablaufplan), siehe (5.33) auf Seite 76. Mit der Bestimmung dieser Parameter findet eine Bindung der Operationen F_{s_t} der Teilzuweisungen s_t an die Funktionseinheiten $m \in \mathcal{M}$ in den Prozessorelementen statt. Somit wird im Allgemeinen mit der Ablaufplanung eine Parallelverarbeitung in den Prozessorelementen erreicht. Der Ablaufplan muss die Kausalitätsbedingung, siehe Definition 5.4 auf Seite 77, erfüllen.

Meist wird zwischen dem globalen und dem lokalen Ablaufplan unterschieden. Der globale Ablaufplan ergibt sich im Wesentlichen aus den Abfolgevektoren $\boldsymbol{\alpha}^{S_1}, \boldsymbol{\alpha}^{S_2}, \ldots, \boldsymbol{\alpha}^{S_p}$ und $\widehat{\boldsymbol{\alpha}}^{P_p}$. Er definiert die Abarbeitungsreihenfolge entsprechend der Abfolgevektoren. Gegebenenfalls muss diese Abarbeitungsreihenfolge mittels der Leerlaufverzögerungen $\boldsymbol{\iota}^{S_1}$, $\boldsymbol{\iota}^{S_2}$, $\ldots$, $\boldsymbol{\iota}^{S_p}$ und $\widehat{\boldsymbol{\iota}}^{P_p}$ gestreckt oder mittels der Verzögerungsvektoren $\boldsymbol{\tau}^{\text{offs},1}, \boldsymbol{\tau}^{\text{offs},2}, \ldots,$ $\boldsymbol{\tau}^{\text{offs},p}$ verzögert werden, um die Kausalitätsbedingung zu erfüllen. Mit dem lokalen Ablaufplan legen wir die Länge des Iterationsintervalls λ und die Startzeitpunkte t_{0,s_t} für die Ausführung der Teilzuweisungen s_t fest. Bei der lokalen Ablaufplanung findet auch die Bindung der Teilzuweisungen an die Funktionseinheiten m statt.

Für die lokalen Ablaufplanung mit der Bindung an die Funktionseinheiten existieren bereits mehrere Lösungen, wie z. B. [Fim02], [MFM04] und [TTZ97][HT04]. Bei all diesen Arbeiten wird die ganzzahlige lineare Optimierung als Optimierungswerkzeug verwendet und ist Softwarepipelining erlaubt. Bei dem Ansatz in [MFM04] ist zusätzlich eine zyklische Bindung der Funktionseinheiten an die Teilzuweisungen möglich. Die Lösungsmethode [HT04] erlaubt auch die globale Ablaufplanung, wie in [SMHT08] gezeigt wurde.

[6] Die Zwischenspeicher werden in diesen Arbeiten als Hintergrundspeicher (engl.: background memory) bezeichnet.

Tabelle 7.7: Eigenschaften der Funktionseinheiten (Eingabeverzögerung $o_{m(s_t)} = l_{m(s_t)}$)

FE	Funktionen F_{s_t} mit Latenz $l_{m(s_t)}$ in den Klammern
m_1	verteileTW(1), F_p(1), ersetzeTW(1)
m_2	add(2), muladd(3)

Um die Ansätze zur lokalen Ablaufplanung für unsere Abbildung nutzen zu können, muss der Algorithmus zuerst teilwortparallelisiert werden, um die Operationen beim lokalen Ablaufplan und der Bindung berücksichtigen zu können. Anschließend können die oben genannten Methoden zur Bestimmung des lokalen Ablaufplans genutzt werden.

Am Beispiel des FIR-Filters soll abschließend die Ablaufplanung illustriert werden.

Beispiel 7.22 (Ablaufplanung für das FIR-Filter)
Für das mehrfach copartitionierten FIR-Filter mit Teilwortparallelität, siehe Beispiele 7.10, 7.19 und 7.21, soll der Ablaufplan ermittelt werden.

Das FIR-Filter wurde in Beispiel 7.10 mit $\mathbf{\Theta}^{S_1} = \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$, $\mathbf{\Theta}^{P_1} = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ *teilwortparallelisiert. Mit der zweiten Copartitionierungsstufe (Bsp. 7.19), erfolgte die Anpassung an das Rechenfeld mit* $\mathbf{\Theta}^{P_2} = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ *und die Ausnutzung der lokalen Speicher mit* $\mathbf{\Theta}^{P_2} = \left(\begin{smallmatrix}4 & 0\\ 0 & 1\end{smallmatrix}\right)$ *und* $\boldsymbol{\alpha}^{S_2} = \begin{pmatrix}1 & 2\end{pmatrix}$. *Eine dritte Copartitionierung mit* $\mathbf{\Theta}^{S_3} = \left(\begin{smallmatrix}1 & 0\\ 0 & 10\end{smallmatrix}\right)$, $\mathbf{\Theta}^{P_3} = \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$, $\boldsymbol{\alpha}^{S_3} = \begin{pmatrix}2 & 1\end{pmatrix}$ *und* $\widehat{\boldsymbol{\alpha}}^{P_3} = \begin{pmatrix}1 & 2\end{pmatrix}$ *passte den Algorithmus an den Zwischenspeicher* L_1 *an, wie in Beispiel 7.21 gezeigt wurde. Für die Hilfsmatrix* $\widehat{\mathbf{\Theta}}^{P_3}$ *erhalten wir damit* $\widehat{\mathbf{\Theta}}^{P_3} = \left(\begin{smallmatrix}32 & 0\\ 0 & 2\end{smallmatrix}\right)$.

Mit Hilfe dieser Partitionierungsmatrizen, der Abfolgevektoren und der Hilfsmatrix bestimmen wir die Ablaufvektoren $\boldsymbol{\tau}^{S_1} = \begin{pmatrix}1 & 0\end{pmatrix}$, $\boldsymbol{\tau}^{S_2} = \begin{pmatrix}1 & 0\end{pmatrix}$, $\boldsymbol{\tau}^{S_3} = \begin{pmatrix}0 & 1\end{pmatrix}$ *und* $\widehat{\boldsymbol{\tau}}^{P_3} = \begin{pmatrix}1 & 32\end{pmatrix}$, *wobei die Leerlaufverzögerungen* $\boldsymbol{\iota}^{S_1} = \boldsymbol{\iota}^{S_2} = \boldsymbol{\iota}^{S_3} = \widehat{\boldsymbol{\iota}}^{P_3} = \mathbf{0}$ *sind. Die Partitionsperioden betragen* $T^{S_1} = 1$, $T^{S_1} = 1$, $T^{S_2} = 2$, $T^{S_3} = 10$ *und* $\widehat{T}^{P_3} = 64$.

Bevor wir die Verzögerungsvektoren bestimmen, ermitteln wir den lokalen Ablaufplan mit der Bindung. Als Funktionseinheiten für die Abarbeitung der TWP-Operationen, siehe Algorithmus 7.3 auf Seite 167, stehen uns die beiden FUs m_1 *und* m_2 *jeweils einmal zur Verfügung. Tabelle 7.7 zeigt die Eigenschaften der Funktionseinheiten.*

Die anderen Operationen F_{s_t} *des teilwortparallelen FIR-Filters sind Zuweisungsoperationen und haben ein Offset von* $o_{m(s_t)} = 0$ *und eine Latenz von* $l_{m(s_t)} = 0$.

Abbildung 7.14 zeigt eine Lösung für den lokalen Ablaufplan. Das Iterationsintervall hat hierbei die Länge $\lambda = 3$.

Die teilzuweisungsabhängigen Verzögerungen betragen: $t_{0,1_1} = 0$, $t_{0,1_2} = 1$, $t_{0,x_{1,1}} = 0$, $t_{0,x_{2,1}} = 1$, $t_{0,p_1} = t_{0,p_2} = 1$, $t_{0,x_{3,1}} = 2$, $t_{0,2_1} = t_{0,2_2} = t_{0,2_3} = 3$, $t_{0,3_1} = t_{0,3_2} = 3$ *und* $t_{0,4_1} = 6$.

Man beachte, dass die beiden Teilzuweisungen der Zuweisungen s_p *und* s_3 *jeweils zur gleichen Zeit der gleichen Funktionseinheit zugeordnet wurden.*

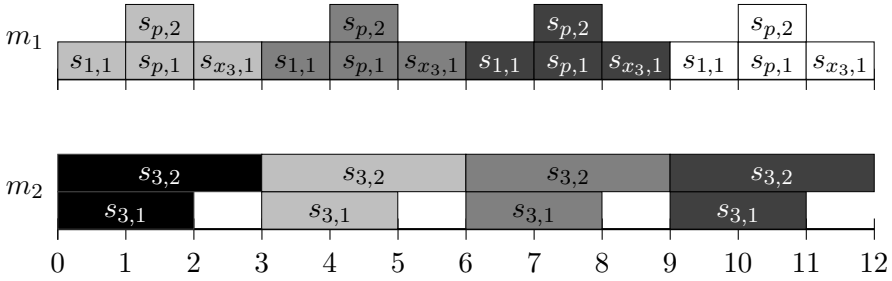


Abbildung 7.14: Gantt-Chart des teilwortparallelen FIR-Filters, welcher auf ein 4×1 Rechenfeld mit zwei Funktionseinheiten abgebildet wurde (Iterationsintervall $\lambda = 3$)

Abschließend überprüfen wir, ob die Kausalitätsbedingung immer eingehalten wurde und passen gegebenenfalls die Verzögerungsvektoren an. Der Verzögerungsvektor $\boldsymbol{\tau}^{\text{offs},1}$ *muss per Definition, siehe* (6.2) *auf Seite* 80, $\boldsymbol{\tau}^{\text{offs},1} = \mathbf{0}$ *sein.*

Der Abhängigkeitsvektor $\mathbf{d}^1_{s_1,s_{1_1}} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ *des Algorithmus* 7.3 *zerfällt durch die zweite und die dritte Copartitionierungsstufe in drei partitionierte Abhängigkeitsvektoren:*

$$\mathbf{d}^{\kappa}_{1,1,1,1} = \left(\left(\begin{smallmatrix}1&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right)\right)^t, \tag{7.62}$$

$$\mathbf{d}^{\kappa}_{2,1,1,1} = \left(\left(\begin{smallmatrix}-1&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}1&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right)\right)^t, \tag{7.63}$$

$$\mathbf{d}^{\kappa}_{2,2,1,1} = \left(\left(\begin{smallmatrix}-1&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}-3&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}1&0\end{smallmatrix}\right)\right)^t. \tag{7.64}$$

Für den partitionierten Abhängigkeitsvektor $\mathbf{d}^{\kappa}_{2,1,1,1}$ *erhalten wir mit*

$$\boldsymbol{\tau}^{\kappa} = \left(\lambda \cdot \boldsymbol{\kappa}^{S_2} \quad \boldsymbol{\tau}^{\text{offs},2} \quad \lambda \cdot T^{S_2} \cdot \boldsymbol{\kappa}^{S_3} \quad \boldsymbol{\tau}^{\text{offs},3} \quad \lambda \cdot T^{S_2} \cdot T^{S_3} \cdot \widehat{\boldsymbol{\kappa}}^{P_3}\right) \tag{7.65}$$

$t^{\triangle}(\mathbf{d}^{\kappa}_{2,1,1,1}) = \boldsymbol{\tau}^{\kappa} \cdot \mathbf{d}^{\kappa}_{2,1,1,1} + t_{0,1_2} - t_{v,1} = -3$ *einen negativen Wert, wobei* $\boldsymbol{\tau}^{\text{offs},2} = \boldsymbol{\tau}^{\text{offs},3} = \mathbf{0}$ *und* $t_{v,1} = 1$ *ist.*

Mittels des Verzögerungsvektors $\boldsymbol{\tau}^{\text{offs},2} = \begin{pmatrix}3 & 0\end{pmatrix}$ *erhalten wir nun* $t^{\triangle}(\mathbf{d}^{\kappa}_{2,1,1,1}) = 0$. *Die Kausalitätsbedingung wird mit diesem Verzögerungsvektor eingehalten.*

Die partitionierten Abhängigkeitsvektoren der anderen Abhängigkeitsvektoren erfüllen ebenfalls die Kausalitätsbedingung, somit beträgt der Ablaufplanvektor:

$$\boldsymbol{\tau}^{\kappa} = \left(\lambda \cdot \boldsymbol{\kappa}^{S_2} \quad \boldsymbol{\tau}^{\text{offs},2} \quad \lambda \cdot T^{S_2} \cdot \boldsymbol{\kappa}^{S_3} \quad \boldsymbol{\tau}^{\text{offs},3} \quad \lambda \cdot T^{S_2} \cdot T^{S_3} \cdot \widehat{\boldsymbol{\kappa}}^{P_3}\right) \tag{7.66}$$

$$= \left(3 \cdot \left(\begin{smallmatrix}1&0\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}3&0\end{smallmatrix}\right) \quad 3 \cdot 2 \cdot \left(\begin{smallmatrix}0&1\end{smallmatrix}\right) \quad \left(\begin{smallmatrix}0&0\end{smallmatrix}\right) \quad 3 \cdot 2 \cdot 10 \cdot \left(\begin{smallmatrix}1&32\end{smallmatrix}\right)\right). \tag{7.67}$$

Für die Abarbeitungsdauer des Algorithmus erhalten wir somit $t^d = 3852$. *Dieser Wert ist nur geringfügig höher (0,3 %) als der theoretisch mögliche Wert* $t^d_{opt} = \frac{255 \cdot 20 \cdot 3}{4} = 3840$.

7.2.5 Zusammenfassung

In diesem Abschnitt wurde gezeigt, wie die mehrstufige modifizierte Copartitionierung, vorgestellt in Kapitel 5 ab Seite 59, für die Abbildung eines Algorithmus auf ein Rechenfeld genutzt werden kann.

Mit der ersten Copartitionierungsstufe wird die Ausnutzung der Teilwortparallelität mittels der Partitionsmatrix $\mathbf{\Theta}^{P_1}$ erreicht. Die entscheidende Transformation des Algorithmus in einen teilwortparallelen Algorithmus wurde in Kapitel 6 ab Seite 79 vorgestellt.

Mittels der zweiten Copartitionierungsstufe wird der Algorithmus an das Rechenfeld angepasst (Partitionsmatrix $\mathbf{\Theta}^{P_2}$) und werden die lokalen Register in den PEs ausgenutzt (Partitionsmatrix $\mathbf{\Theta}^{S_2}$). Somit erreichen wir mit den ersten beiden Copartitionierungsstufen die optimale Anpassung an die Parallelität in den Funktionseinheiten, die Teilwortparallelität, und die Parallelität des Rechenfeldes. Für die Bestimmung der Partitionierungmatrix $\mathbf{\Theta}^{S_2}$ wurde die Transfermenge $\mathcal{U}^b(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k})$ eingeführt. Diese Menge beinhaltet die Elemente $(\boldsymbol{\kappa}^b, \boldsymbol{\kappa}^{b+1}, \ldots, \boldsymbol{\kappa}^k)$, welche zwischen zwei Partitionen der k-ten Partitionierungsstufe, gegeben durch den Vektor $\widehat{\mathbf{d}}^{\mathbf{\Theta}^k}$, transferiert werden.

Mittels weiterer Copartitionierungsstufen kann die Datenwiederverwendung bei der Abarbeitung des Algorithmus durch Ausnutzung der Zwischenspeicher L_l verbessert werden. Die Partitionsmatrix $\mathbf{\Theta}^{P_{2+l}}$ dieser Copartitionierungsstufen beträgt immer $\mathbf{\Theta}^{P_{2+l}} = \mathbf{E}$, da nur die Abarbeitungsreihenfolge der Iterationen verändert werden soll. Für die Bestimmung der Partitionsmatrix $\mathbf{\Theta}^{S_{2+l}}$ werden wieder die Transfermengen $\mathcal{U}^b(s, \widehat{\mathbf{d}}^{\mathbf{\Theta}^k})$ verwendet. Mit der Ausnutzung der Zwischenspeicher kann der Energiebedarf der Architektur für die Abarbeitung des Algorithmus gesenkt werden, wie am Beispiel 7.21 gezeigt.

Abschließend wurde kurz die Bestimmung des Ablaufplans mit der Bindung der Funktionseinheiten an die Funktionen F_{s_t} der Teilzuweisungen s_t vorgestellt. Diese Ablaufplanung mit der Bindung erlaubt die Ausnutzung der Parallelität in den Prozessorelementen aufgrund mehrerer Funktionseinheiten. Hierbei ist auch Softwarepipelining möglich.

Für die einzelen Abbildungsschritte wurden Strategien zur Bestimmung der Parameter für die Abbildung vorgestellt. Eine ausführliche Entwurfsraumexploration über alle Abbildungsschritte hinweg fand dabei nicht statt. Das soll Aufgabe zukünftiger Untersuchungen sein.

Kapitel 8

Realisierungen

In diesem Abschnitt sollen drei parallele Realisierungen von Algorithmen und die Ergebnisse bezüglich Rechenbeschleunigung vorgestellt werden. Am Beispiel des FIR-Filters und des STAF-Algorithmus wird die Abbildung von Algorithmen auf ein Rechenfeld mit einem Prozessorelement ($\boldsymbol{\vartheta}^{\square} = \big(1\big)$) gezeigt und deren Geschwindigkeitsgewinn präsentiert, wobei das Prozessorelement Teilwortparallelität erlaubt. Anschließend wird anhand des Kantenerkennungsalgorithmus die Abbildung auf ein Rechenfeld mit mehreren Prozessorelementen, die teilwortparallele Befehle verarbeiten können, illustriert.

8.1 Rechenfeld mit einem Prozessorelement

Der Schwerpunkt der Abbildung von Algorithmen auf Rechenfelder mit einem Prozessorelement ($\boldsymbol{\vartheta}^{\square} = \big(1\big)$) lag bei der Ausnutzung der Teilwortparallelität. Als Zielarchitektur dienten die Mehrzweckprozessoren von Intel und AMD, wie sie in Abschnitt 2.5.1 ab Seite 18 vorgestellt wurden. Die Abbildungen der Algorithmen auf diese Architekturen erfolgten mit Hilfe von Inlineassemblerbefehlen, welche von Dietz und Fisher für SWARC [FD00] entwickelt wurden. Später wurden auch Intrinsic-Funktionen verwendet.

Im Allgemeinen konnten mit den Inlineassemblerbefehlen bessere Beschleunigungen erreicht werden als mit den Intrinsic-Funktionen. Doch die Arbeit mit Inlineassemblerbefehlen ist schwieriger, da beispielsweise die Registerausnutzung durch den Programmierer festgelegt werden muss.

8.1.1 FIR-Filter

Die Abbildung des FIR-Filters auf ein Rechenfeld mit einen Prozessorelement, welches Teilwortparallelität erlaubt, wurde bereits bei der Präsentation der Abbildungsstrategie in den vorherigen Kapiteln vorgestellt. In diesem Abschnitt sollen nun die Ergebnisse

der Abbildung des FIR-Filters auf einen Intel Core2 Duo mit 2,66 GHz präsentiert werden. Für die Teilwortparallelität wurden die Befehle des MMX-Befehlssatz genutzt. Der MMX-Befehlssatz arbeitet mit einer Datenwortbreite von 64 Bit für das DvB. Die Daten des FIR-Filters sind 16 Bit breit. Somit erreichen wir einen Grad der Teilwortparallelität von $B = 4$. Die Intel-Architektur erlaubt weiterhin das unausgerichtete Lesen und Schreiben der Daten als DvBs. Auch das Lesen und Schreiben von einzelnen Teilwörtern ist möglich.

Das Ergebnis der Teilwortparallelisierung wurde in Algorithmus 7.3 auf Seite 167 präsentiert. Die Partitionierungsmatrizen hierfür lauteten $\mathbf{\Theta}^{S_1} = \mathbf{E}$ und $\mathbf{\Theta}^{P_1} = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$. Eine Packoperation F_p entsteht aufgrund der Partitionierung des Abhängigkeitsvektors $\mathbf{d}^1_{s_2 s_{2_3}} = \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$.

Die Abbildung auf einen Kern des Intel-Prozessor erfolgte mit $\mathbf{\Theta}^{P_2} = \mathbf{E}$.[1]

Drei Versionen des teilwortparallelen FIR-Filters wurden implementiert.

- **pack:** Realisierung der Packoperationen F_p mittels Schiebe- und Logikoperationen, da kein Doppelwortschiebebefehl zur Verfügung stand. Eine lokale Datenwiederverwendung durch Ausnutzung der lokalen Register findet nicht statt ($\mathbf{\Theta}^{S_2} = \mathbf{E}$).
- **direkt:** Vermeidung der Packoperationen F_p durch mehrfaches Lesen der unausgerichteten Daten aus dem Speicher. Eine lokale Datenwiederverwendung durch Ausnutzung der lokalen Register findet nicht statt ($\mathbf{\Theta}^{S_2} = \mathbf{E}$).
- **mem:** Vermeidung der Packoperationen F_p durch mehrfaches Lesen der unausgerichteten Daten aus dem Speicher. Zusätzlich wird die Datenwiederverwendung durch Ausnutzung der sechs freien, lokalen MMX-Register mittels $\mathbf{\Theta}^{S_2} = \left(\begin{smallmatrix} 6 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ in der zweiten Copartitionierungsstufe verbessert.

Die Abfolgevektoren für die drei Versionen lauten $\boldsymbol{\alpha}^{S_2} = \begin{pmatrix} 1 & 2 \end{pmatrix}$ und $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix} 2 & 1 \end{pmatrix}$, wobei $\boldsymbol{\alpha}^{S_2}$ nur für Version „mem“ relevant ist. Die Versionen wurden jeweils mit Inlineassemblerbefehlen (ASM) und Intrinsic-Funktionen realisiert.

