Cone–basierte, hierarchische Modellpartitionierung zur parallelen compilergesteuerten Logiksimulation beim VLSI–Design

K. Hering, R. Haupt, Th. Villmann

Institut für Informatik
Universität Leipzig, Augustusplatz 10-11
04109 Leipzig

Universität Leipzig / Institut für Informatik
Report Nr. 13 (1995)
ftp.uni-leipzig.de – directory: /pub/ifi/reports

Zusammenfassung

Eine wichtige Form der Verifikation von komplette Prozessorstrukturen umfassenden VLSI–Entwürfen stellt die funktionelle Logiksimulation auf Gatter– und Register–Ebene dar. Im Kontext der Entwicklung eines parallelen Logiksimulationssystems auf Basis des nach dem *clock-cycle–Algorithmus* arbeitenden funktionellen Simulators *TEXSIM* (IBM) ist die der parallelen Simulation vorangehende Modellpartitionierung Gegenstand der Betrachtung. Ausgehend von einem strukturellen Hardware–Modell wird auf der Basis des Cone–Begriffs ein zweistufiger *hierarchischer* Partitionierungsansatz im Rahmen einer k–stufigen Strategie vorgestellt. Dieser Ansatz gibt Untersuchungen zur Kombination von Algorithmen Raum. Ein *Superpositionsprinzip* für Partitionen gestattet die Verschmelzung der Resultate von Partitionierungsverfahren einer Hierarchiestufe. Mit dem *Backward-Cone-Concentration–Algorithmus* (n-BCC) und dem *Minimum-Overlap-Cone-Cluster–Algorithmus* (MOCC) werden im Rahmen unseres *bottom-up* –Partitionierungsansatzes zwei neue Modellpartitionierungsverfahren eingeführt.

Abstract

The functional logic simulation on the basis of the gate– and register–level plays an important role for the verification of VLSI–designs containing whole processor structures. We consider the model partitioning of such structures which are necessary for parallel simulation on a parallel logic simulator, based on the so called *clock-cycle–algorithm*. Especially, but not required, we focus on the usage of the functional simulator *TEXSIM* (IBM). Starting from a structural hardware model we introduce a 2–level *hierarchical* partitioning strategy, embedded in a general hierarchical k–level approach, using the concept of cones. This ansatz enables us to investigate the combination of algorithms. A *superposition principle* allows to unite the results of different partitioning algorithms within one hierarchical level. In the frame of our *bottom-up* partitioning strategy we introduce two new model partitioning algorithms, the *backward-cone-concentration algorithm* (n-BCC) and the *minimum-overlap-cone-cluster algorithm* (MOCC).

Inhaltsverzeichnis

1 Vorbemerkungen

Der VLSI-Schaltkreisentwurf ist ein wichtiger Bestandteil der Entwicklung neuer Computersysteme, der in den letzten Jahren zunehmend selbst automatisiert wurde. Das ist auf die stetig wachsende Zahl von funktionellen Elementen je Prozessor zurückzuführen. Waren noch 1965 ca. 5 funktionelle Elemente je Chip zu integrieren, stieg die Zahl auf bereits 100.000 im Jahr 1985. Heute hat diese Zahl den Stand von 10^6 Elementen je Prozessor überschritten. In naher Zukunft wird dieser Wert in der Größenordnung von ca. 10^7 Elementen liegen. Bei der Entwicklung dieser Prozessoren spielt die Simulation als Verifikationsmethode in der Entwurfsphase eine entscheidende Rolle zur Fehlererkennung und -beseitigung. Der Aufwand solcher Simulationen ist jedoch sehr hoch: Die Simulationszeiten differieren gegenüber den Real-CPU-Zeiten um einen Faktor, der in der Endphase des Verifikationsprozesses bis zu 7 