Die verschiedenen Realisierungen wurden mit dem GNU Compiler Version 4.3.1 mit der Optimierungsstufe 3 übersetzt. Tabelle 8.1 zeigt die ermittelten Berechnungszeiten für die verschiedenen Realisierungen bei unterschiedlichen Filterlängen J, wobei die Anzahl I der Ausgabedaten $I = 8192$ beträgt. Für die Ermittlung auswertbarer Zeiten wurden 5000 Iterationen des Algorithmus hintereinander ausgeführt. Zum Vergleich enthält die Tabelle 8.1 die Berechnungszeiten, welche mittels des Ausgangsalgorithmus (Orig) ermittelt wurden.

Die Berechnungszeiten zeigen, dass alle teilwortparallelen Realisierungen deutlich schneller sind, als die Realisierung des Ausgangsalgorithmus. Die minimale Beschleunigung bezüglich des Ausgangsalgorithmus beträgt 2,32 für die Intrinsic-Realisierung mit Packoperationen (pack) bei $J = 32$ und die maximale Beschleunigung beträgt 6,5 für die

[1] Eine Parallelverarbeitung der beiden Kerne des Intel-Prozessors kann mit thread-basierten Parallelisierungsmethoden erreicht werden, welche in dieser Arbeit aber nicht betrachten werden.

Tabelle 8.1: Berechnungszeiten des FIR-Filters abhängig von der Filterlänge J (in Sek.)

J	Orig	Inlineassemblerbefehle			Intrinsic-Funktionen		
		pack	direkt	mem	pack	direkt	mem
4	0,39	0,13	0,08	0,06	0,16	0,10	0,07
8	0,71	0,24	0,18	0,11	0,28	0,22	0,13
12	1,03	0,38	0,27	0,16	0,43	0,31	0,19
16	1,36	0,56	0,50	0,24	0,56	0,43	0,25
20	1,67	0,67	0,55	0,28	0,71	0,52	0,30
24	2,00	0,78	0,62	0,34	0,86	0,64	0,37
28	2,33	0,92	0,72	0,38	0,97	0,76	0,43
32	2,65	1,01	0,77	0,43	1,14	0,89	0,49

Inlineassemblerrealisierung mit Speicheroptimierung (mem) bei $J = 4$. Die Inlineassemblerrealisierung der Version „mem“ führt für alle Filterlängen zu der höchsten Beschleunigung (5,67 - 6,50). Die hohen Beschleunigungswerte von mehr als 4,0 bei einer Teilwortparallelität von $B = 4$ resultiert aus der lokalen Datenwiederverwendung.

Abbildung 8.1 zeigt die Beschleunigungen für die verschiedenen Realisierungen und Versionen im Vergleich zum Ausgangsalgorithmus. Der Vergleich der Inlineassemblerrealisierungen mit den Intrinsic-Realisierungen zeigt, dass mit den Inlineassemblerrealisierungen meist etwas schnellere Berechnungszeiten erreicht werden können als mit den Intrinsic-Realisierungen. Nur bei den Versionen „direkt“ mit den Filterlängen $J = 16$ und $J = 20$ führt die Intrinsic-Realisierung zu besseren Beschleunigungen als die Inlineassemblerrealisierung.

Vergleicht man die drei Versionen „direkt“, „pack“ und „mem“ miteinander, so wird deutlich, dass die Version „direkt“ schneller als die Version „pack“ ist. Der Grund hierfür liegt in der aufwändigen Realisierung der Packoperation F_p. Diese Operation muss mittels mehrerer Schiebe- und Logikoperationen realisiert werden und benötigt dadurch länger als das unausgerichtete Lesen der Daten aus dem Speicher. Diese ungenügende Realisierung der Packoperation führte zu dem in Abschnitt 6.6.1 vorgestellten Doppelwortschiebebefehl „schiebeTWinDvBs“. Die Version „mem“ mit der Speicheroptimierung ist, wie zu erwarten, besser als die Version „direkt“.

Die Veränderungen der Beschleunigung in Abhängigkeit von der Filterlänge J können nicht plausibel erklärt werden.

8.1.2 STAF-Algorithmus

Der kurzfristige Analysefilteralgorithmus (englisch: short term analysis filtering (STAF) algorithm) ist Teil des GSM Standards. Die Beschreibung des Algorithmus ist in Anhang A.2 zu finden.

Basis unserer Untersuchung ist das GSM 06.10 verlustbehaftete Sprachkompressionsprogramm *toast* von J. Degener und C. Bormann [DB94]. Die Ein- und Ausgabedaten der

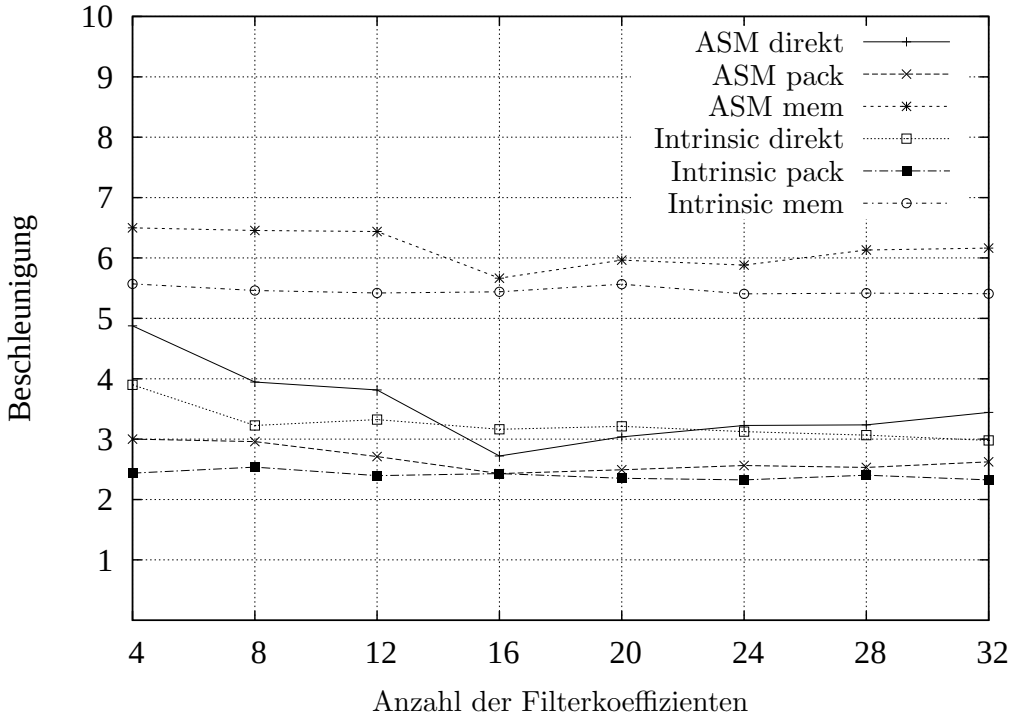


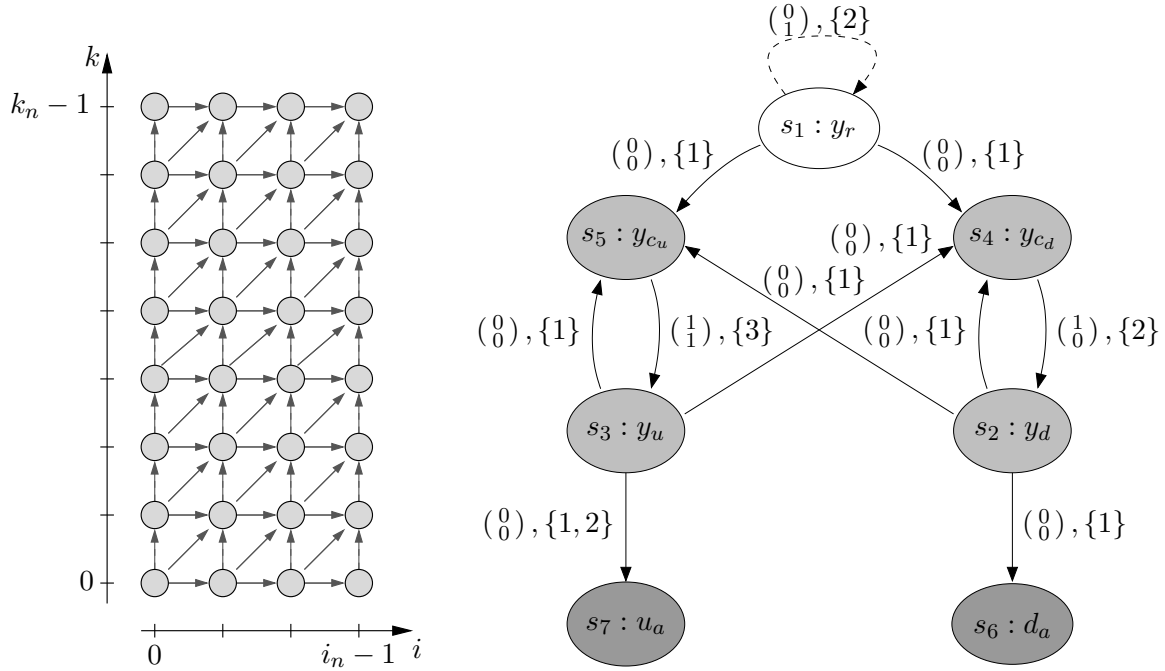
Abbildung 8.1: Beschleunigungen der FIR-Filterversionen gegenüber Originalversion

Realisierung sind Ganzzahlen mit einer Wortbreite von 16 Bit. Unter Verwendung des MMX-Befehlssatzes erhalten wir somit wieder eine Teilwortparallelität von $B = 4$.

Die Multiplikation- und Additionoperationen des STAF-Algorithmus wurden in dem toast-Progamm verändert, um die Ausgabedaten als 16-Bit-Werte speichern zu können. Das hat zur Folge, dass für die teilwortparallele Realisierung des Rechenkerns des STAF-Algorithmus, das sind zwei Additionen und zwei Multiplikationen, 30 MMX-Befehle und sechs Register benötigt werden.

Betrachten wir den reduzierten Abhängigkeitsgraphen des STAF-Algorithmus in Abbildung 8.2(b), so erkennen wir, dass dieser mehrere Kreise besitzt. Für den Kreis $\mathcal{W}_1^\circ$, zwischen den Knoten s_2 und s_4, erhalten wir als Summe der Abhängigkeitsvektoren $\mathbf{d}_1^\Sigma = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$. Damit ist entsprechend Satz 7.1 von Seite 146 keine teilwortparallele Abarbeitung in Raumrichtung i_1 möglich. Folglich wählen wir als Partitionsmatrix $\mathbf{\Theta}^{P_1} = \left(\begin{smallmatrix}1&0\\0&4\end{smallmatrix}\right)$. Abbildung 8.3 zeigt diese Partitionierung. Hierbei wurde als Abfolgevektor $\widehat{\boldsymbol{\alpha}}^{P_1} = \begin{pmatrix}1 & 2\end{pmatrix}$ gewählt, um eine maximale Datenwiederverwendung zu erreichen.

Die Zuweisung s_1 mit der linksseitigen Variable $y_r[.]$ ist eine Eingabe- und Propagierungszuweisung. Da der Propagierungsvektor $\mathbf{d}^1_{s_1 s_{1_2}} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ beträgt, muss das $y_r[.]$-Datum für eine teilwortparallele Berechnung mittels der Operation *verteileTW* auf die anderen Teilwörter eines DvBs $\mathbf{y}_r[.]$ verteilt werden, wie in Abschnitt 6.5.4 ab Seite 128 vorgestellt wurde. Die $y_d[.]$-Daten können als DvBs $\mathbf{y}_d[.]$ direkt bei der nächsten Berechnung wiederverwendet werden, da der Abgängigkeitsvektor $\mathbf{d}_d = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ durch die Partitionierung nicht in zwei partitionierte Abhängigkeitsvektoren zerfällt. Der Abhängigkeitsvektor



(a) Iterationsraum (b) Reduzierter Abhängigkeitsgraph (RAG)

Abbildung 8.2: Iterationsraum und RAG des STAF-Algorithmus

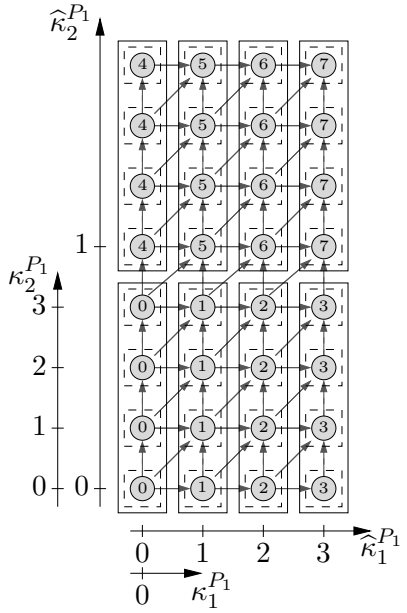


Abbildung 8.3: Copartitionierung des STAF-Algorithmus ($\mathbf{\Theta}^{S_1} = \mathbf{E}$)

Tabelle 8.2: Untersuchungsergebnisse für STAF-Algorithmus (in Takten $\times 10^3$)

	AMD Athlon	Intel Pentium II
Orig	124 188	126 995
MMX	42 547	45 154

$\mathbf{d}_u = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ zerfällt hingegen in zwei partitionierte Abhängigkeitsvektoren, weshalb eine Packoperation F_p zur Bestimmung der DvBs $\mathbf{y}_u[.]$ notwendig wird.

Die zweite Copartitionierungsstufe dient der Anpassung an das Rechenfeld mit einem Prozessorelement ($\mathbf{\Theta}^{P_2} = \mathbf{E}$) und der Ausnutzung der lokalen Register. Ein lokales Register steht hierfür noch zur Verfügung. Doch dieses ist nicht ausreichend für eine LS-2-Partition der Größe 1×2, weshalb wir für die Größe der LS-2-Partition $\mathbf{\Theta}^{S_2} = \mathbf{E}$ wählen müssen.

Die Realisierung wurde auf einem AMD Athlon Prozessor (700 MHz) und einem Intel Pentium II Prozessor (267 MHz) getestet. Für die Ermittlung der Berechnungsdauer des STAF-Algorithmus innerhalb des toast Programms wurde die Rabbit-Bibliothek [Hel00] verwendet.

Tabelle 8.2 zeigt die ermittelten Untersuchungsergebnisse für den Ausgangsalgorithmus (Orig) und den teilwortparallelen Algorithmus (MMX) für die beiden Zielarchitekturen. Die teilwortparallele Realisierung wurde mit Inlineassemblerbefehlen implementiert.

Für den AMD Athlon-Prozessor erhalten wir eine Beschleunigung von 2,96 und für den Intel Pentium-II-Prozessor eine Beschleunigung von 2,81. Diese Beschleunigungswerte sind deutlich geringer als die theoretische Beschleunigung von $B = 4$. Die Ursache hierfür ist die aufwändige teilwortparallele Realisierung des Rechenkerns des STAF-Algorithmus.

Weitere Untersuchungen zur Abbildung des STAF-Algorithmus auf eine solche Architektur sind in [SMC02a] und [SMC02b] zu finden.

8.2 Rechenfeld mit mehreren Prozessorelementen

Am Beispiel des Kantenerkennungsalgorithmus (kurz: KEA) soll die Abbildung eines Algorithmus auf ein Rechenfeld der Größe $\boldsymbol{\vartheta}^{\square} = \left(\begin{smallmatrix}2\\4\end{smallmatrix}\right)$ mit Teilwortparallelität in den Prozessorelementen vorgestellt werden. Der KEA besteht im Wesentlichen aus zwei Faltungen: einem horizontalen und einem vertikalen Sobelfilter. Der Algorithmus wurde vorverarbeitet, sodass er nur aus ein- und zweiwertigen Operationen besteht, welche unsere Zielarchitektur zur Verfügung stellt. Ziel der Vorverarbeitung war die maximale Wiederverwendung der Zwischenergebnisse und die Ausrichtung der Abhängigkeitsvektoren, siehe 7.1 auf Seite 141. Algorithmus 8.1 zeigt den KEA in Form eines UItA.

Die Zuweisungen $s_i \in \mathcal{S}$, $1 \le i \le 12$ des Algorithmus sind in den gemeinsamen Iterationsraum $\mathcal{I} = \left\{\mathbf{i} = \left(\begin{smallmatrix}x\\y\end{smallmatrix}\right) \mid 0 \le x \le X-1 \wedge 0 \le y \le Y-1\right\}$ eingebettet, wobei $X = 1280$ und $Y = 1024$ ist. Jede Zuweisung besitzt nur eine Teilzuweisung. Die Nicht-Null Abhängigkeitsvektoren des Algorithmus lauten: $\mathbf{d}^1_{s_1 s_3} = \left(\begin{smallmatrix}0\\2\end{smallmatrix}\right)$, $\mathbf{d}^1_{s_1 s_5} = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$, $\mathbf{d}^1_{s_2 s_4} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, $\mathbf{d}^1_{s_2 s_6} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$,

Algorithmus 8.1 Kantenerkennungsalgorithmus als uniform iterativer Algorithmus

$$
\begin{array}{lll}
s_1: & p\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = p_i\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_1 \\
s_2: & d\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = 2 \cdot p\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_2 = \mathcal{I}_1 \\
s_3: & h_1\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = p\left[\begin{smallmatrix}x\\y-2\end{smallmatrix}\right] + p\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_3 \\
s_4: & h_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = h_1\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] + d\left[\begin{smallmatrix}x\\y-1\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_4 = \mathcal{I}_3 \\
s_5: & v_1\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = p\left[\begin{smallmatrix}x-2\\y\end{smallmatrix}\right] + p\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_5 \\
s_6: & v_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = v_1\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] + d\left[\begin{smallmatrix}x-1\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_6 = \mathcal{I}_5 \\
s_7: & h_3\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = h_2\left[\begin{smallmatrix}x-2\\y\end{smallmatrix}\right] - h_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_7 \\
s_8: & h\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = |h_3\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right]|, & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_8 = \mathcal{I}_7 \\
s_9: & v_3\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = v_2\left[\begin{smallmatrix}x\\y-2\end{smallmatrix}\right] - v_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_9 = \mathcal{I}_7 \\
s_{10}: & v\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = |v_3\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right]|, & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_{10} = \mathcal{I}_7 \\
s_{11}: & m\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = h\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] + v\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_{11} = \mathcal{I}_7 \\
s_{12}: & o\left[\begin{smallmatrix}x-2\\y-2\end{smallmatrix}\right] = \min(255, m\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right]), & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_{12} = \mathcal{I}_7 \\
s_{13}: & p_o\left[\begin{smallmatrix}x-2\\y-2\end{smallmatrix}\right] = o\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_{13} = \mathcal{I}_7
\end{array}
$$

mit $\mathbf{i} = \left(\begin{smallmatrix}x\\y\end{smallmatrix}\right)$,
$\mathcal{I}_1 = \left\{\mathbf{i} \mid \begin{smallmatrix}0\le x<X\\0\le y<Y\end{smallmatrix}\right\}$, $\mathcal{I}_3 = \left\{\mathbf{i} \mid \begin{smallmatrix}0\le x<X\\2\le y<Y\end{smallmatrix}\right\}$, $\mathcal{I}_5 = \left\{\mathbf{i} \mid \begin{smallmatrix}2\le x<X\\0\le y<Y\end{smallmatrix}\right\}$ und $\mathcal{I}_7 = \left\{\mathbf{i} \mid \begin{smallmatrix}2\le x<X\\2\le y<Y\end{smallmatrix}\right\}$.

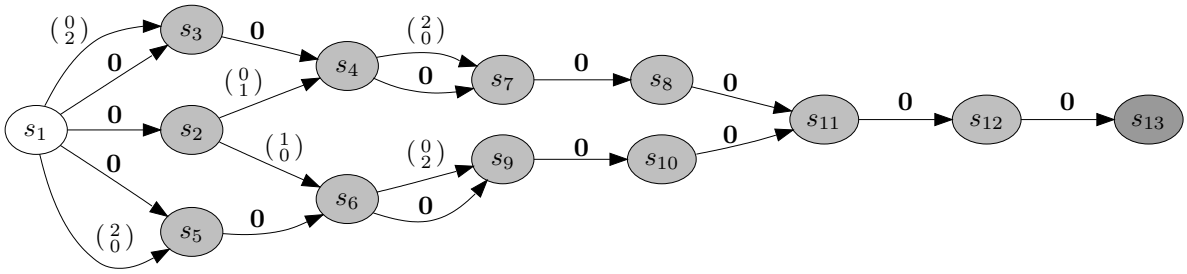


Abbildung 8.4: RAG des KEA mit 16 Bit Operationen

$\mathbf{d}^1_{s_4 s_7} = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$, und $\mathbf{d}^1_{s_6 s_9} = \left(\begin{smallmatrix}0\\2\end{smallmatrix}\right)$. Der reduzierte Abhängigkeitsgraph, siehe Abbildung 8.4, zeigt die Datenabhängigkeiten zwischen den Zuweisungen.

Die Eingabe $p_i[.]$ des KEA ist ein Bild der Größe $Y \times X = 1280 \times 1024$ und die Ausgabe $p_o[.]$ ist ein Bild der Größe $(Y-2) \times (X-2) = 1278 \times 1022$. Die Bilder sind zeilenweise $\mathbf{\Gamma}^M_{p_i} = \mathbf{\Gamma}^M_{p_o} = \begin{pmatrix}1 & 0\end{pmatrix}$ im Speicher abgelegt. Die Berechnung und die Speicherung der Daten des Algorithmus erfolgt mit einer Wortbreite von 16 Bit.

Der KEA soll auf ein schwach programmierbares Rechenfeld, siehe Abschnitt 2.5.2 ab Seite 19, abgebildet werden. Das Rechenfeld hat die Größe $\boldsymbol{\vartheta}^{\square} = \left(\begin{smallmatrix}2\\4\end{smallmatrix}\right)$. Für den Datentransfer zwischen den schwach programmierbaren Prozessorelementen (SPPEs) existieren bidirektionale Kanäle $\mathcal{Q}^{\text{bi}} = \{\mathbf{q}_1 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \mathbf{q}_2 = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\}$ zwischen den horizontal und vertikal benachbarten SPPEs. Das Lesen der Daten von der Speicherarchitektur und Schreiben der Daten in die Speicherarchitektur ist nur mit ausgerichteten Speicherzugriffen auf

Tabelle 8.3: Eigenschaften der Funktionseinheiten (Eingabeverzögerung $o_{m(s)} = 1$)

FE	Fläche c_m	Funktionen F_s mit Latenz $l_{m(s)}$ in den Klammern
m_1	4	abs(2), min(3), shr(1), add/sub(1), pack1(1), pack2(1)
m_2	2	shr(1), pack1(1), pack2(1)
m_3	3	add/sub(1), abs(2), min(3)

Basis von 64 Bit DvBs möglich. Jedes SPPE kann 64 bit Daten verarbeiten.

Drei Funktionseinheiten m_1, m_2 und m_3 werden bei der Abbildung betrachtet. Die Parameter dieser FEs sind in Tabelle 8.3 angegeben. Die Funktion *shr* wird hierbei für die Multiplikation-mit-Zwei-Operation verwendet. Die zwei Funktionseinheiten m_2 und m_3 erlauben Teilwortparallelität auf 64 Bit DvBs mit vier 16 Bit Teilwörtern. Packbefehle zum Umorganisieren der Teilwörter stehen ebenfalls zur Verfügung.

8.2.1 Teilwortparallelisierung

Die erste Copartitionierungsstufe verwenden wir für die Ausnutzung der Teilwortparallelität in den SPPEs, wobei für die Partitionsmatrix $\mathbf{\Theta}^{S_1} = \mathbf{E}$ gilt. Die Zielarchitektur bietet eine $B = 4$-fache Teilwortparallelität. Der gemeinsame Iterationsraum $\mathcal{I}$ des KEA ist zweidimensional. Somit sind zwei Partitionierungsmatrizen $\mathbf{\Theta}_1^{P_1} = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ und $\mathbf{\Theta}_2^{P_1} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ möglich. Die Auswahl der Partitionsmatrix ist abhängig von der Anzahl an Packoperationen, welche durch die Teilwortparallelisierung entstehen.

Unabhängig von der Partitionsmatrix zerfallen drei Abhängigkeitsvektoren in zwei partitionierte Abhängigkeitsvektoren. Bei der Teilwortparallelisierung mit Matrix $\mathbf{\Theta}_1^{P_1} = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ sind das die Abhängigkeitsvektoren $\mathbf{d}^1_{s_1 s_5}$, $\mathbf{d}^1_{s_2 s_6}$ und $\mathbf{d}^1_{s_4 s_7}$ und bei der Teilwortparallelisierung mit Matrix $\mathbf{\Theta}_2^{P_1} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ sind es die Vektoren $\mathbf{d}^1_{s_1 s_3}$, $\mathbf{d}^1_{s_2 s_4}$ und $\mathbf{d}^1_{s_6 s_9}$. Somit erhalten wir immer drei Packoperationen aufgrund des Zerfalls der Abhängigkeitsvektoren.

Da jede Zuweisung des KEA nur eine Teilzuweisung hat, ergeben sich keine zusätzlichen Packoperationen aufgrund mehrerer Teilzuweisungen.

Die Anzahl der Packoperationen aufgrund der Teilwortparallelisierung der Ein- und Ausgabedaten ist stark abhängig davon, ob die Ein- oder Ausgabe-DvBs bezüglich der Berechnungs-DvBs gleichartig angeordnet sind oder nicht, siehe Abschnitt 6.5 ab Seite 109. Bei der Teilwortparallelisierung mit Partitionsmatrix $\mathbf{\Theta}_1^{P_1}$ sind die DvBs gleichartig angeordnet, bei Verwendung der Partitionsmatrix $\mathbf{\Theta}_2^{P_1}$ jedoch nicht. Somit wählen wir die Matrix $\mathbf{\Theta}_1^{P_1}$ für die Teilwortparallelisierung. In Abbildung 8.5(a) ist diese Partitionierung des Iterationsraums dargestellt.

Algorithmus 8.2 zeigt den teilwortparallelen Kantenerkennungsalgorithmus. Die dick dargestellten Variablen $\mathbf{y}[\widehat{\boldsymbol{\kappa}}^1]$ sind DvBs mit vier Teilwörtern. Die Additions-, Subtraktions-, Multiplikations-, Absolutwert- und die Min-Operationen sind nun teilwortparallele Operationen. Die Packoperation F_p der neuen Zuweisungen s_{5_1}, s_{6_1}, s_{7_1} und s_{13_1} entspricht der in Abschnitt 6.2 ab Seite 91 vorgestellten Packoperation $\mathbf{y} = F_p\big(\mathbf{x}_1, \mathbf{x}_2, d_1^P\big)$. Die

Algorithmus 8.2 Kantenerkennungsalgorithmus mit TWP-Operationen

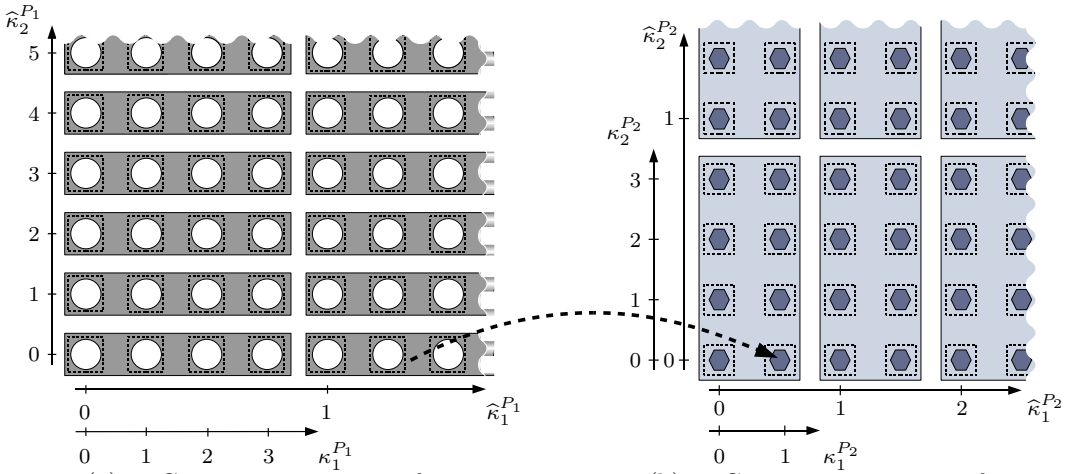
$$
\begin{aligned}
s_1: &\quad \mathbf{p}[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{p}_i[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_1^1\\
s_2: &\quad \mathbf{d}[\widehat{\boldsymbol{\kappa}}^{P_1}] = 2 \cdot \mathbf{p}[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_1^1\\
s_3: &\quad \mathbf{h}_1[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{p}[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}0\\2\end{smallmatrix}\right)] + \mathbf{p}[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_3^1\\
s_4: &\quad \mathbf{h}_2[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{h}_1[\widehat{\boldsymbol{\kappa}}^{P_1}] + \mathbf{d}[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_3^1\\
s_{5_1}: &\quad \mathbf{t}_{5,1}[\widehat{\boldsymbol{\kappa}}^{P_1}] = F_p\left(\mathbf{p}[\widehat{\boldsymbol{\kappa}}^{P_1}], \mathbf{p}[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)], 2\right), && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_5^1\\
s_5: &\quad \mathbf{v}_1[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{t}_{5,1}[\widehat{\boldsymbol{\kappa}}^{P_1}] + \mathbf{p}[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_5^1\\
s_{6_1}: &\quad \mathbf{t}_{6,1}[\widehat{\boldsymbol{\kappa}}^{P_1}] = F_p\left(\mathbf{d}[\widehat{\boldsymbol{\kappa}}^{P_1}], \mathbf{d}[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)], 1\right), && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_5^1\\
s_6: &\quad \mathbf{v}_2[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{v}_1[\widehat{\boldsymbol{\kappa}}^{P_1}] + \mathbf{t}_{6,1}[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_5^1\\
s_{7_1}: &\quad \mathbf{t}_{7,1}[\widehat{\boldsymbol{\kappa}}^{P_1}] = F_p\left(\mathbf{h}_2[\widehat{\boldsymbol{\kappa}}^{P_1}], \mathbf{h}_2[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)], 2\right), && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_7^1\\
s_7: &\quad \mathbf{h}_3[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{t}_{7,1}[\widehat{\boldsymbol{\kappa}}^{P_1}] - \mathbf{h}_2[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_7^1\\
s_8: &\quad \mathbf{h}[\widehat{\boldsymbol{\kappa}}^{P_1}] = |\mathbf{h}_3[\widehat{\boldsymbol{\kappa}}^{P_1}]|, && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_7^1\\
s_9: &\quad \mathbf{v}_3[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{v}_2[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}0\\2\end{smallmatrix}\right)] - \mathbf{v}_2[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_7^1\\
s_{10}: &\quad \mathbf{v}[\widehat{\boldsymbol{\kappa}}^{P_1}] = |\mathbf{v}_3[\widehat{\boldsymbol{\kappa}}^{P_1}]|, && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_7^1\\
s_{11}: &\quad \mathbf{m}[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{h}[\widehat{\boldsymbol{\kappa}}^{P_1}] + \mathbf{v}[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_7^1\\
s_{12}: &\quad \mathbf{o}[\widehat{\boldsymbol{\kappa}}^{P_1}] = \mathbf{min}(\mathbf{255}, \mathbf{m}[\widehat{\boldsymbol{\kappa}}^{P_1}]), && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_7^1\\
s_{13,1}: &\quad \mathbf{t}_{13,1}[\widehat{\boldsymbol{\kappa}}^{P_1}] = F_p\left(\mathbf{o}[\widehat{\boldsymbol{\kappa}}^{P_1}], \mathbf{o}[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)], 2\right), && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{13}^1\\
s_{13}: &\quad \mathbf{p}_o[\widehat{\boldsymbol{\kappa}}^{P_1} - \left(\begin{smallmatrix}1\\2\end{smallmatrix}\right)] = \mathbf{t}_{13,1}[\widehat{\boldsymbol{\kappa}}^{P_1}], && \widehat{\boldsymbol{\kappa}}^{P_1} \in \widehat{\mathcal{K}}_{13}^1
\end{aligned}
$$

mit $\widehat{\boldsymbol{\kappa}}^{P_1} = \begin{pmatrix} \widehat{\kappa}_1^{P_1} \\ \widehat{\kappa}_2^{P_1} \end{pmatrix}$,

$\widehat{\mathcal{K}}_1^1 = \left\{ \widehat{\boldsymbol{\kappa}}^{P_1} \;\middle|\; \begin{smallmatrix} 0 \le \widehat{\kappa}_1^{P_1} \le \lceil \frac{X-1}{4} \rceil \\ 0 \le \widehat{\kappa}_2^{P_1} < Y \end{smallmatrix} \right\}$, $\widehat{\mathcal{K}}_3^1 = \left\{ \widehat{\boldsymbol{\kappa}}^{P_1} \;\middle|\; \begin{smallmatrix} 0 \le \widehat{\kappa}_1^{P_1} \le \lceil \frac{X-1}{4} \rceil \\ 2 \le \widehat{\kappa}_2^{P_1} < Y \end{smallmatrix} \right\}$,

$\widehat{\mathcal{K}}_5^1 = \left\{ \widehat{\boldsymbol{\kappa}}^{P_1} \;\middle|\; \begin{smallmatrix} 0 \le \widehat{\kappa}_1^{P_1} \le \lceil \frac{X-1}{4} \rceil \\ 0 \le \widehat{\kappa}_2^{P_1} < Y \end{smallmatrix} \right\}$, $\widehat{\mathcal{K}}_7^1 = \left\{ \widehat{\boldsymbol{\kappa}}^{P_1} \;\middle|\; \begin{smallmatrix} 0 \le \widehat{\kappa}_1^{P_1} \le \lceil \frac{X-1}{4} \rceil \\ 2 \le \widehat{\kappa}_2^{P_1} < Y \end{smallmatrix} \right\}$ und

$\widehat{\mathcal{K}}_{13}^1 = \left\{ \widehat{\boldsymbol{\kappa}}^{P_1} \;\middle|\; \begin{smallmatrix} 1 \le \widehat{\kappa}_1^{P_1} \le \lceil \frac{X+1}{4} \rceil \\ 2 \le \widehat{\kappa}_2^{P_1} < Y \end{smallmatrix} \right\}$.



(a) 1. Copartitionierungsstufe (b) 2. Copartitionierungsstufe

Abbildung 8.5: Partitionierung des Iterationsraums

vierte Packzuweisung s_{13_1} entstand aufgrund der Korrektur des unausgerichteten Speicherzugriffs der Ausgabezuweisung s_{13} des Algorithmus 8.1. In Abschnitt 6.5.3 wurde die Transformation eines unausgerichteten Speicherzugriffs in einen ausgerichteten Speicherzugriff vorgestellt.

Abbildung 8.6 zeigt den RAG des teilwortparallelen KEA 8.2. Die Knoten der zusätzlichen Packzuweisungen sind mittels einer dickeren Umrandung hervorgehoben.

8.2.2 Abbildung auf das Rechenfeld

Die Parallelität der zweiten Ebene erhalten wir durch das parallele Arbeiten der Prozessorelemente. Die beiden Partitionierungsmatrizen $\mathbf{\Theta}_1^{P_2} = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 4 \end{smallmatrix}\right)$ und $\mathbf{\Theta}_2^{P_2} = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 2 \end{smallmatrix}\right)$ sind für eine effiziente Abbildung des KEAs mit TWP-Operationen (Alg. 8.2) auf das Rechenfeld mit Größe $\boldsymbol{\vartheta}^{\square} = \left(\begin{smallmatrix} 2 \\ 4 \end{smallmatrix}\right)$ möglich. Abbildung 8.5(b) illustriert die zweite Copartitionierungsstufe mit $\mathbf{\Theta}_1^{P_2} = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 4 \end{smallmatrix}\right)$, wobei für die Partitionsmatrix $\mathbf{\Theta}^{S_2} = \mathbf{E}$ gewählt wurde. Acht Iterationen gehören zu einer Copartition der zweiten Copartitionierungsstufe. Dabei ist zu beachten, dass jede Iteration $\widehat{\boldsymbol{\kappa}}^{P_1}$ eine Copartition der ersten Copartitionierungsstufe ist.

Ein weiterer Aspekt der Auswahl der Partitionsmatrix $\mathbf{\Theta}^{P_2}$ ist der Grad an Datenwiederverwendung in der ersten Abarbeitungsrichtung $\widehat{\alpha}_1^{P_2}$. Für den KEA erhalten wir die folgenden Transferwerte $U^{S_2}\left(\widehat{\mathbf{d}}^{P_2}\right)$ abhängig von der Partitionsmatrix $\mathbf{\Theta}^2$:

$$\begin{aligned}
\mathbf{\Theta}_1^2 = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 4 \end{smallmatrix}\right): \quad & U^{S_2}(\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)) = 128, \quad U^{S_2}(\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)) = 16, \quad U^{S_2}(\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)) = 10, \\
\mathbf{\Theta}_2^2 = \left(\begin{smallmatrix} 4 & 0 \\ 0 & 2 \end{smallmatrix}\right): \quad & U^{S_2}(\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)) = 128, \quad U^{S_2}(\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)) = 8, \quad U^{S_2}(\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)) = 20.
\end{aligned}$$

Der Transferwert $U^{S_2}\left(\widehat{\mathbf{d}}^{P_2}\right)$, siehe Seite 175 ff., gibt an, wie viele Daten des Algorithmus auf Basis von DvBs von Partition $\widehat{\boldsymbol{\kappa}}^{P_2}$ in Partition $\widehat{\boldsymbol{\kappa}}^{P_2} + \widehat{\mathbf{d}}^{P_2}$ wiederverwendet werden.

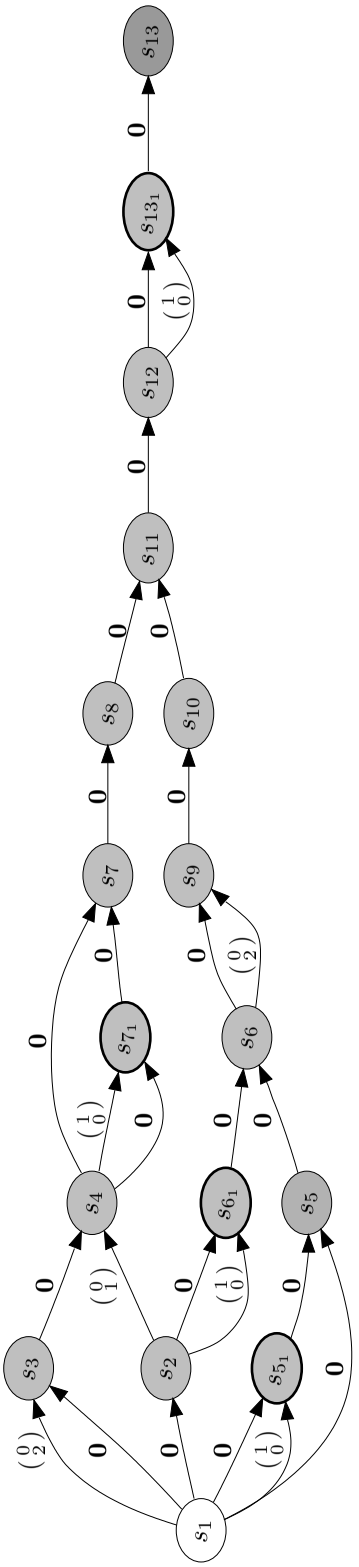


Abbildung 8.6: RAG des teilwortparallelisierten KEA, siehe Alg. 8.2

Tabelle 8.4: Experimentelle Ergebnisse für KEA mit TWP auf einem 4×2 Rechenfeld

PE Ressourcen		$\boldsymbol{\tau}^{\kappa} = \left(\boldsymbol{\kappa}^{S_2} \quad \boldsymbol{\kappa}^{P_2} \quad \widehat{\boldsymbol{\kappa}}^{P_2}\right)$	Abarbeitungsdauer
R1	2x m_1	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\1\end{pmatrix} \quad \begin{pmatrix}4096\\8\end{pmatrix}\right)$	331 783
R2	3x m_1	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}2560\\5\end{pmatrix}\right)$	207 358
R3	4x m_1	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}2048\\4\end{pmatrix}\right)$	165 896
R4	1x m_2, 1x m_3	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}5120\\10\end{pmatrix}\right)$	414 731
R5	1x m_2, 2x m_3	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\6\end{pmatrix} \quad \begin{pmatrix}2560\\5\end{pmatrix}\right)$	207 368
R6	2x m_2, 1x m_3	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}5120\\10\end{pmatrix}\right)$	414 727
R7	2x m_2, 2x m_3	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}2560\\5\end{pmatrix}\right)$	207 367
R8	1x m_2, 3x m_3	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\6\end{pmatrix} \quad \begin{pmatrix}2560\\5\end{pmatrix}\right)$	207 368
R9	2x m_2, 3x m_3	$\left(\begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\end{pmatrix} \quad \begin{pmatrix}2048\\4\end{pmatrix}\right)$	165 895

Die lokale Wiederverwendung $U^{S_2}\left(\widehat{\mathbf{d}}^{P_2} = \left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\right) = 128$ ist für beide Partitionierungsmatrizen gleich. Die beste Datenwiederverwendung erhalten wir mit Partitionsmatrix $\boldsymbol{\Theta}_2^2$ und Abarbeitungsreihenfolge $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix}2 & 1\end{pmatrix}$. In diesem Fall werden $U^{S_2}\left(\widehat{\mathbf{d}}^{P_2} = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\right) = 20$ Daten in der nachfolgenden Partition wiederverwendet. Sollte diese Abarbeitungsreihenfolge nicht möglich sein, so sollte die Partitionsmatrix $\boldsymbol{\Theta}_1^2$ mit Abarbeitungsreihenfolge $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix}1 & 2\end{pmatrix}$ verwendet werden.

8.2.3 Ablaufplanung und Abbildung auf Funktionseinheiten

Unter Verwendung eines gemischt-ganzzahligen linearen Programms, entwickelt an der Universität Erlangen-Nürnberg [TTZ97][HT04], wurde die ressourcenbeschränkte Ablaufplanung durchgeführt. Die Tabelle 8.4 zeigt die ermittelten globalen Ablaufvektoren $\boldsymbol{\tau}^{\kappa} = \left(\boldsymbol{\kappa}^{S_2} \quad \boldsymbol{\kappa}^{P_2} \quad \widehat{\boldsymbol{\kappa}}^{P_2}\right)$ für den KEA, welcher auf ein 4×2 Rechenfeld mit Teilwortparallelität in den PEs abgebildet wurde, für verschiedene PE-Architekturen. Der spaltenweise Ablaufplan $\widehat{\boldsymbol{\alpha}}^{P_2} = \begin{pmatrix}2 & 1\end{pmatrix}$ wurde für alle Experimente verwendet. Die Abarbeitungsdauer für die Experimente bei einem Eingabebild der Größe 1280×1024 sind als Taktzyklen angegeben. In Abbildung 8.7 ist der lokale Ablaufplan für Realisierung R4 als Gantt-Plan angegeben. Die Kästchen mit der gleichen Hintergrundfarbe sind die Zuweisungen einer Iteration.

Die ersten drei PE-Architekturen von Tabelle 8.4 unterscheiden sich nur bezüglich der Anzahl an Funktionseinheiten m_1. Die Beschleunigung beträgt 2,0 für R2 und 1,43 für R3 im Vergleich zu R1. Diese Resultate sind gleich (2,0 bei R3) oder in der Nähe (1,5 bei R2) der theoretisch maximalen Beschleunigung. Die anderen PE-Architekturen nutzen die Funktionseinheiten m_2 und m_3, welche unterschiedliche Funktionalitäten haben, siehe Tabelle 8.3. Aus den Ergebnissen von 8.4 wird ersichtlich, dass die PE-Architekturen R4 und R6 sowie die PE-Architekturen R5, R7 und R8 fast die gleiche Abarbeitungsdauer haben. Folglich sind die PE-Architekturen R4 und R5 die besseren Lösungen, denn diese Architekturen benötigen weniger Funktionseinheiten. Abschließend sollen die Ergebnisse der Rechenfeldrealisierung von Tabelle 8.4 mit den experimentellen Ergebnissen einer Realisierung auf einem Einzelprozessor mit und ohne Teilwortparallelität, siehe

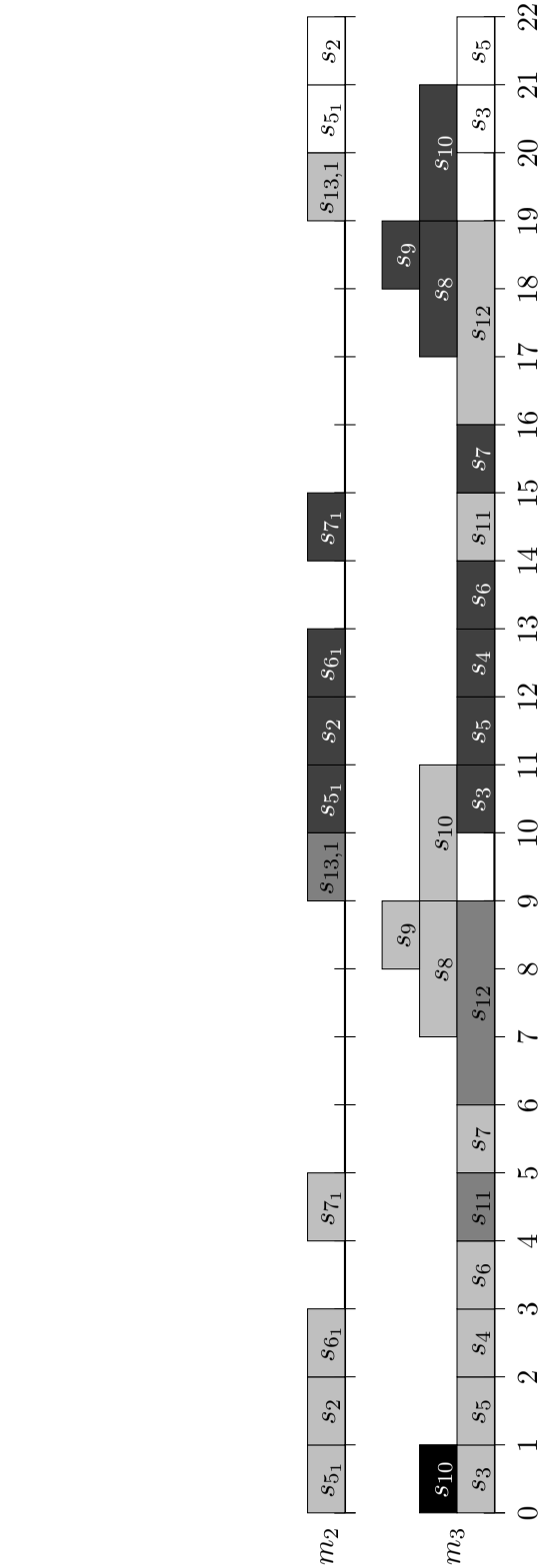


Abbildung 8.7: Gantt-Chart des KEA, welches auf ein 4×2 Rechenfeld mit je einem FE m_2 und m_3 pro PE abgebildet wurde (Iterationsintervall $\lambda = 10$)

Tabelle 8.5: Experimentelle Ergebnisse für KEA auf einem Einzelprozessor

PE Ressourcen		$\boldsymbol{\tau}^{\kappa} = \widehat{\boldsymbol{\kappa}}^{P_1}$	Abarbeitungsdauer
KEA mit TWP (Algorithmus 8.2)			
R3	4x m_1	$\begin{pmatrix}4 & 1284\end{pmatrix}$	1 314 824
R4	1x m_2, 1x m_3	$\begin{pmatrix}10 & 3210\end{pmatrix}$	3 287 048
R9	2x m_2, 3x m_3	$\begin{pmatrix}4 & 1284\end{pmatrix}$	1 314 823
KEA ohne TWP (Algorithmus 8.1)			
R3	4x m_1	$\begin{pmatrix}3 & 33846\end{pmatrix}$	3 938 311
R4	1x m_2, 1x m_3	$\begin{pmatrix}10 & 12820\end{pmatrix}$	13 127 689
R9	2x m_2, 3x m_3	$\begin{pmatrix}4 & 5128\end{pmatrix}$	5 251 077

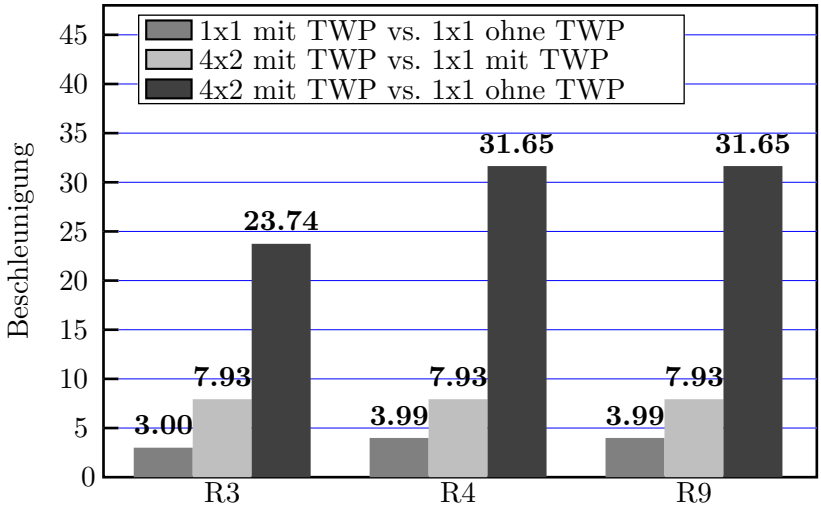


Abbildung 8.8: Beschleunigungen zwischen KEA Versionen

Tabelle 8.5, verglichen werden.

Abbildung 8.8 zeigt ein Balkendiagramm, welches die Beschleunigung der Rechenfeldrealisierung (4×2) im Vergleich zur Realisierung auf einem Einzelprozessor (1×1) mit und ohne Teilwortparallelität in den PE-Ressourcen R3, R4 und R9 darstellt. Die Beschleunigung für die Realisierung auf einem Einzelprozessor mit Teilwortparallelität im Vergleich zur Realisierung ohne Teilwortparallelität beträgt 3,00 für R3 und 3,99 für R4 und R9. Die fast maximale Beschleunigung von 3,99 (max: 4,00) für R4 und R9 ergibt sich aufgrund der Abarbeitung der Packbefehle in den Funktionseinheiten m_2, welche bei der Realisierung ohne Teilwortparallelität wenig genutzt wurde. Die Beschleunigung für die Parallelisierung auf dem Rechenfeld beträgt 7,93 für alle PE-Ressourcen. Damit erreichen wir fast (99,1 %) die theoretisch maximale Beschleunigung von 8,00. Somit ergibt sich eine maximale Beschleunigung von $3,99 \cdot 7,93 = 31,65$ für die Rechenfeldrealisierung mit Teilwortparallelität im Vergleich zur Realisierung auf einem Einzelprozessor ohne Teilwortparallelität. Werden die zusätzlichen Packbefehle wie bei R3 nicht in einer extra Funktionseinheit abgearbeitet, so ist die Gesamtbeschleunigung mit $3,00 \cdot 7,93 = 23,74$ deutlich geringer.

8.3 Zusammenfassung

Anhand von drei Beispielen wurde die Abbildung von drei verschiedenen Algorithmen auf ein Prozessorelement mit Teilwortparallelität oder ein Rechenfeld mit Teilwortparallelität illustriert. Dabei wurde die in Kapitel 7 vorgestellte Abbildungsstrategie verwendet.

Das FIR-Filter wurde auf ein Intel Core2 Prozessor abgebildet, welcher Teilwortparallelität erlaubt. Neben der Ausnutzung der Teilwortparallelität wurden auch die acht lokalen Register bewusst ausgenutzt. Das führte zu einer Beschleunigung des teilwortparallelen Algorithmus im Vergleich zum Algorithmus mit Einzeldaten von 5,67 bis 6,50 abhängig von der Filterlänge J.

Der STAF-Algorithmus, ein wichtiger Algorithmus auf dem GSM Standard, wurde auf den AMD Athlon- bzw. den Intel Pentium-II-Prozessor abgebildet. Die lokalen Register wurden für den aufwändigen Berechnungskern benötigt. Somit konnte die lokale Datenwiederverwendung leider nicht weiter verbessert werden. Die ermittelte Beschleunigung der teilwortparallelen Realisierung gegenüber der Realisierung mit Einzeldaten beträgt 2,96 für den AMD Athlon-Prozessor und 2,81 für den Intel Pentium-II-Prozessor.

Abschließend wurde die Abbildung des Kantenerkennungsalgorithmus (KEA) auf das schwach programmierbare Rechenfeld (SPRF) vorgestellt. Das Rechenfeld hat die Größe von $\boldsymbol{\vartheta}^{\square} = \binom{2}{4}$. Die schwach programmierbaren Prozessorelemente des Rechenfelds erlauben Teilwortparallelität mit den Faktor $B = 4$. Als maximale Beschleunigung erreichten wir bei der Abbildung des KEAs auf das SPRF einen Faktor von 31,65 gegenüber einer Realisierung auf einem Prozessorelement ohne Teilwortparallelität. Diese Beschleunigung liegt in der Nähe der theoretischen Beschleunigung von $(2 \cdot 4) \cdot 4 = 32$.

Kapitel 9

Zusammenfassung und Ausblick

9.1 Zusammenfassung

In dieser Arbeit wurde eine Abbildungsstrategie zur Abbildung von Algorithmen der Algorithmenklasse der uniform iterativen Algorithmen auf Rechenfelder mit Teilwortparallelität vorgestellt. Bei der Abbildung der Algorithmen auf die Zielarchitektur erfolgte auch eine Anpassung an Speicherarchitektur mit der das Rechenfeld verbunden ist. Anhand von drei Beispielen wurde die Effektivität dieser Abbildungsstrategie illustriert und deren Potential aufgezeigt.

Nach der Vorstellung der Zielarchitektur im Kapitel 2 und der Algorithmenklasse im Kapitel 3 wurden im Kapitel 4 die beiden Algorithmentransformationen Reindizierung und mehrstufige Partitionierung vorgestellt. Beide Transformationen sind die Grundmethoden für den in der Arbeit entwickelten Abbildungsprozess. Dabei wurde in Abschnitt 4.2.2 eine neue Methode zur Partitionierung von Datenabhängigkeiten vorgestellt. Die mehrstufige Partitionierung wurde in Kapitel 5 zur mehrstufigen modifizierten Copartitionierung (MMC) erweitert. Die MMC beinhaltet auch die Ablaufplanung und die Bindung der Funktionseinheiten der Prozessorelemente an die Teilzuweisungen des Algorithmus. Die allgemeine Beschreibung der Ablaufplanung wurde um die Leerlaufverzögerung erweitert, um für jede Partitionierung des Algorithmus eine konfliktfreie Ablaufplanung definieren zu können.

In Kapitel 6 wurde die Transformation eines Algorithmus mit Einzeldatenoperationen in einen Algorithmus mit teilwortparallelen Operationen vorgestellt. Hierfür wurde die erste Stufe der mehrstufigen modifizierten Copartitionierung verwendet. Die notwendigen Packoperationen für die teilwortparallele Version des Algorithmus wurden systematisch ermittelt. Weiterhin wurde die Transformation der Ein- und Ausgabezuweisungen des Ausgangsalgorithmus abgeleitet, welche von der Art der Speicherung der Daten im Hauptspeicher sowie deren Lese- und Schreibmöglichkeiten abhängen.

Die Präsentation der gesamten Abbildungsstrategie fand in Kapitel 7 statt. Die Abbildungsstrategie beinhaltet zwei Schritte, die „Vorverarbeitung“ und die „mehrstufige

modifizierte Copartitionierung“.

Die „Vorverarbeitung“ besteht aus den beiden Teilschritten „Reindizierung“ und „Anpassung an die Algorithmenkonvention“. Für die „Reindizierung“ wurde eine mehrstufige Optimierung auf Basis der ganzzahligen linearen Programmierung entwickelt, mit der die Ausrichtung und die Länge der Abhängigkeitsvektoren, die Anzahl der Packoperationen und das Volumen des den Iterationsraum umschließenden Hyperquarders optimiert werden kann. Mit der „Anpassung an die Algorithmenkonvention“ wird der Algorithmus so umformuliert, dass er die Algorithmenkonvention von Abschnitt 3.4 erfüllt. Durch die Vorverarbeitung wird der Algorithmus so verändert, dass der veränderte Algorithmus mittels der mehrstufigen modifizierten Copartitionierung schnell und effizient auf die Zielarchitektur abgebildet werden kann.

Der Abbildungsschritt der „mehrstufigen modifizierten Copartitionierung“ unterteilt sich in die einzelnen Copartitionierungsstufen und die abschließende Ablaufplanung. Mit jeder Stufe erfolgt eine Anpassung des Algorithmus an bestimmte Parameter der Zielarchitektur:

- **1. Copartitionierungsstufe:**
  - Ausnutzung der Teilwortparallelität
- **2. Copartitionierungsstufe:**
  - Anpassung an Rechenfeld,
  - Ausnutzung der lokalen Register in den Prozessorelementen
- **3. bis n. Copartitionierungsstufe:**
  - Anpassung an die Speicherarchitektur
- **Ablaufplanung:**
  - Bindung der Teilzuweisungen des Algorithmus an die Funktionseinheiten in den Prozessorelementen,
  - Ablaufplanung unter Beachtung der Latenzen und der Eingabeverzögerungen der Funktionseinheiten.

In der Arbeit wurde gezeigt, dass der Algorithmus systematisch an die Parameter der Zielarchitektur angepasst werden kann. Für die Bestimmung der Parameter zur Ausnutzung der lokalen Register in den Prozessorelementen und zur Anpassung an die Speicherarchitektur wurde die Transfermenge eingeführt, welche die Daten einer Zuweisung enthält, die zwischen zwei Partitionen transferiert werden müssen.

Abschließend wurde im Kapitel 8 anhand des FIR-Filters und des STAF-Algorithmus die Abbildung auf ein Prozessorelement mit Teilwortparallelität und anhand des Kantenalgorithmus die Abbildung auf ein Rechenfeld mit Teilwortparallelität illustriert. Die ermittelten Abbildungen auf die Zielarchitektur mit Hilfe der vorgestellten Abbildungsstrategie führten zu einer effizienten Abarbeitung des Algorithmus auf der jeweiligen Zielarchitektur.

9.2 Ausblick

Nachfolgend werden mögliche Weiterentwicklungen der in Kapitel 7 vorgestellten Abbildungsstrategie und potentielle Verknüpfungspunkte mit anderen Abbildungsmethoden vorgestellt.

Die Befehlssätze von Architekturen mit Teilwortparallelität besitzen meist auch teilwortparallele Befehle, die nicht dem Eine-Operation-Mehrere-Daten-Prinzip, siehe Abschnitt 2.2 ab Seite 9, folgen. In Abschnitt 2.2 wurden diese Befehle als Sonderbefehle bezeichnet. Diese Befehle werden mit der vorgestellten Abbildungsstrategie nur unzureichend genutzt. Wie in [SMC03] gezeigt wurde, kann bei Verwendung dieser Sonderbefehle die Kausalitätsbedingungen weicher formuliert werden, was die Abarbeitung der Algorithmen auf der Zielarchitektur weiter beschleunigt.

Pieter Op de Beeck beschäftigte sich in den Arbeiten [OdBGB+03] und [OdBMCD03] mit der Organisation der Daten im Speicher. Hierbei untersuchte er für Anwendungen mit mehreren Algorithmen, wie die Daten auf Basis von Teilwörtern im Speicher organisiert werden müssen, damit Speicherzugriffe und Packbefehle zwischen den einzelnen Algorithmen der Anwendung vermieden werden. Die dabei ermittelten Vorgaben für die Abarbeitung der jeweiligen Algorithmen könnten nun als Restriktionen für die Teilwortparallelisierung der einzelnen Algorithmen der Anwendung verwendet werden. Somit würde man eine sowohl speicheroptimierte als auch teilwortparallel effiziente Realisierung für die Anwendung erhalten. Weiterhin könnten in seine Untersuchungen die Packbefehle, welche durch die Teilwortparallelisierung der einzelnen Algorithmen entstehen, mit einbezogen werden, indem diese mittels der Methoden aus Kapitel 6 bestimmt werden.

Die in Kapitel 7 vorgestellte Abbildungsstrategie betrachtet die Anpassung an die Speicherarchitektur nach der Parallelisierung des Algorithmus. Im Gegensatz dazu untersucht die Datentransfer- und Speicheruntersuchung (DTSE - Data Transfer and Storage Exploration) [CWD+98] die Speicheranpassung vor der Ablaufplanung/Parallelisierung des Algorithmus. Hierbei werden weitere Aspekte, wie der Aufbau und die Arbeitsweise der Speicher, in die Untersuchung mit einbezogen. Außerdem erfolgt bei der DTSE die Anpassung an die Speicherarchitektur nicht nur bezüglich eines einzelnen Alorithmus sondern bezüglich aller Algorithmen einer Anwendung. Beide Methoden können miteinander kombiniert werden, ähnlich wie in [SCM00] präsentiert. Dabei wird zuerst die Anwendung, welche i. A. mehrere Algorithmen enthält, mittels der DTSE an die Hintergrundspeicher (engl.: Background Memory) angepasst, wobei bei dieser Anpassung zwischen plattformunabhängiger und plattformabhängiger Hintergrundspeicheroptimierung (engl.: Background Memory Optimization) unterschieden wird. Im zweiten Schritt kann dann die mehrstufige modifizierte Copartitionierung genutzt werden, um die Algorithmen der Anwendung einzeln bezüglich der Parallelverarbeitung und der lokalen Speicheranpassung zu optimieren. Die Ergebnisse der Hintergrundspeicheroptimierung sind dabei Randbedingungen für die mehrstufige modifizierte Copartitionierung (MMC). Abbildung 9.1 illustriert dieses Vorgehen.

Eine weitere interessante Verknüpfung könnte mit der in [ACE08] vorgestellten Methode

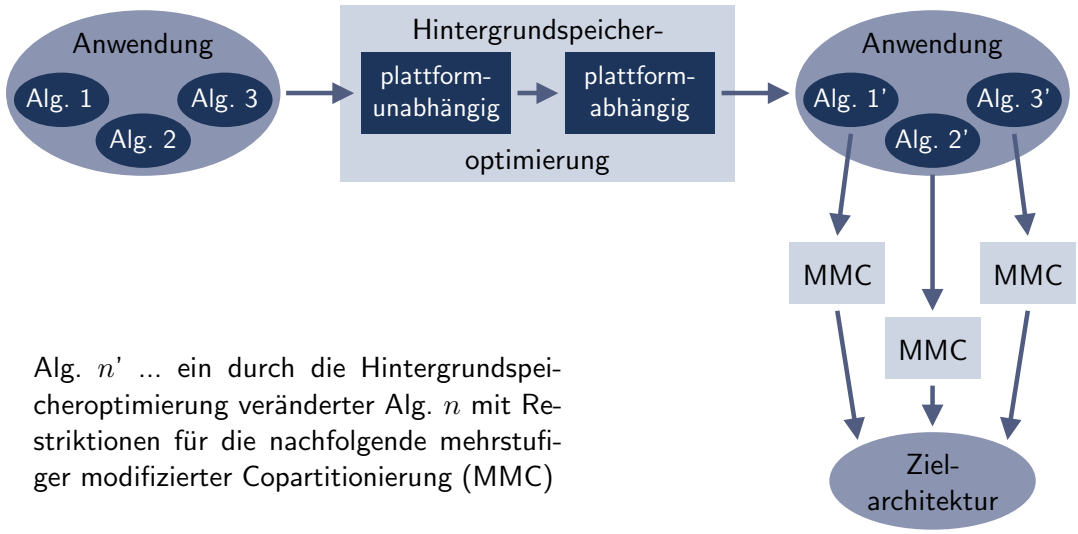


Abbildung 9.1: Kombination von Hintergrundspeicheroptimierung und mehrstufiger modifizierter Copartitionierung (MMC)

zur Parallelisierung von Algorithmen auf Kahn Prozessnetzwerken (engl.: Kahn Process Networks) entstehen. Kahn Prozessnetzwerke sind gut geeignet für die systematische Abbildung auf Mehrkern-Architekturen [SKD02], wie z. B. dem Cell-Prozessor [Int07b]. Die Methode zur Parallelisierung basiert wie die Abbildungsstrategie in dieser Arbeit auf einer Beschreibung der Algorithmen als Polyeder. Somit könnte eine systematische Abbildungsstrategie für die Abbildung von Algorithmen auf Mehrkern-Architekturen mit Teilwortparallelität entwickelt werden. Zielarchitektur wäre zum Beispiel der Cell-Prozessor mit seinen acht synergischen Verarbeitungselementen (engl.: Synergistic Processing Elements), welche Teilwortparallelität auf einem 128 Bit breiten Datenpfad erlauben. Die Teilwortparallelität würde mit den Methoden dieser Arbeit und die Parallelität der Mehrkern-Architekturen mit den Methoden aus [ACE08] ausgenutzt.

Literaturverzeichnis

[ACE08] Parallelization using polyhedral analysis - CoSy white paper, March 2008.

[Adv05] Advanced Micro Devices, Inc. *Software Optimization Guide for AMD64 Processors*, Sep. 2005.

[Adv07] Advanced Micro Devices. *AMD64 Technology, 128-Bit SSE5 Instruction Set*, Aug. 2007.

[AMD02] AMD. *AMD 64-bit technology, AMD x86-64 architecture programmer's manual*, September 2002.

[BDT06] BDTI. Implementing simd in software. INSIDE DSP, June 2006.

[BKM+05] Peter Benkart, Alexander Kaiser, Andreas Munding, Markus Bschorr, Hans-Joerg Pfleiderer, Erhard Kohn, Arne Heittmann, Holger Huebner, and Ulrich Ramacher. 3d chip stack technology using through-chip interconnects. *IEEE Des. Test*, 22(6):512–518, 2005.

[BSK+99] M. Berekovic, H. J. Stolberg, M. B. Kulaczewski, P. Pirsch, H. Müller, H. Runge, J. Kneip, and B. Stabernack. Instruction set extensions for MPEG-4 video. *Journal of VLSI Signal Processing Systems*, 23(1):27–50, October 1999.

[CDWD01] Francky Catthoor, Koen Danckaert, Sven Wuytack, and Nikil D. Dutt. Code transformations for data transfer and storage exploration preprocessing in multimedia processors. *IEEE Des. Test*, 18(3):70–82, 2001.

[CEV01] J. Corbal, R. Espasa, and M. Valero. On the efficiency of reductions in -SIMD media extensions. In *International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, Barcelona, Spain, September 2001.

[CWD+98] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology – Exploration of Memory Organisation for Embedded Multimedia System Design.* Kluwer Acad. Publ., Boston, 1998.

[DB94] J. Degener and C. Bormann. *GSM 06.10 13 kbit/s RPE/LTP speech compression.* Technische Universität Berlin, http://kbs.cs.tu-berlin.de/ jutta/toast.html, 1994. Implementierung in der Programmiersprache C.

[DCDM00] K. Danckaert, F. Catthoor, and H. De Man. A loop transformation approach for combined parallelization and data transfer and storage optimization. In *Proc. ACM Conf. on Par. and Dist. Proc. Techniques and Applications (PDPTA 00)*, pages 2591–2597, Las Vegas, USA, June 2000.

[DKR92] A. Darte, L. Khachiyan, and Y. Robert. Linear scheduling is close to optimality. *Proceedings of Int. Conf. on Application Specific Array Processors*, pages 37–46, 1992.

[Dun90] Ralph Duncan. A survey of parallel computer architectures. *IEEE Computer*, pages 5–16, February 1990.

[Dut04] Hritam Dutta. Mapping of Hierarchically Partitioned Regular Algorithms onto Processor Arrays. Master's thesis, University of Erlangen-Nürnberg, October 2004.

[Eck01] Uwe Eckhardt. *Algorithmus-Architektur-Codesign für den Entwurf digitaler Systeme mit eingebettetem Prozessorarray und Speicherhierarchie.* Dissertation, Technische Universität Dresden, May 2001.

[EM99] U. Eckhardt and R. Merker. Hierarchical algorithm partitioning at system level for an improved utilization of memory structures. *IEEE Transactions on CAD*, 18(1):14–24, January 1999.

[EOO+05] A. Eichenberger, Ka. O'Brien, Ke. O'Brien, P. Wu, T. Chen, P. Oden, D. Prener, J. Shepherd, B. So, Z. Sura, A. Wang, T. Zhang, P. Zhao, and M. Gschwind. Optimizing compiler for the CELL processor. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT '05)*, pages 161–172, St. Louis, MO, USA, September 2005. IEEE Computer Society.

[EOO+06] A. E. Eichenberger, J. K. O'Brien, K. M. O'Brien, P. Wu, T. Chen, P. H. Oden, D. A. Prener, J. C. Shepherd, B. So, Z. Sura, A. Wang, T. Zhang, P. Zhao, M. K. Gschwind, R. Archambault, Y. Gao, and R. Koo. Using advanced compiler technology to exploit the performance of the cell broadband enginetm architecture. *IBM Syst. J.*, 45(1):59–84, 2006.

[EWO04] Alexandre E. Eichenberger, Peng Wu, and Kevin O'Brien. Vectorization for SIMD architectures with alignment constraints. In *Proc. of the Conf. on Programming Language Design and Implementation (PLDI '04)*, pages 82–93, New York, NY, USA, June 2004.

[FD00] R. J. Fisher and H. G. Dietz. The SCC compiler: SWARing at MMX and 3DNow! In *Languages and Compilers for Parallel Computing LCPC 2000*. Springer-Verlag, New York, August 2000.

[Fea88] Paul Feautrier. Array expansion. In *Proceedings of the 2nd International Conference on Supercomputing (ICS '88)*, pages 429–441, New York, NY, USA, 1988. ACM.

[Fim02] Dirk Fimmel. *Optimaler Entwurf paralleler Rechenfelder unter Verwendung ganzzahliger linearer Optimierung*. Dissertation, Technische Universität Dresden, April 2002.

[Fly66] Michael J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, December 1966.

[FM99] Dirk Fimmel and Renate Merker. Localization of data transfer in processor arrays. In P. Amestoy, P. Berger, and et al (eds.), editors, *Euro-Par '99 Parallel Processing*, Lecture Notes in Computer Science, No. 1685, pages 401–408, Toulouse, France, September 1999. Springer-Verlag.

[FN91] Alfred Fettweis and Gunnar Nitsche. Numerical integration of partial differential equations using principles of multidimensional wave digital filters. *Journal VLSI Signal Processing Systems*, 3(1-2):7–24, 1991.

[Fre06] Freescale Semiconductor. *AltiVec Technology Programming Environments Manual*, 04 2006.

[Fri00] Jose Fridman. Sub-word parallelism in digital signal processing. *IEEE Signal Processing Magazine*, 2000.

[GCC] Gnu GCC home page. http://gcc.gnu.org.

[GMI+05] John Glossner, Mayan Moudgill, Daniel Iancu, Gary Nacer, Sanjay Jintukar, Stuart Standlay, Michael Samori, Tanuj Raja, and Michael Schulte. The sandbridge sandblaster convergance platform, 2005.

[GNU08] GNU Project, Free Software Foundation. *GNU Linear Programming Kit, Reference Manual, Version 4.31*, Sept. 2008.

[HDT09] Frank Hannig, Hritam Dutta, and J"urgen Teich. Parallelization approaches for hardware accelerators — loop unrolling versus loop partitioning. In *ARCS '09: Proceedings of the 22nd International Conference on Architecture of Computing Systems*, pages 16–27, Berlin, Heidelberg, 2009. Springer-Verlag.

[Hel00] D. Heller. Rabbit, a performance counters library for Intel/AMD processors and linux. Technical report, Scalable Computing Laboratory, Ames Laboratory, U.S. D.O.E., Iowa State University, http://www.scl.ameslab.gov/Projects/Rabbit/index.html, October 2000.

[HHF03] Michael Hosemann, Rene Habendorf, and Gerhard Fettweis. Hardware–software codesign of a 14.4Mbit - 64 state - viterbi decoder for an application-specific digital signal processor. In *IEEE Workshop on Signal Processing Systems, Design and Implementation (SIPS 2003)*, pages 45–50, Seoul, South Korea, August 2003.

[HHF04] Michael Hosemann, Rene Habendorf, and Gerhard Fettweis. Implementing a receiver for terrestrial digital video broadcasting in software on an application-specific dsp. In *IEEE Workshop on Signal Processing Systems, Design and Implementation (SIPS 2004)*, pages 53–58, Austin Texas, USA, October 2004.

[HM03] H. Hunter and J. Moreno. A new look at exploiting data parallelism in embedded systems. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 159–169. ACM Press, 2003.

[HT04] Frank Hannig and Jürgen Teich. Resource constrained and speculative scheduling of an algorithm class with run-time dependent conditionals. In *Proceedings of the 15th IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP 2004)*, pages 17–27, Galveston, TX, U.S.A., September 2004.

[IBM] IBM XL C and C++ compilers. http://www-01.ibm.com/software/awdtools/xlcpp/.

[ILO07] ILOG Inc. *CPLEX 11.0 User's Manual*, Sept. 2007.

[Ins01] Texas Instruments. *TMS320C64x Technical Overview*. Texas Instruments, Jan. 2001.

[Int04] Intel Corporation. *IA-32 Intel architecture software developer s manual volume 1: basic architecture*, 2004.

[Int07a] Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual*, May 2007.

[Int07b] International Business Machines (IBM) Corporation. *Cell Broadband Engine Programming Handbook*, April 2007.

[KBW03] J. Kim, S. Bunchua, and D. S. Wills. Fast color image processing using quantized color instruction set. In *International Conference on Information Technology: Computers and Communications*, Las Vegas, Nevada, April 2003.

[KHK+06a] A. Kupriyanov, F. Hannig, D. Kissler, J. Teich, R. Schaffer, and R. Merker. An architecture description language for massively parallel processor architectures. In *Proceedings 9th ITG/GMM/GI Workshop, Methoden und*

Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Dresden, Germany, February 2006.

[KHK+06b] Alexey Kupriyanov, Frank Hannig, Dmitrij Kissler, Rainer Schaffer, and Jürgen Teich. MAML – an architecture description language for modeling and simulation of processor array architectures, part I. Technical Report 03-2006, University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design, Am Weichselgarten 3, 91058 Erlangen, Germany, March 2006.

[KHKT06] Dmitrij Kissler, Frank Hannig, Alexey Kupriyanov, and Jürgen Teich. A highly parameterizable parallel processor array architecture. In *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*, pages 105–112, Bangkok, Thailand, December 2006. IEEE.

[KW03] Jongmyon Kim and D. Scott Wills. High-performance and energy-efficient heterogeneous subword parallel instructions. In *IEEE workshop on Signal processing systems SIPS 2003*, pages 75–80, Seoul, South Korea, September 2003.

[LA00] S. Larsen and S. Amarasinghe. Exploiting superword level parallelism with multimedia instruction sets. In *Proc. of Conf. on Programming Language Design and Implementation*, Vancouver, B.C., June 2000.

[Lee96] Ruby B. Lee. Subword parallelism with max-2. *IEEE Micro*, 16(4):51–59, 1996.

[Lee00] Ruby B. Lee. Subword permutation instructions for two-dimensional multimedia processing in microsimd architectures. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2000)*, pages 3–14, Boston, USA, July 2000.

[Leu00] Rainer Leupers. Code selection for media processors with SIMD instructions. In *Proceedings Design, Automation and Test in Europe Conference & Exhibition DATE '00*, March 2000.

[LRNB+07] A. Lambrechts, P. Raghavan, D. Novo Bruno, E. Rey Ramos, M. Jayapala, J. Absar, F. Catthoor, and D. Verkest. Enabling word-width aware energy optimizations for embedded processors. In *Proc. Intl. Wsh. on Optimisations for DSP and Embedded Systems (ODES)*, pages 66–75, San Jose CA, March 2007.

[MCCS02] Daniel Menard, Daniel Chillet, François Charot, and Olivier Sentieys. Automatic floating-point to fixed-point conversion for DSP code generation. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '02)*, pages 270–276, New York, NY, USA, 2002. ACM.

[MFM04] Jan Müller, Dirk Fimmel, and Renate Merker. Exploitation of instruction-level parallelism for optimal loop scheduling. In *Proceedings 8th IEEE Annual Workshop on Interaction between Compilers and Architectures (INTERACT)*, pages 13–21, Madrid, Spain, February 2004.

[MGQS03] Daniel Menard, Michel Guitton, Philippe Quemerais, and Olivier Sentieys. Efficient implementation of a rake receiver on the TMS320C64x. In *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 2161–2165, Monterey, USA, Nov. 2003.

[MKC00] R. Manniesing, I. Karkowski, and H. Corporaal. Automatic SIMD parallelization of embedded applications based on pattern recognition. In *Euro-Par 2000*, pages 349–356, Munich, August 2000.

[ML01] J. P. McGregor and Ruby B. Lee. Architectural enhancements for fast subword permutations with repetitions in cryptographic applications. In *Proceedings of ICCD 2001, International Conference on Computer Design*, pages 453–461, Austin, USA, September 2001.

[NBB+08] F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. Van der Perre, and F. Catthoor. A unified instruction set programmable architecture for multi-standard advanced forward error correction. In *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 31–36, Washington DC, Oct. 2008.

[NRZ06] Dorit Nuzman, Ira Rosen, and Ayal Zaks. Auto-vectorization of interleaved data for SIMD. In *PLDI '06: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, pages 132–143, New York, NY, USA, 06 2006. ACM.

[NZ08] Dorit Nuzman and Ayal Zaks. Outer-loop vectorization: revisited for short SIMD architectures. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 2–11, New York, NY, USA, 10 2008. ACM.

[OdBGB+03] P. Op de Beeck, C. Ghez, E. Brockmeyer, M. Miranda, F. Catthoor, and G. Deconinck. Background data organisation for the low-power implementation in real-time of a digital audio broadcast receiver on a SIMD processor. In *Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, Munich, Germany, March 2003.

[OdBMCD03] P. Op de Beeck, M. Miranda, F. Catthoor, and G. Deconinck. Background data format organisation for efficent sub-word parallel program generation. In *Workshop on Comilers and Operation Systems for Low Power (COLP'03)*, New Orleans, USA, September 2003.

[PW96] A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Micro*, 16(4):42–50, August 1996.

[Ram06] M. Ramanathan. Extending the world's most popular processor architecture. Technical report, Intel Corporation, Sept. 2006.

[Rao85] S.K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays*. PhD thesis, Stanford University, 1985.

[RCDM94] J. Rosseel, F. Catthoor, and H. De Man. An optimisation methodology for array mapping of affine recurrence equations in video and image processing applications. In *Proceedings Conference on Application-Specific Array Processing*, August 1994.

[RMK92] S. V. Rajopadhye, L. Mui, and S. Kiaei. Piecewise linear schedules for recurrence equations. *VLSI Signal Processing*, V:375–384, Oct 1992.

[RTRK89] V. Roychowdhury, L. Thiele, S.K. Rao, and T. Kailath. On the localisation of algorithms for VLSI processor arrays. *VLSI Signal Processing*, III:459–470, 1989.

[Sad08] Shyam Sadasivan. An introduction to the arm cortex-m3 processor, October 2008.

[Sch86] Alexander Schrijver. *Theory of Integer and Linear Programming*. John Wiley & Sons, New York, 1986.

[SCM00] Rainer Schaffer, Francky Catthoor, and Renate Merker. Combining background memory management and regular array co-partitioning, illustrated on a full motion estimation kernel. In *Proceedings International Conference on VLSI Design*, pages 104–109, Calcutta, India, January 2000. IEEE Computer Society.

[SDT01] J. Sebot and N. Drach-Temam. Memory bandwidth: The true bottleneck of SIMD multimedia performance on a superscalar processor. In *Lecture Notes in Computer Science*, pages 439–447. Springer-Verlag, Manchester, United Kingdom, August 2001.

[SHC05] Jaewook Shin, Mary Hall, and Jacqueline Chame. Superword-level parallelism in the presence of control flow. In *CGO '05: Proceedings of the international symposium on Code generation and optimization*, pages 165–175, Washington, DC, USA, 03 2005. IEEE Computer Society.

[Sie03] Sebastian Siegel. Untersuchung von modifizierten Abläufen für den Entwurf von Prozessorarrays. Diplomarbeit, Technische Universität Dresden, Institut für Grundlagen der Elektrotechnik und Elektronik, 2003.

[Sie08] Sebastian Siegel. *Abbildungsverfahren zur effizienten Implementierung rechenintensiever Algorithmen auf Prozessorarrays*. PhD thesis, Technische Universität Dresden, Februar 2008.

[Sil] Silicon Hive home page. http://www.siliconhive.com.

[SKD02] Todor Stefanov, Bart Kienhuis, and Ed Deprettere. Algorithmic transformation techniques for efficient exploration of alternative application instances. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 7–12, New York, NY, USA, 2002. ACM.

[SM04] Sebastian Siegel and Renate Merker. Optimized data-reuse in processor arrays. In *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 2004)*, pages 315–325, Galveston, Texas, USA, September 2004.

[SM06] Sebastian Siegel and Renate Merker. Efficient realization of data dependencies in algorithm partitioning under resource constraints. In W. E. Nagel et al., editor, *European Conference on Parallel Computing (Euro-Par 2006)*, volume 4128 of *Lecture Notes in Computer Science (LNCS)*, pages 1181–1191, Dresden, Germany, August 2006. Springer.

[SMC02a] Rainer Schaffer, Renate Merker, and Francky Catthoor. Exploitation of subword parallelism on the example of the STAF algorithm. In *Proceedings Workshop on System Design Automation (SDA 2002)*, pages 111–118, Pirna-Zehista, Germany, April 2002.

[SMC02b] Rainer Schaffer, Renate Merker, and Francky Catthoor. Systematic design of programs with sub-word parallelism. In *International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, pages 393–298, Warsaw, Poland, September 2002.

[SMC03] Rainer Schaffer, Renate Merker, and Francky Catthoor. Causality constraints for processor architectures with sub-word parallelism. In *Euromicro Symposium on Digital System Design (DSD 2003)*, pages 82–89, September 2003.

[SMHT08] Rainer Schaffer, Renate Merker, Frank Hannig, and Jürgen Teich. Utilization of all levels of parallelism in a processor array with subword parallelism. In *Proceedings of the 11th Euromicro Conference on Digital System Design (DSD)*, Parma, Italy, September 2008.

[Tex06] Texas Instruments. *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide*, August 2006.

[Thi89] Lothar Thiele. On the design of piecewise regular processor arrays. In *Proc. IEEE Symp. on Circuits and Systems*, pages 2239–2242, Portland, 1989.

[Thi92] L. Thiele. Compiler techniques for massive parallel architectures. In P. Dewilde and J. Vandewalle, editors, *Computer Systems and Software Engenieering*, pages 101–149. Kluwer Academic Publishers, 1992.

[TOdBM+05] C. Tenllado, P. Op de Beeck, M. Miranda, G. Deconinck, Catthoor F., and M. Prieto. Background data management in subword parallelizing compilers: A test case. In *International Workshop on Optimizations for DSP and Embedded Systems (ODES)*, San Jose, CA, March 2005.

[TPP+05] C. Tenllado, L. Pinuel, M. Prieto, F. Tirado, and F. Catthoor. Improving superword level parallelism support in modern compilers. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2005*, pages 303–308, Jersey City, NJ, USA, September 2005.

[TT93] J. Teich and L. Thiele. Partitioning of processor arrays: A piecewise regular approach. *INTEGRATION: The VLSI Journal*, 14(3):297–332, 1993.

[TTZ97] Jürgen Teich, Lothar Thiele, and Lee Z. Zhang. Partitioning processor arrays under resource constraints. *Journal of VLSI Signal Processing Systems*, 15, 1997.

[VEP99] J. T. J. Van Eijndhoven and E. J. D. Pol. Trimedia cpu64 architecture. In *Proceedings of the 1999 IEEE International Conference on Computer Design (ICCD '99)*, page 586, Washington, DC, USA, 1999. IEEE Computer Society.

[VJB05] Peter Vanbroekhoven, Gerda Janssens, and Maurice Bruynooghe. Transformation to dynamic single assignment using a simple data flow analysis. In *In Proceedings of The Third Asian Symposium on Programming Languages and Systems*, pages 330–346. Springer Verlag, 2005.

[WEW05] Peng Wu, Alexandre E. Eichenberger, and Amy Wang. Efficient SIMD code generation for runtime alignment and length conversion. In *Proceedings of the international symposium on code generation and optimization CGO '05*, pages 153–164, Washington, DC, USA, March 2005. IEEE Computer Society.

[WEWZ05] Peng Wu, Alexandre E. Eichenberger, Amy Wang, and Peng Zhao. An integrated simdization framework using virtual vectors. In *Proceedings of the 19th Annual International Conference on Supercomputing*, pages 169–178, Cambridge, Massachusetts, June 2005.

[Wil93] Doran K. Wilde. A library for doing polyhedral operations. Technical report, IRISA, December 1993.

[WL98] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *Journal of Parallel Computing*, 24(3–4):445–476, May 1998. Special Double Issue: Languages and Compilers for Parallel Computers.

Anhang A

Algorithmen

In diesem Abschnitt werden ausgewählte Algorithmen in Form eines uniforme iterativen Algorithmus vorgestellt, welche in der Arbeit verwendet werden.

A.1 IIR-Filter

Algorithmus A.1 zeigt das Filter mit unendlicher Impulsantwort (engl. infinite impulse response filter, kurz: IIR-Filter).

In Abbildung A.1 ist der reduzierte Abhängigkeitsgraph des beschriebenen IIR-Filters dargestellt.

Algorithmus A.1 IIR-Filter als uniform iterativer Algorithmus

$$
\begin{aligned}
s_1: \quad & y_a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} a[j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{1,1} \\ y_a\left[\begin{smallmatrix} i-1 \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{1,2} \end{cases} \\
s_2: \quad & y_x\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} x[i-j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{2,1} \\ x[i-j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{2,2} \\ y_x\left[\begin{smallmatrix} i-1 \\ j-1 \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{2,3} \end{cases} \\
s_3: \quad & y_{s_x}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = y_a\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \cdot y_x\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{3,1} \\
s_4: \quad & y_{y_x}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} y_{s_x}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{4,1} \\ y_{y_x}\left[\begin{smallmatrix} i \\ j+1 \end{smallmatrix}\right] + y_{s_x}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{4,2} \end{cases} \\
s_5: \quad & y_b\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} b[j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{5,1} \\ y_b\left[\begin{smallmatrix} i-1 \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{5,2} \end{cases} \\
s_6: \quad & y_{y_t}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} y[i-j], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{6,1} \\ y_y\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{6,2} \\ y_{y_t}\left[\begin{smallmatrix} i-1 \\ j-1 \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{6,3} \end{cases} \\
s_7: \quad & y_{s_y}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = y_b\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \cdot y_{y_t}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{7,1} \\
s_8: \quad & y_{y_y}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = \begin{cases} y_{s_y}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{8,1} \\ y_{y_y}\left[\begin{smallmatrix} i \\ j+1 \end{smallmatrix}\right] + y_{s_y}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{8,2} \end{cases} \\
s_9: \quad & y_y\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] = y_{y_x}\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] + y_{y_y}\left[\begin{smallmatrix} i \\ j+1 \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_9 \\
s_{10}: \quad & y[i] = y_y\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right] \in \mathcal{I}_{10}
\end{aligned}
$$

mit
$\mathcal{I}_{1,1} = \mathcal{I}_{2,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = 0 \wedge 0 \le j \le J_x - 1\}$
$\mathcal{I}_{1,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le I - 1 \wedge 0 \le j \le J_x - 1\}$
$\mathcal{I}_{2,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le I - 1 \wedge j = 1\}$
$\mathcal{I}_{2,3} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le I - 1 \wedge 1 \le j \le J_x - 1\}$
$\mathcal{I}_{3,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge 0 \le j \le J_x - 1\}$
$\mathcal{I}_{4,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge j = J_x - 1\}$
$\mathcal{I}_{4,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge 0 \le j \le J_x - 2\}$
$\mathcal{I}_{5,1} = \mathcal{I}_{6,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = 0 \wedge 1 \le j \le J_y - 1\}$
$\mathcal{I}_{5,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le I - 1 \wedge 1 \le j \le J_y - 1\}$
$\mathcal{I}_{6,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge j = 0\}$
$\mathcal{I}_{6,3} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le I - 1 \wedge 1 \le j \le J_y - 1\}$
$\mathcal{I}_{7,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge 1 \le j \le J_y - 1\}$
$\mathcal{I}_{8,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge j = J_y - 1\}$
$\mathcal{I}_{8,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge 1 \le j \le J_y - 2\}$
$\mathcal{I}_9 = \mathcal{I}_{10} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le I - 1 \wedge j = 0\}$

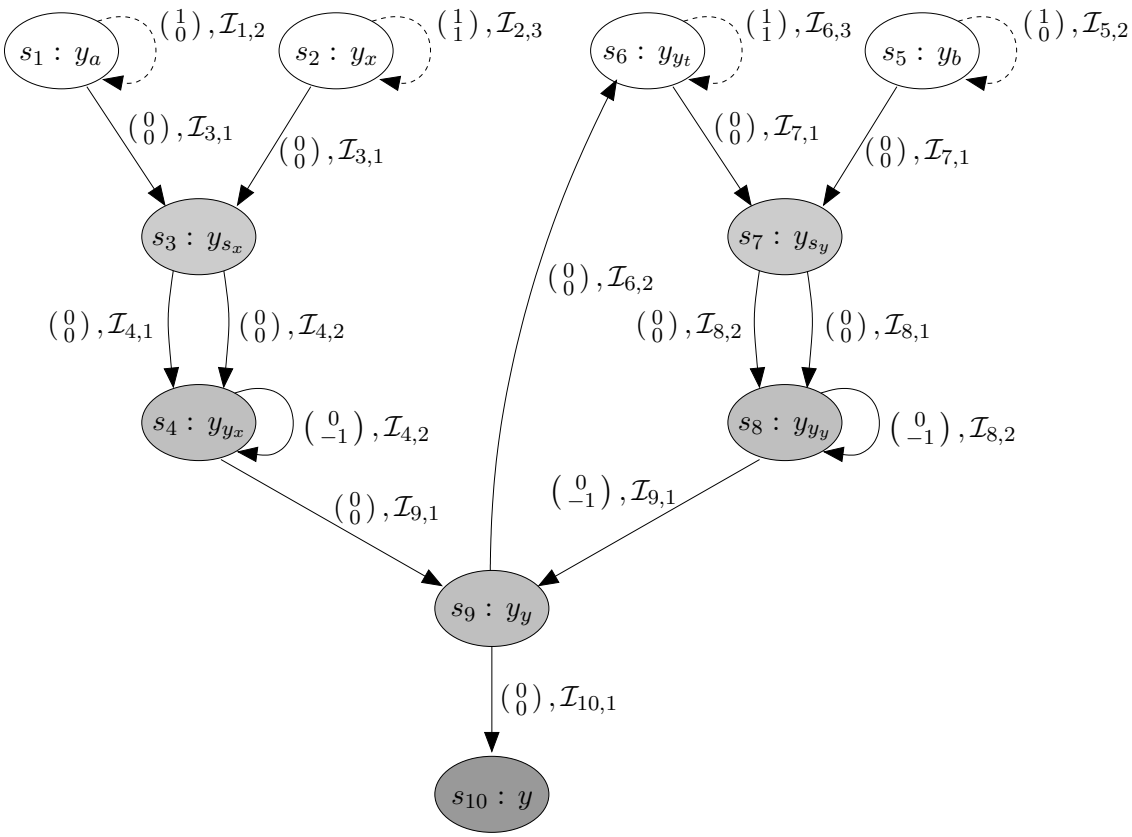


Abbildung A.1: Reduzierter Abhängigkeitsgraph des IIR-Filters

A.2 STAF-Algorithmus

Der kurzfristige Analysefilteralgorithmus (englisch: short term analysis filtering (STAF) algorithm) ist Teil des GSM Standards. Die uniforme Rekurrenzgleichnung des STAF-Algorithmus ist in Algorithmus A.2 angegeben.

Algorithmus A.2 STAF-Algorithmus

$$
\begin{aligned}
s_1: \quad & y_r\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] = \begin{cases} r[i], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{1,1} \\ y_r\left[\begin{smallmatrix} i \\ k-1 \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{1,2} \end{cases} \\
s_2: \quad & y_d\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] = \begin{cases} d_e[k], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{2,1} \\ y_{c_d}\left[\begin{smallmatrix} i-1 \\ k \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{2,2} \end{cases} \\
s_3: \quad & y_u\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] = \begin{cases} u_e[k-i], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{3,1} \\ u_e[k-i], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{3,2} \\ y_{c_u}\left[\begin{smallmatrix} i-1 \\ k-1 \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{3,3} \end{cases} \\
s_4: \quad & y_{c_d}\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] = y_d\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] + y_r\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \cdot y_u\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right], \quad \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{4,1} \\
s_5: \quad & y_{c_u}\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] = y_u\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] + y_r\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \cdot y_d\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right], \quad \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{5,1} \\
s_6: \quad & d_a[k] = y_d\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right], \quad \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{6,1} \\
s_7: \quad & u_a[k-i] = \begin{cases} y_u\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{7,1} \\ y_u\left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right], & \left[\begin{smallmatrix} i \\ k \end{smallmatrix}\right] \in \mathcal{I}_{7,2} \end{cases}
\end{aligned}
$$

mit $\mathcal{I}_{1,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le i_n - 1 \wedge k = 0\}$,
$\mathcal{I}_{1,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le i_n - 1 \wedge 1 \le k \le k_n - 1\}$,
$\mathcal{I}_{2,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = 0 \wedge 0 \le k \le k_n - 1\}$,
$\mathcal{I}_{2,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le i_n - 1 \wedge 0 \le k \le k_n - 1\}$,
$\mathcal{I}_{3,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = 0 \wedge 0 \le k \le k_n - 1\}$,
$\mathcal{I}_{3,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le i_n - 1 \wedge k = 0\}$,
$\mathcal{I}_{3,3} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le i_n - 1 \wedge 1 \le k \le k_n - 1\}$,
$\mathcal{I}_{4,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le i_n - 1 \wedge 0 \le k \le k_n - 1\}$,
$\mathcal{I}_{5,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 0 \le i \le i_n - 1 \wedge 0 \le k \le k_n - 1\}$,
$\mathcal{I}_{6,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = i_n - 1 \wedge 0 \le k \le k_n - 1\}$,
$\mathcal{I}_{7,1} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid i = i_n - 1 \wedge 0 \le k \le k_n - 1\}$,
$\mathcal{I}_{7,2} = \{\mathbf{i} = \left(\begin{smallmatrix} i \\ k \end{smallmatrix}\right) \in \mathbb{Z}^2 \mid 1 \le i \le i_n - 2 \wedge k = k_n - 1\}$,
$i_n = 8$, $k_n = \{13, 14, 120\}$.

A.3 Kantenerkennungsalgorithmus

Algorithmus A.3 KEA mit Datenwiederverwendung in einer Raumrichtung

$$
\begin{array}{lll}
s_1: & y_i\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = p_i\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_1 \\
s_2: & h_1\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = y_i\left[\begin{smallmatrix}x\\y-1\end{smallmatrix}\right] + 2 \cdot y_i\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] + y_i\left[\begin{smallmatrix}x\\y+1\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_2 \\
s_3: & v_1\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = y_i\left[\begin{smallmatrix}x\\y-1\end{smallmatrix}\right] - y_i\left[\begin{smallmatrix}x\\y+1\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_3 \\
s_4: & h_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = |h_1\left[\begin{smallmatrix}x-1\\y\end{smallmatrix}\right] - h_1\left[\begin{smallmatrix}x+1\\y\end{smallmatrix}\right]|, & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_4 \\
s_5: & v_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = |v_1\left[\begin{smallmatrix}x-1\\y\end{smallmatrix}\right] + 2 \cdot v_1\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] + v_1\left[\begin{smallmatrix}x+1\\y\end{smallmatrix}\right]|, & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_5 \\
s_6: & m\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = \min(255, h_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] + v_2\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right]), & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_6 \\
s_7: & p_o\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] = m\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right], & \left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathcal{I}_7
\end{array}
$$

mit $\mathcal{I}_1 = \{\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathbb{Z}^2 \mid 0 \leq x \leq N-1 \wedge 0 \leq y \leq M-1\}$
$\mathcal{I}_2 = \mathcal{I}_3 = \{\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathbb{Z}^2 \mid 0 \leq x \leq N-1 \wedge 1 \leq y \leq M-2\}$
$\mathcal{I}_4 = \mathcal{I}_5 = \mathcal{I}_6 = \mathcal{I}_7 = \{\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] \in \mathbb{Z}^2 \mid 1 \leq x \leq N-2 \wedge 1 \leq y \leq M-2\}$

A.4 Wellendigitalfilter

Algorithmus A.4 zeigt eine Version des Wellendigitalfilters, wie er in [FN91] vorgestellt wurde.

Algorithmus A.4 Wellendigitalfilter mit elf Zuweisungen und 32 Datenabhängigkeiten

$$
\begin{aligned}
b_1 &: \quad b_1[\mathbf{i}] = a_1[\mathbf{i}] - y_1\left(a_1[\mathbf{i}] + a_6[\mathbf{i}]\right), && \mathbf{i} \in \mathcal{I} \\
b_2 &: \quad b_2[\mathbf{i}] = a_2[\mathbf{i}] - y_1\left(a_2[\mathbf{i}] + a_7[\mathbf{i}]\right), && \mathbf{i} \in \mathcal{I} \\
b_3 &: \quad b_3[\mathbf{i}] = -b_4[\mathbf{i}] - a_5[\mathbf{i}] - \left(a_3[\mathbf{i}] + a_4[\mathbf{i}] + a_5[\mathbf{i}]\right), && \mathbf{i} \in \mathcal{I} \\
a_4 &: \quad a_4[\mathbf{i}] = -0.5\left(\left(b_6[\mathbf{i} - \mathbf{d}_1] + b_4[\mathbf{i} - \mathbf{d}_1]\right) - \left(b_6[\mathbf{i} - \mathbf{d}_2] - b_4[\mathbf{i} - \mathbf{d}_2]\right)\right), && \mathbf{i} \in \mathcal{I} \\
b_4 &: \quad b_4[\mathbf{i}] = b_5[\mathbf{i}] + a_4[\mathbf{i}] - a_5[\mathbf{i}], && \mathbf{i} \in \mathcal{I} \\
a_5 &: \quad a_5[\mathbf{i}] = -0.5\left(\left(b_7[\mathbf{i} - \mathbf{d}_3] + b_5[\mathbf{i} - \mathbf{d}_3]\right) - \left(b_7[\mathbf{i} - \mathbf{d}_4] - b_5[\mathbf{i} - \mathbf{d}_4]\right)\right), && \mathbf{i} \in \mathcal{I} \\
b_5 &: \quad b_5[\mathbf{i}] = -\left(a_3[\mathbf{i}] + a_4[\mathbf{i}]\right) + y_2\left(a_3[\mathbf{i}] + a_4[\mathbf{i}] + a_5[\mathbf{i}]\right), && \mathbf{i} \in \mathcal{I} \\
a_6 &: \quad a_6[\mathbf{i}] = -0.5\left(\left(b_6[\mathbf{i} - \mathbf{d}_1] + b_4[\mathbf{i} - \mathbf{d}_1]\right) + \left(b_6[\mathbf{i} - \mathbf{d}_2] - b_4[\mathbf{i} - \mathbf{d}_2]\right)\right), && \mathbf{i} \in \mathcal{I} \\
b_6 &: \quad b_6[\mathbf{i}] = -b_1[\mathbf{i}] - \left(a_1[\mathbf{i}] + a_6[\mathbf{i}]\right), && \mathbf{i} \in \mathcal{I} \\
a_7 &: \quad a_7[\mathbf{i}] = -0.5\left(\left(b_7[\mathbf{i} - \mathbf{d}_3] + b_5[\mathbf{i} - \mathbf{d}_3]\right) + \left(\left(b_7[\mathbf{i} - \mathbf{d}_4] - b_5[\mathbf{i} - \mathbf{d}_4]\right)\right), && \mathbf{i} \in \mathcal{I} \\
b_7 &: \quad b_7[\mathbf{i}] = -b_2[\mathbf{i}] - \left(a_2[\mathbf{i}] + a_7[\mathbf{i}]\right), && \mathbf{i} \in \mathcal{I}
\end{aligned}
$$

mit $\mathbf{d}_1 = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}^t$, $\mathbf{d}_2 = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}^t$, $\mathbf{d}_3 = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}^t$, $\mathbf{d}_4 = \begin{pmatrix} 0 & -1 & 1 \end{pmatrix}^t$ und
$\mathcal{I} = \{\mathbf{i} = \begin{pmatrix} x & y & z \end{pmatrix}^t \mid 0 \leq x < 10 \wedge 0 \leq y < 10 \wedge 0 \leq x + y + 2 \cdot z < 10\}$

Anhang B

Graphen und Bäume

Graphen und Bäume, als spezielle Graphen, sind beliebte Hilfsmittel, um komplexe Zusammenhänge einfach darzustellen. Die in dieser Arbeit verwendeten Graphen, der reduzierte Abhängigkeitsgraph (Definition 3.4) und der Baum der partitionierten Abhängigkeitsvektoren (Definition 4.3) zeigen dies.

Im nachfolgenden Abschnitt wird zuerst der Graph und seine Eigenschaften vorgestellt, bevor in Abschnitt B.2 der Baum und dessen Eigenschaften präsentiert werden.

B.1 Graphen

Die Definition eines Graphen lautet:

Definition B.1 (Graph mit Mehrfachkanten)
Ein (gerichteter) Graph mit Mehrfachkanten ist ein Paar $G = (\mathcal{V}, \mathcal{E})$. Hierbei ist $\mathcal{V}$ eine endliche Menge von Knoten v. Die Menge $\mathcal{E}$ ist eine Multimenge über dem kartesischen Produkt $\mathcal{V} \times \mathcal{V}$. Sie enthält alle Kanten e des Graphens.

Die Knoten eines Graphen werden im Allgemeinen als Punkte oder Kreise dargestellt. Die Kanten sind Pfeile, welche vom Knoten $u = \sigma(e) \in \mathcal{V}$ zum Knoten $v = \delta(e) \in \mathcal{V}$ zeigen, wenn $e \in \mathcal{E}$ ist.

Von einem Knoten $u \in \mathcal{V}$ können Vorgänger und Nachfolger definiert werden.

Definition B.2 (Vorgänger, Nachfolger)
Es sei $G = (\mathcal{V}, \mathcal{E})$ ein Graph und $e \in \mathcal{E}$ eine Kante von Knoten $u = \sigma(e)$ nach Knoten $v = \delta(e)$. Dann ist v direkter Nachfolger von u und u direkter Vorgänger von v.

Die Anzahl an direkten Vorgängern und Nachfolgern werden als Eingangsgrad und Ausgangsgrad bezeichnet.

Definition B.3 (Eingangsgrad, Ausgangsgrad)
Es sei $G = (\mathcal{V}, \mathcal{E})$ ein Graph. Der Eingangsgrad $i(u)$ eines Knotens $u \in \mathcal{V}$ ist gleich der Anzahl seiner direkten Vorgänger, d. h.

$$i(u = \sigma(e)) = |(v = \delta(e)|e \in \mathcal{E})| \,. \tag{B.1}$$

Der Ausgangsgrad $o(u)$ eines Knotens $u \in \mathcal{V}$ ist gleich der Anzahl seiner direkten Nachfolger, d. h.

$$o(u = \delta(e)) = |(w = \sigma(e)|e \in \mathcal{E})| \,. \tag{B.2}$$

Ein Knoten u heißt isoliert, wenn $o(u) = i(u) = 0$ gilt.

Die Kanten $e \in \mathcal{E}$ eines Graphen $G = (\mathcal{V}, \mathcal{E})$ können zu Pfaden $\mathcal{W}$ zusammen gefasst werden.

Definition B.4 (Pfad)
Ein Pfad $\mathcal{W} = \{e_0, e_1, \ldots, e_{m-1}\}$ in einem Graphen $G = (\mathcal{V}, \mathcal{E})$ ist eine geordnete Menge von Kanten mit $m \in \mathbb{N}_0$ und $\delta(e_{i-1}) = \sigma(e_i)$ für alle $i \in \{1, ..., m-1\}$.

Die Länge des Pfades ist gegeben durch m. Ein Pfad der Länge $m = 0$ heißt leerer Pfad. Den Knoten $u_0 = \sigma(e_0) = \sigma(\mathcal{W})$ bezeichnen wir als Anfangsknoten von $\mathcal{W}$, den Knoten $v_{m-1} = \delta(e_{m-1}) = \delta(\mathcal{W})$ als Endknoten von $\mathcal{W}$. Alle anderen Knoten von $\mathcal{W}$ heißen innere Knoten.

Definition B.5 (Einfacher Pfad, Zyklus, Weg, Kreis, Schlinge)
Es sei $\mathcal{W} = \{e_0, e_1, \ldots, e_{m-1}\}$ ein Pfad in einem Graphen G. Der Pfad $\mathcal{W}$ heißt:

einfach, *wenn er keine Kante mehrfach durchläuft, d. h. wenn alle e_i paarweise verschieden sind $(i = 0, ..., m-1)$.*

Zyklus, *wenn er geschlossen ist, d. h. wenn $\delta(e_{m-1}) = \sigma(e_0)$ gilt.*

Weg $\mathcal{W}^w$, *wenn er keinen Knoten mehrfach durchläuft, d. h. wenn alle $\sigma(e_i)$ untereinander und alle $\delta(e_j)$ untereinander paarweise verschieden sind $(i, j = 0, ..., m-1)$.*

Kreis $\mathcal{W}^\circ$, *wenn er ein Weg ist und geschlossen ist, d. h. ein Zyklus ist.*

Schlinge $\mathcal{W}^s$, *wenn der Pfad $\mathcal{W}$ nur eine Kante besitzt und der Pfad gleichzeitig ein Kreis ist.*

Der leere Pfad ist einfach und ist auch ein Weg, aber ist kein Zyklus und kein Kreis.

B.2 Bäume

Der Baum ist graphentheoretisch ein spezieller Graph. Seine Definition lautet:

Definition B.6 (Baum)
Ein gerichteter Graph $T = (\mathcal{V}, \mathcal{E})$ ist ein Baum, wenn

- *er genau einen Knoten v mit Eingangsgrad $i(v) = 0$ enthält,*
- *alle anderen Knoten u den Eingangsgrad $i(u) \leq 1$ haben,*
- *er keine Zyklen enthält.*

Definition B.7 (Wurzel, Blatt, Innerer Knoten)
Der Graph $T = (\mathcal{V}, \mathcal{E})$ sei ein Baum. Der Knoten $v \in \mathcal{V}$ mit Eingangsgrad $i(v) = 0$ wird Wurzel bezeichnet. Die Knoten $u \in \mathcal{V}$ mit Ausgangsgrad $o(u) = 0$ werden Blätter genannt. Knoten, welche weder Wurzel noch Blätter sind, werden als innere Knoten bezeichnet. Für innere Knoten gilt: $i(v) \neq 0$ und $o(u) \neq 0$.

Definition B.8 (Kindknoten, Elternknoten)
Der Graph $T = (\mathcal{V}, \mathcal{E})$ sei ein Baum. Der Knoten $w \in \mathcal{V}$ ist Kindknoten des Knoten $v \in \mathcal{V}$, wenn eine Kante $e \in \mathcal{E}$ mit $\sigma(e) = v$ und $\delta(e) = w$ existiert. Der Knoten v ist bezüglich des Knoten w der Elternknoten. Kindknoten sind die direkten Nachfolger eines Knoten und der Elternknoten ist der direkte Vorgänger eines Knoten.

Definition B.9 (Tiefe)
Der Graph $T = (\mathcal{V}, \mathcal{E})$ sei ein Baum. Die Tiefe $d(v)$ eines Knotens $v \in \mathcal{V}$ ist die Länge des Weges $\mathcal{W}^w$ von der Wurzel $r \in \mathcal{V}$ nach v. Die Tiefe $d(T) = \max_{\forall v \in \mathcal{V}} d(v)$ des Baums T ist die maximale Tiefe eines Knoten dieses Baums.

Bäume werden meist von oben nach unten oder von links nach rechts gezeichnet, wobei die Wurzel oben bzw. links angeordnet wird. Die Knoten v gleicher Tiefe $d(v) = const.$ werden auf gleichen Niveau angeordnet. Die Pfeilspitzen an den Kanten werden meist weggelassen, denn die Kanten sind immer von oben nach unten bzw. von links nach rechts gerichtet.

Anhang C

Ganzzahlig Lineare Optimierung

Die ganzzahlig lineare Programmierung (kurz: GLP, engl.: integer linear programming) [Sch86] ist Optimierungverfahren mit einer linearen Zielfunktion

$$\min: \ \mathbf{c}^t\mathbf{x} \qquad \text{oder} \qquad \max: \ \mathbf{c}^t\mathbf{x}, \tag{C.1}$$

die von einer Menge von Ungleichungen

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}, \qquad \mathbf{c}, \mathbf{x} \in \mathbb{Q}^n, \ \mathbf{A} \in \mathbb{Q}^{m \times n}, \ \mathbf{b} \in \mathbb{Q}^m \tag{C.2}$$

eingeschränkt wird. Im Gegensatz zur linearen Programmierung sind bei der ganzzahlig linearen Programmierung einzelne oder alle Elemente des Vektors $\mathbf{x}$ nur im ganzzahligen Raum $x_i \in \mathbb{Z}$ definiert.

In dieser Arbeit wird die ganzzahlig lineare Programmierung für die Bestimmung der Reindizierungsparameter verwendet. Als Löser für die GLPs können zum Beispiel die Programme ILOG CPLEX [ILO07], GLPK [GNU08] oder LPSolve verwendet werden.

C.1 Linearisierung von Beschränkungen

Viele Beschränkungen einer Problems müssen für die ganzzahlig lineare Programmierung linearisiert werden, denn erst dadurch kann eine Problemstellung als GLP formuliert werden.

C.1.1 Auswahl entsprechend dem Wert einer Binärvariable

Die Variable y soll entsprechend der Binärvariable b den Wert von x_1 oder x_2 annehmen

$$y = \begin{cases} x_1, & \text{wenn} \quad b = 1, \\ x_2, & \text{wenn} \quad b = 0. \end{cases} \tag{C.3}$$

Die Linearisierung der Bedingung führt zu vier Ungleichungen:

$$x_1 \leq y + 2X_{\text{diff}}(1-b), \tag{C.4}$$
$$x_1 \geq y - 2X_{\text{diff}}(1-b), \tag{C.5}$$
$$x_2 \leq y + 2X_{\text{diff}}b, \tag{C.6}$$
$$x_2 \geq y - 2X_{\text{diff}}b, \tag{C.7}$$

Für den Wert des Faktors X_{diff} gilt: $X_{\text{diff}} = \max(x_1, x_2) - \min(x_1, x_2)$.

C.1.2 Vergleich

Die Binärvariable $b \in \{0, 1\}$ soll entsprechend des Vergleichs $x < 0$, $x > 0$, $x \leq 0$, $x \geq 0$, $x = 0$ und $x \neq 0$ „1“ oder „0“ gesetzt werden, wobei $x \in \mathbb{Z}$ eine Ganzzahl im Bereich $x_u \leq x \leq x_o$ ist.

Zuerst betrachten wir den Vergleich $x \leq 0$:

$$b = \begin{cases} 1, & \text{wenn} \quad x \leq 0, \\ 0, & \text{sonst.} \end{cases} \tag{C.8}$$

Diesen Vergleich linearisieren wir mit:

$$x - 2X(1-b) \leq 0 \tag{C.9}$$
$$x + 2Xb \geq 1, \tag{C.10}$$

wobei $X = \max(|x_u|, |x_o|)$ ist.

Soll nun ein Vergleich $x < 0$ durchgeführt werden, so muss das x des Kleiner-Gleich-Vergleichs (C.8) gleich $x + 1$ sein. Für die Linearisierung des Kleiner-Vergleichs bedeutet das:

$$x - 2X(1-b) \leq -1 \tag{C.11}$$
$$x + 2Xb \geq 0, \tag{C.12}$$

mit $X = \max(|x_u|, |x_o|)$.

Die Linearisierung des Größer-Gleich-Vergleichs und des Größer-Vergleichs erhalten wir durch Negierung der Variable x beim Kleiner-Gleich- bzw. Kleiner-Vergleich:

$$x \geq 0 \quad \Rightarrow \quad -x \leq 0, \qquad\qquad x > 0 \quad \Rightarrow \quad -x < 0. \tag{C.13}$$

Soll die Variable x auf Gleichheit oder Ungleichheit bezüglich Null überprüft werden, so führt man zuerst einen Kleiner-Gleich- (b^-) und einen Größer-Gleich-Vergleich (b^+) durch:

$$b^+ = \begin{cases} 1, & \text{wenn} \quad x \geq 0, \\ 0, & \text{sonst}, \end{cases} \qquad b^- = \begin{cases} 1, & \text{wenn} \quad x \leq 0, \\ 0, & \text{sonst.} \end{cases} \tag{C.14}$$

Da die Variable x entweder größer-gleich oder kleiner-gleich Null ist, folgt für die Ergebnisvariable b:

$$\text{bei Geichheit: } b = b^+ + b^- - 1 \qquad \text{bzw. bei Ungleichheit: } b = 2 - b^+ - b^-. \tag{C.15}$$

C.1.3 Minimum oder Maximum von zwei Werten

Soll von zwei Werten x_1 und x_2 das Minimum oder das Maximum ermittelt werden

$$y = \min(x_1, x_2) \qquad \text{oder} \qquad y = \max(x_1, x_2), \tag{C.16}$$

so vergleichen wir zuerst die beiden Werte mit einem Kleiner-Gleich- bzw. Größer-Gleich-Vergleich. Die sich ergebende Binärvariable wird anschließend zur Auswahl des x_1 bzw. x_2 Wertes genutzt.

Anhang D

Optimierung Vorverarbeitung

Die Reindizierung als Vorverarbeitungsschritt bei unserer Abbildungsstrategie wurde in Abschnitt 7.1.1 vorgestellt. Für die Ermittlung der zuweisungsabhängigen Reindizierungsvektoren $\mathbf{c}_s$ wurden ganzzahlig lineare Programme (GLP) entwickelt, welche mittels der Optimierungswerkzeuge ILOG CPLEX 11.0 [ILO07] oder GLPK 4.31 [GNU08] gelöst werden können.

In den nachfolgenden Abschnitten werden die Beschreibungen der ganzzahlig linearen Programme vorgestellt. Hierbei werden die im Anhang C vorgestellten Linearisierungsmethoden verwendet.

Ausgangspunkt der ganzzahlige linearen Programme sind die reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$, welche entsprechend (7.4) auf Seite 141 berechnet werden.

Da der reindizierte Abhängigkeitsvektor $\widetilde{\mathbf{d}}(e)$ sich aus dem Abhängigkeitsvektor $\mathbf{d}(e)$ und der *Differenz* der Reindizierungsvektoren $\mathbf{c}_{\delta(e)}$ und $\mathbf{c}_{\sigma(e)}$ bestimmt, gibt es mehrere identische Lösungen. Mit der nachfolgenden Bedingung ist nur noch eine Lösung möglich:

$$\forall j : \exists c_{s,j} = 0, \qquad s \in \mathcal{S},\ 1 \leq j \leq n, \qquad 0 \leq c_{s,j}. \tag{D.1}$$

Hierfür überprüfen wir mittels der binären Variable $b^c_{s,j}$, siehe Anhang C.1.2, ob die Variable $c_{s,j} = 0$ ist. Durch die Bedingungen $\sum_{s \in \mathcal{S}} b^c_{s,j} \geq 1$ für $j \in \{1, \ldots, n\}$ wird nun sichergestellt, dass mindestens ein $c_{s,j}$ je Raumrichtung j gleich Null ist.

D.1 Ausrichtung der Abhängigkeitsvektoren

In den Kapiteln 7.1 und 7.2.4 wurde gezeigt, dass die Abhängigkeitsvektoren $\mathbf{d}(e)$, $e \in \mathcal{E}$ je Raumrichtung j möglichst in eine Richtung weisen sollen, um einen gültigen Ablaufplan $t(\mathbf{i}^\kappa, s_t)$ zu erhalten und maximale Parallelverarbeitung zu ermöglichen.

Für die einfache Untersuchung, bei der für jede Raumrichtung j überprüft wird, ob die Elemente $\widetilde{d}_j(e)$ der reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$ größer-gleich oder kleiner-gleich Null sind, siehe Abschnitt 7.1.1, muss für alle Elemente $\widetilde{d}_j(e)$ dieser Raumrichtung

nur $\widetilde{d}_j(e) \geq 0$, (positiv ausgerichtet) oder $\widetilde{d}_j(e) \leq 0$ (positiv ausgerichtet) festgelegt werden. Ohne Angabe einer Zielfunktion kann der Optimierer nun sehr schnell ermitteln, ob eine Lösung mit dieser Vorgabe existiert oder nicht. Da die Anzahl n der Raumrichtungen meist sehr gering ist, können alle Raumrichtungen sehr schnell überprüft werden. Vorallem für große UItA hat sich herausgestellt, dass dieses Vorgehen sehr effizient gegenüber einer komplexen Maximierung der Anzahl der Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren ist. Weiterhin können die Ergebnisse der Untersuchungen, wie zum Beispiel „Ausrichtung in Raumrichtung j nur in positiver Raumrichtung möglich“, als Vorgaben spätere Optimierungen beschleunigen.

Für eine komplexe Maximierung der Anzahl der Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren, bestimmten wir zuerst über (D.2) und (D.3) die Anzahl der reindizierten Abhängigkeitsvektorelemente, die in die positive bzw. negative Richtung weisen

$$b_j^+ = \sum_{e \in \mathcal{E}} b_j^+(e), \qquad b_j^+(e) = \begin{cases} 1 & \text{wenn} \quad d_j(e) > 0, \\ 0 & \text{sonst}, \end{cases} \tag{D.2}$$

$$b_j^- = \sum_{e \in \mathcal{E}} b_j^-(e), \qquad b_j^-(e) = \begin{cases} 1 & \text{wenn} \quad d_j(e) < 0, \\ 0 & \text{sonst}. \end{cases} \tag{D.3}$$

Die Linearisierung des Größer-Vergleichs $d_j(e) > 0$ und Kleiner-Vergleichs $d_j(e) < 0$ erfolgt entsprechend der Beschreibung in Anhang C.1.2. Von den Werten b_j^+ und b_j^- ermitteln wir das Minimum je Raumrichtung:

$$b_j^{\min} = \min(b_j^+,\ b_j^-). \tag{D.4}$$

Ist $b_j^{\min} = 0$, so sind alle Datenabhängigkeiten in der Raumrichtung j gerichtet. Die Anzahl der Raumrichtungen j, in denen alle Abhängigkeitsvektorelemente $\widetilde{d}_j(e)$ in eine Richtung weisen, kann somit wie folgt ermittelt werden:

$$r = \sum_{j=1}^{n} b_j^b, \qquad b_j^b = \begin{cases} 1 & \text{wenn } b_j^{\min} = 0 \\ 0 & \text{sonst.} \end{cases} \tag{D.5}$$

Die Zielfunktion für das GLP lautet somit: max r.

Können die Abhängigkeitsvektoren nicht in alle Raumrichtungen ausgerichtet werden ($r < n$), so muss sicher gestellt werden, dass keine konträren Abhängigkeitsvektoren entstehen. Hierfür stellen wir basierend auf (7.6) auf Seite 142 folgende Bedingung auf. Für alle Datenabhängigkeitskombinationen $(e_1, e_2) \in \mathcal{C}^2_{|\mathcal{E}|}$, mit $e_1, e_2 \in \mathcal{E}$ muss gelten:

$$\left(\bigwedge_{n_1=1}^{N} \bigwedge_{n_2=1}^{N} \left(n_1 \widetilde{\mathbf{d}}(e_1) \neq -n_2 \widetilde{\mathbf{d}}(e_2) \right) \right) \vee \left(\widetilde{\mathbf{d}}(e_1) = \mathbf{0} \right) \vee \left(\widetilde{\mathbf{d}}(e_2) = \mathbf{0} \right) = 1, \tag{D.6}$$

wobei gilt $\forall (e_1, e_2) \in \mathcal{C}^2_{|\mathcal{E}|}$ und $n_1, n_2 \in \mathbb{N}$. Der erste Teil der Bedingung überprüft, ob es einen Faktor $k = -\frac{n_1}{n_2}$ gibt. Mit dem zweiten und dritten Teil wird getestet, ob der

Abhängigkeitsvektor $\widetilde{\mathbf{d}}(e_1)$ bzw. $\widetilde{\mathbf{d}}(e_2)$ ein Nullvektor ist, denn in diesem Fall existiert ein Faktor $k = \frac{n_2}{n_1}$, obwohl die Abhängigkeitsvektoren nicht konträr sind.

Die Länge der reindizierten Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$, $e \in \mathcal{E}$ muss auf $-\mathbf{N} \leq \widetilde{\mathbf{d}}(e) \leq \mathbf{N}$ beschränkt werden. Der Wert von N sollte klein gewählt werden (z. B. $N = 3$), denn die Anzahl an Nebenbedingungen für das GLP steigt mit größer werdenden N rasant. Den ersten Teil von (D.6) realisieren wir über eine ODER-Kombination, deren Ergebnis invertiert werden muss:

$$\widetilde{d}^{c}(e_1, e_2) = \begin{cases} 1 & \text{wenn } \bigvee_{n_1=1}^{N} \bigvee_{n_2=1}^{N} n_1 \widetilde{\mathbf{d}}(e_1) = -n_2 \widetilde{\mathbf{d}}(e_2), \\ 0 & \text{sonst,} \end{cases} \qquad \forall (e_1, e_2) \in \mathcal{C}^2_{|\mathcal{E}|}, \tag{D.7}$$

$$\widetilde{d^{\overline{c}}}(e_1, e_2) = 1 - \widetilde{d}^{c}(e_1, e_2), \qquad \widetilde{d^{\overline{c}}}(e_1, e_2) \in \{0, 1\}. \tag{D.8}$$

Bei dem Vergleich $n_1 \widetilde{\mathbf{d}}(e_1) = -n_2 \widetilde{\mathbf{d}}(e_2)$ müssen nur jene Faktoren n_1 und n_2 betrachtet werden, bei denen der größte gemeinsame Teiler der beiden Faktoren eins ist.

Mit der binären Variable $\widetilde{d}^{0}(e)$ wird überprüft, ob $\widetilde{\mathbf{d}}(e) = \mathbf{0}$ ist:

$$\widetilde{d}^{0}(e) = \begin{cases} 1 & \text{wenn } \sum_{j=1}^{n} |\widetilde{d}_j(e)| = 0, \\ 0 & \text{sonst,} \end{cases}, \qquad \forall e \in \mathcal{E}, \qquad \widetilde{d}^{0}(e) \in \{0, 1\}. \tag{D.9}$$

Die Bestimmung der Absolutwerte $|\widetilde{d}_j(e)|$ wird im nächsten Abschnitt vorgestellt.

Somit wird die Bedingung zur Vermeidung von konträren Abhängigkeitsvektoren nun wie folgt formuliert:

$$1 = \widetilde{d^{\overline{c}}}(e_1, e_2) \vee \widetilde{d}^{0}(e_1) \vee \widetilde{d}^{0}(e_2), \qquad \forall (e_1, e_2) \in \mathcal{C}^2_{|\mathcal{E}|}. \tag{D.10}$$

D.2 Minimierung der Packoperationen

Nach der Maximierung der Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren soll nun die Anzahl der Packoperationen aufgrund des Zerfalls der Abhängigkeitsvektoren und der Datenein- und -ausgabe minimiert werden. Die Anzahl der Packoperationen aufgrund des Zerfalls der Abhängigkeitsvektoren ist abhängig von der Verschiebung der Iterationsräume zueinander und soll mit dem GLP minimiert werden. Die Anzahl der Packoperationen aufgrund Datenein- und -ausgabe ist nur von der Raumrichtung j, in der die Teilwörter zu DvBs zusammengefasst werden, abhängig und kann entsprechend Seite 143 ff. abgeschätzt werden. Diese Werte sind Eingabeparameter für unser GLP.

Die ermittelte Anzahl r an Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren, siehe (D.5), ist eine Vorgabe für dieses GLP.

Für die Bestimmung der Packoperationen aufgrund des Zerfalls der Abhängigkeitsvektoren muss die Anzahl B der Teilwörter, die in einem DvB zusammengefasst werden sollen,

angegeben werden. Zusätzlich kann noch die Raumrichtung j vorgegeben werden, in der die Teilwörter zu DvBs zusammengefasst werden sollen.

Eine Packoperation entsteht, wenn die Menge $\mathcal{X}^2(\widetilde{\mathbf{d}}(e))$, siehe (4.31) auf Seite 46, ein Element besitzt. Das ist der Fall, wenn in der Raumrichtung j in der die Teilwörter zu DvBs zusammengefasst werden, $\widetilde{d}_j(e) \mod B \neq 0$ gilt, wie in (4.31) auf Seite 46 und Abschnitt 6.2 auf Seite 91 gezeigt wurde. Da die zusätzlichen Variablen, welche aufgrund der Linearisierung der Modolo-Operation entstehen, für die Berechnung der Manhatten-Distanz verwendet werden sollen, wenden wir die Modolo-Operation auf die Absolutwerte $\widetilde{d}^a_j(e) = |\widetilde{d}_j(e)|$ aller Elemente $0 \leq j \leq n$ der Abhängigkeitsvektoren $\widetilde{\mathbf{d}}(e)$, $e \in \mathcal{E}$ an. Die Linearisierung der Absolutwertbildung lautet:

$$2N \cdot b_j(e) + \widetilde{d}_j(e) \geq \widetilde{d}^a_j(e) \tag{D.11}$$

$$\widetilde{d}_j(e) \leq \widetilde{d}^a_j(e) \tag{D.12}$$

$$2N \cdot b_j(e) + \widetilde{d}_j(e) \leq 2 \cdot N - \widetilde{d}^a_j(e) \tag{D.13}$$

$$\widetilde{d}_j(e) \geq -\widetilde{d}^a_j(e), \tag{D.14}$$

wobei die Länge des Abhängigkeitsvektors $\widetilde{\mathbf{d}}(e)$ je Raumrichtung auf $-N \leq \widetilde{d}_j(e) \leq N$ und damit der Absolutwert $\widetilde{\mathbf{d}}^a(e)$ auf $0 \leq \widetilde{d}^a_j(e) \leq N$ beschränkt wird. Die Variable $b_j(e) \in \{0, 1\}$ hat den Wert '0', wenn $\widetilde{d}_j(e) \geq 0$ oder '1' wenn $\widetilde{d}_j(e) \leq 0$. Bei $\widetilde{d}_j(e) = 0$ kann $b_j(e)$ sowohl den Wert '0' als auch '1' annehmen.

Nun führen wir für alle Elemente $\widetilde{d}^a(e)$ eine Modolorechnung $\widetilde{d}^m_j(e) = \widetilde{d}^a_j(e) \mod B$ durch:

$$\widetilde{d}^a_j(e) = \widetilde{d}^m_j(e) + B \cdot \widetilde{d}^d_j(e), \qquad 0 \leq \widetilde{d}^m_j(e) < B, \quad \widetilde{d}^m_j(e), \widetilde{d}^d_j(e) \in \mathbb{N}. \tag{D.15}$$

Ist $\widetilde{d}^m_j(e) > 0$, so wird ein Packbefehl benötigt, wenn die Teilwörter in der Raumrichtung j gepackt werden. Diese Information wird für jede Datenabhängigkeit e und jede Raumrichtung j ermittelt und in der Variable $p^d_j(e)$ hinterlegt:

$$p^d_j(e) = \begin{cases} 0 & \text{wenn} \quad \widetilde{d}^m_j(e) = 0, \\ 1 & \text{sonst,} \end{cases} \qquad e \in \mathcal{E},\ 1 \leq j \leq n. \tag{D.16}$$

Die Variable p_j gibt nun an, wie viele Packoperationen je Raumrichtung j benötigt werden, wenn in dieser Raumrichtung die Daten in DvB gepackt werden:

$$p_j = \sum_{e \in \mathcal{E}} p^d_j(e) + p^{e/a}_j, \qquad 0 \leq p_j \leq |\mathcal{E}|, \tag{D.17}$$

wobei $p^{e/a}_j$ die Anzahl zusätzlicher Packoperationen aufgrund der Datenein- und-ausgabe bei Partitionierung in Raumrichtung j ist. Die Werte für $p^{e/a}_j$ können entsprechend der Beschreibung auf Seite 143 ff. bestimmt werden.

Da die Teilwörter nur in eine Raumrichtung gepackt werden können, wird mit r_j und

$$\sum_{j=0}^{n} r_j = 1, \qquad r_j \in \{0,1\}, \tag{D.18}$$

eine Raumrichtung ausgewählt bzw. mit

$$r_j = \begin{cases} 1, & \text{wenn ausgewählt,} \\ 0, & \text{sonst.} \end{cases} \qquad \sum_{j=1}^{n} r_j = 1, \qquad r_j \in \{0,1\} \tag{D.19}$$

eine Raumrichtung vorgegeben.

In Abschnitt 7.1.1 wurde eine Einschränkung für die Auswahl der Raumrichtung j präsentiert. Existieren in dem Algorithmus Kreise $\mathcal{W}^\circ$, deren Summe der Abhängigkeitsvektoren $\mathbf{d}^\Sigma$ ein Vektor mit einem Nicht-Null-Element ist, wobei dieses Nicht-Null-Element d_j^Σ betragsmäßig kleiner B ist und gibt es mindestens einen Knoten auf dem Kreis, der eine Berechnungszuweisung ist, so kann eine Teilwortparallelisierung in der Raumrichtung j nicht erfolgen. Diese Untersuchung ist unabhängig von den Reindizierungsvektoren und kann am Ausgangsalgorithmus erfolgen. Durch Definition von $r_j = 0$ wird sichergestellt, dass die Raumrichtung j nicht für die Teilwortparallelisierung ausgewählt wird.

Nun kann mit der nachfolgenden Zielfunktion und Gleichung die minimale Anzahl von Packoperationen bestimmt werden:

$$\min: \; p = \sum_{j=1}^{n} p_j^r, \tag{D.20}$$

$$0 \leq p_j^r \leq p_j \tag{D.21}$$

$$0 \geq p_j^r - |\mathcal{E}| r_j \tag{D.22}$$

$$p_j \leq p_j^r + |\mathcal{E}|(1 - r_j) \tag{D.23}$$

D.3 Länge der Abhängigkeitsvektoren

Ist die Anzahl der Raumrichtung, in denen die Abhängigkeitsvektoren gerichtet sind, maximal, und wurde die Anzahl an Packoperationen minimiert, so soll nun die Länge der Abhängigkeitsvektoren minimiert werden. Die bisher ermittelten Parameter werden hierbei als Beschränkungen für diese Optimierung verwendet. Ziel der Reduzierung der Länge der Abhängigkeitsvektoren ist ein geringer Datentransfer und eine baldige Datenwiederverwendung im Rechenfeld. Als Maß für die Länge wählen wir die Manhattan-Distanz m, siehe (7.18) auf Seite 148. Die Manhattan-Distanz m erhalten wir somit durch das Aufsummieren der Absolutwerte $\widetilde{d}_j^a(e)$:

$$m = \sum_{j=1}^{n} \sum_{e \in \mathcal{E}} \widetilde{d}_j^a(e). \tag{D.24}$$

Der Wertebereich für m beträgt $0 \leq m \leq nN|\mathcal{E}|$, wobei N die obere Grenze für $\widetilde{d}_j^a(e)$ ist.

Durch das Packen der Teilwörter in DvBs ist die Manhattan-Distanz nach (D.24) nicht mehr korrekt. Zerfällt der Abhängigkeitsvektor $\widetilde{\mathbf{d}}(e)$ durch das Packen der Teilwörter in zwei partitionierte Abhängigkeitsvektoren ($p_j(e) = 1$), so ergibt sich für die Manhattan-Distanz-Anteile:

$$m_j(e) = \begin{cases} 2\widetilde{d}_j^d(e) + 1, & \text{wenn} \quad r_j = 1, \\ 2\widetilde{d}_j^a(e), & \text{wenn} \quad r_j = 0. \end{cases} \tag{D.25}$$

Erhalten wir durch das Packen nur einen partitionierten Abhängigkeitsvektor aus Abhängigkeitsvektor $\widetilde{\mathbf{d}}(e)$ ($p_j(e) = 0$), so lauten die Manhattan-Distanz-Anteile:

$$m_j(e) = \begin{cases} \widetilde{d}_j^d(e), & \text{wenn} \quad r_j = 1, \\ \widetilde{d}_j^a(e), & \text{wenn} \quad r_j = 0. \end{cases} \tag{D.26}$$

Zuerst ermitteln wir nun die möglichen Manhatten-Distanz-Anteile $m_j^r(e)$ bzw. $m_j^{\overline{r}}(e)$ abhängig von von r_j:

$$\widetilde{d}_j^d(e) \leq m_j^r(e) \leq 2\widetilde{d}_j^d(e) + 1, \qquad \widetilde{d}_j^a(e) \leq m_j^{\overline{r}}(e) \leq 2\widetilde{d}_j^a(e), \tag{D.27}$$

$$\widetilde{d}_j^d(e) \geq m_j^r(e) - 3 \cdot N \cdot p_j(e), \qquad \widetilde{d}_j^a(e) \geq m_j^{\overline{r}}(e) - 3 \cdot N \cdot p_j(e), \tag{D.28}$$

$$2\widetilde{d}_j^d(e) + 1 \leq m_j^r(e) + 3 \cdot N \cdot (1 - p_j(e))\,, \quad 2\widetilde{d}_j^a(e) \leq m_j^{\overline{r}}(e) + 3 \cdot N \cdot (1 - p_j(e))\,. \tag{D.29}$$

Diese werden über alle Datenabhängigkeiten $e \in \mathcal{E}$ aufsummiert:

$$m_j^r = \sum_{e \in \mathcal{E}} m_j^r(e), \qquad m_j^{\overline{r}} = \sum_{e \in \mathcal{E}} m_j^{\overline{r}}(e). \tag{D.30}$$

Somit müssen nun drei Bedingungen formuliert werden, um die Manhattan-Distanz m_j je Raumrichtung j zu bestimmen:

$$m_j^r \leq m_j \leq m_j^{\overline{r}}, \tag{D.31}$$

$$m_j^r \geq m_j - 4N|\mathcal{E}|(1 - r_j), \tag{D.32}$$

$$m_j^{\overline{r}} \leq m_j + 4N|\mathcal{E}|r_j. \tag{D.33}$$

Die Manhattan-Distanz m ist nun die Summe aller m_j:

$$m = \sum_{j=1}^{n} m_j. \tag{D.34}$$

Die Zielfunktion unseres GLPs lautete damit: min m.

D.4 Minimierung des gemeinsamen Iterationsraums

Nach der Maximierung der Raumrichtung mit ausgerichteten Abhängigkeitsvektoren, der Minimierung der Packbefehle und der Minimierung der Länge der Abhängigkeitsvektoren soll abschließend das Volumen des gemeinsamen Iterationsraums minimiert werden. Die Ergebnisse der vorherigen Optimierungen sind hierbei wieder Beschränkungen für dieses GLP.

Durch das Verschieben der Iterationsräume $\mathcal{I}_s$ der Zuweisungen $s \in \mathcal{S}$ eines Algorithmus kann die Größe des gemeinsamen Iterationsraums $\mathcal{I}$ beeinflusst werden. Bei der Minimierung des gemeinsamen Iterationsraums $\mathcal{I} = \operatorname{conv} \bigcup_{s\in\mathcal{S}} \mathcal{I}_s$ sollen zur Vereinfachung nur die die Iterationsräume $\mathcal{I}_s$ und $\mathcal{I}$ umgebenden Hyperquader $\mathcal{I}_s^{\square}$ und $\mathcal{I}^{\square}$ betrachtet werden, denn die Ausdehnung eines Hyperquaders in jeder Raumrichtung und damit das Volumen des Hyperquarders kann leichter bestimmt werden.

Zuerst betrachten wir ein Algorithmus ohne Teilwortparallelität. Die Eckpunkte $\widetilde{\mathbf{v}} \in \widetilde{\mathcal{V}}_s^{\square}$ des reindizierten Hyperquaders $\widetilde{\mathcal{I}}_s^{\square}$ werden mit der Gleichung $\widetilde{\mathbf{v}} = \mathbf{v}\cdot\mathbf{c}_s$, $\mathbf{v} \in \mathcal{V}_s^{\square}$ bestimmt. Nun ermitteln wir für jede Raumrichtung den minimalen und maximalen Wert $\widetilde{v}_{\min,j}$ bzw. $\widetilde{v}_{\max,j}$, um daraus die Ausdehnung $\widetilde{v}_{\text{diff},j}$ zu berechnen:

$$\widetilde{\mathbf{v}}_{\min} \leq \mathbf{v} \cdot \mathbf{c}_s, \qquad \forall \mathbf{v} \in \mathcal{V}_s^{\square},\ \forall s \in \mathcal{S}, \tag{D.35}$$

$$\widetilde{\mathbf{v}}_{\max} \geq \mathbf{v} \cdot \mathbf{c}_s, \qquad \forall \mathbf{v} \in \mathcal{V}_s^{\square},\ \forall s \in \mathcal{S}, \tag{D.36}$$

$$\widetilde{\mathbf{v}}_{\text{diff}} = \widetilde{\mathbf{v}}_{\max} - \widetilde{\mathbf{v}}_{\min} + \mathbf{1}. \tag{D.37}$$

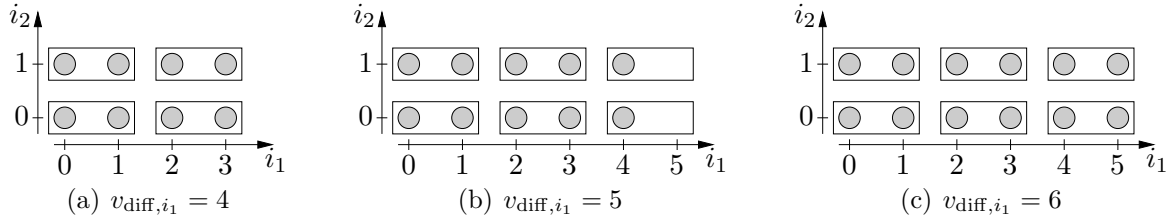
Da bei der linearen Optimierung nur lineare Zielfunktionen zulässig sind, kann das Volumen des Hyperquaders $V(\widetilde{\mathcal{I}}^{\square}) = \prod_{j=1}^{n} \widetilde{v}_{\text{diff},j}$ nicht ermittelt werden. Alternativ wurden zwei lineare Zielfunktionen implementiert. Bei der ersten Zielfunktion werden nur die Ausdehnungen des reindizieren Hyperquaders miteinander addiert:

$$\min : \sum_{j=1}^{n} \widetilde{v}_{\text{diff},j}. \tag{D.38}$$

Das hat zur Folge, dass bei einem rechteckigen Iterationsraum die Verkürzung des Iterationsraums in i_1-Richtung, das gleiche Gewicht hat, wie die in i_2-Richtung, obwohl das Volumen $V_1 = (v_{\text{diff},1} - 1) \cdot v_{\text{diff},2}$ bei $v_{\text{diff},1} < v_{\text{diff},2}$ kleiner als das Volumen $V_2 = v_{\text{diff},1} \cdot (v_{\text{diff},2} - 1)$ ist. Deshalb wichten wir bei der zweiten Zielfunktion die Ausdehnungen $\widetilde{v}_{\text{diff},j}$ des reindizierten Hyperquaders $\widetilde{\mathcal{I}}^{\square}$ mit der jeweils inversen Ausdehnung $1/v_{\text{diff},j}$ des Hyperquaders $\mathcal{I}^{\square}$

$$\min : \sum_{j=1}^{n} \frac{\widetilde{v}_{\text{diff},j}}{v_{\text{diff},j}} \qquad \text{bzw.} \qquad \min : \sum_{j=1}^{n} V(\mathcal{I}^{\square}) \frac{\widetilde{v}_{\text{diff},j}}{v_{\text{diff},j}}. \tag{D.39}$$

Die Zielfunktion kann noch mit dem Volumen $V(\mathcal{I}^{\square})$ des Hyperquaders $\mathcal{I}^{\square}$ multipliziert werden, wenn man ausschließlich ganzzahlige Ergebnisse für die Zielfunktion erhalten möchte.



(a) $v_{\text{diff},i_1} = 4$ (b) $v_{\text{diff},i_1} = 5$ (c) $v_{\text{diff},i_1} = 6$

Abbildung D.1: Minimierung der Ausdehnung bei Teilwortparallelität

Wird der Algorithmus nach der Reindizierung teilwortparallelisiert und ist die Anzahl B an Teilwörtern je DvB bekannt, so kann dies bei der Minimierung des Hyperquaders $\widetilde{\mathcal{I}}^{\square}$ beachtet werden. Man beachte, dass eine Reduzierung der Ausdehnung in der Raumrichtung, in der die Teilwörter in DvBs gepackt werden, nur dann zu einer Reduzierung der Abarbeitung führt, wenn ein ganzes DvB in dieser Raumrichtung eingespart werden kann. Abbildung D.1 zeigt das am Beispiel eines Iterationsraums mit Ausdehnung $v_{\text{diff},i_1} \in \{4, 5, 6\}$ in i_1-Richtung und mit $B = 2$ Teilwörtern je DvB. Bei einer Ausdehnung von $v_{\text{diff},i_1} = 5$ und $v_{\text{diff},i_1} = 6$ werden drei Teilwörter benötigt, um die Daten in i_1-Richtung zu packen, wobei bei $v_{\text{diff},i_1} = 5$ nicht alle DvBs voll besetzt sind. Erst die Reduzierung der Ausdehnung auf $v_{\text{diff},i_1} = 4$ führt zu einer Verringerung der Anzahl an DvBs in dieser Raumrichtung.

Die Ausdehnung des Hyperquaders $\widetilde{\mathbf{v}}^r_{\text{diff},j}$ kann bei der Teilwortparallelisierung zwei Werte je Raumrichtung annehmen.

$$\widetilde{v}^r_{\text{diff},j} = \begin{cases} \left\lceil \frac{\widetilde{v}_{\text{diff},j}}{B} \right\rceil, & \text{wenn} \quad r^j = 1, \\ \widetilde{v}_{\text{diff},j}, & \text{wenn} \quad r^j = 0. \end{cases} \tag{D.40}$$

Die Werte $\left\lceil \frac{\widetilde{v}_{\text{diff},j}}{B} \right\rceil$ für den Fall der TWP-Partitionierung bestimmen wir über die Modolo-Berechnung entsprechend (D.15) und den binären Variablen $\widetilde{v}^b_{\text{diff},j}$:

$$\left\lceil \frac{\widetilde{v}_{\text{diff},j}}{B} \right\rceil = \widetilde{v}^d_{\text{diff},j} + \widetilde{v}^b_{\text{diff},j}, \tag{D.41}$$

$$\widetilde{v}_{\text{diff},j} = \widetilde{v}^m_{\text{diff},j} + B \cdot \widetilde{v}^d_{\text{diff},j}, \qquad 0 \leq \widetilde{v}^m_{\text{diff},j} < B, \tag{D.42}$$

$$\widetilde{v}^b_{\text{diff},j} = \begin{cases} 1, & \text{wenn} \quad \widetilde{v}^m_{\text{diff},j} \neq 0, \\ 0, & \text{sonst.} \end{cases} \tag{D.43}$$

Die einfache Zielfunktion, bei der die Ausdehnungen je Raumrichtung addiert werden, lautet nun:

$$\min : \sum_{j=1}^{n} \widetilde{v}^r_{\text{diff},j}. \tag{D.44}$$

Sollen die Raumrichtungen wie in (D.39) mit der inversen Ausdehnung des Hyperquaders $\mathcal{I}^{\square}$ gewichtet werden, so kann dies nicht mehr in der Zielfunktion realisiert werden, sondern muss in (D.40) eingearbeitet werden

$$\widetilde{v}^r_{\mathrm{diff},j} = \begin{cases} \left\lfloor \frac{B}{\widetilde{v}_{\mathrm{diff},j}} \right\rfloor \left\lceil \frac{\widetilde{v}_{\mathrm{diff},j}}{B} \right\rceil, & \text{wenn} \quad r^j = 1, \\ \frac{1}{v_{\mathrm{diff},j}} \widetilde{v}_{\mathrm{diff},j}, & \text{wenn} \quad r^j = 0, \end{cases} \tag{D.45}$$

bzw.

$$\widetilde{v}^r_{\mathrm{diff},j} = \begin{cases} V(\mathcal{I}^{\square}) \left\lfloor \frac{B}{\widetilde{v}_{\mathrm{diff},j}} \right\rfloor \left\lceil \frac{\widetilde{v}_{\mathrm{diff},j}}{B} \right\rceil, & \text{wenn} \quad r^j = 1, \\ V(\mathcal{I}^{\square}) \frac{1}{v_{\mathrm{diff},j}} \widetilde{v}_{\mathrm{diff},j}, & \text{wenn} \quad r^j = 0. \end{cases} \tag{D.46}$$

Bei der zweiten Version wird jede Raumrichtung wieder mit dem Volumen von $\mathcal{I}^{\square}$ multipliziert, um ausschließlich ganzzahlige Lösungen zu erhalten. Als Zielfunktion verwenden wir weiterhin (D.44).

D.5 Beschleunigung der Optimierung

Die Ermittlung der Reindizierungsfunktionen kann für Algorithmen mit mehreren Zuweisungen und Datenabhängigkeiten sehr aufwändig sein. So wurden weitere Strategien angewendet, um die Optimierung zu beschleunigen:

- Die Maximierung der Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren wird für jede Raumrichtung j einzeln untersucht. Hierbei wird nacheinander für jede Raumrichtung untersucht, ob die Abhängigkeitsvektoren sich in dieser Raumrichtung positiv oder negativ ausrichten lassen. Ist das der Fall, so können die Abhängigkeitsvektoren in dieser Raumrichtung ausgerichtet werden. Die Information, in welcher Raumrichtung j das möglich ist und in welcher Richtung (positiv oder negativ) sich die Abhängigkeitsvektoren ausrichten lassen, werden den nachfolgenden Optimierungen als Beschränkungen mitgeteilt.

- Die Nebenbedingungen zur Vermeidung konträrer Abhängigkeitsvektoren lässt die Größe des GLP sehr stark anwachsen und führt damit zu einer deutlichen Erhöhung der Berechnungszeit. Wird bei der Maximierung der Raumrichtungen mit ausgerichteten Abhängigkeitsvektoren festgestellt, dass die Ausrichtung für alle Raumrichtungen möglich ist, so können diese Nebenbedingungen entfallen.

- Die Minimierung der Packoperationen hat sich als weiteres Problem herausgestellt. Untersuchungen haben gezeigt, dass eine Minimierung der Packoperationen ohne die Nebenbedingungen zur Vermeidung konträrer Abhängigkeitsvektoren zu einer schnellen Zwischenlösung führt. Diese Zwischenlösung wird nun als Nebenbedingung für eine Optimierung mit den Nebenbedingungen zur Vermeidung konträrer Abhängigkeitsvektoren verwendet. Die Zielfunktion dieses GLPs ist die Minimierung der Manhatten-Distanz ohne Betrachtung der Teilwortparallelität. Sollte das GLP nicht lösbar sein, so wird das GLP i. A. recht schnell abgebrochen. Durch

Erhöhung der Zwischenlösung um Eins, kann nun nach einer Lösung für das veränderte GLP gesucht werden. Die Erhöhung der Zwischenlösung wird solange fortgesetzt, bis der Optimierer eine Lösung gefunden hat.

Dieses Vorgehen hat sich als schneller erwiesen, als die Optimierung des Ausgangs-GLPs, da meist die erste Zwischenlösung die Endlösung ist.

- Der Optimierer CPLEX erlaubt die Vorgabe einer Lösung für die Optimierung. Damit kann die Suche des Optimierers nach einer Lösung deutlich verkürzt werden. In unserem Fall bietet sich diese Möglichkeit für die Optimierung der GLPs zur Minimierung der Manhatten-Distanz und zur Minimierung des Volumens an, da die Lösungen der vorherigen GLPs verwendet werden können.

- Der Optimierer CPLEX erlaubt weiterhin die Unterbrechung der Optimierung. Wurde durch den Optimierer bereits eine Lösung gefunden, so wird diese weiter verwendet. Durch einen Abbruch der Optimierung ist zwar nicht sichergestellt, dass die Lösung optimal ist, doch die Lösung ist i. A. recht gut. Die Zwischenlösungen der Optimierung werden während der Optimierung in einer Datei protokolliert.

Rainer Schaffer,

Jahrgang 1972, absolvierte von 1989 bis 1993 eine Berufsausbildung zum Industrieelektroniker. Gleichzeitig erwarb er die allgemeine Hochschulreife durch den Besuch des Abendgymnasiums Dresden. Anschließend studierte er von 1994 bis 1999 Elektrotechnik, Fachrichtung Informationstechnik an der Technischen Universität (TU) Dresden. Nach Abschluss des Studiums war Rainer Schaffer als wissenschaftlicher Mitarbeiter innerhalb mehrerer Projekte an der TU Dresden, Fakultät Elektrotechnik und Informationstechnik tätig. Während des Studiums und seiner Tätigkeit an der TU Dresden hatte er mehrere Forschungsaufenthalte am IMEC Leuven (Belgien). Im Rahmen seiner Tätigkeit an der TU Dresden entstand die Dissertation „Parallelisierung von Algorithmen zur Nutzung auf Architekturen mit Teilwortparallelität“, welche er im Frühjahr 2010 erfolgereich verteidigte.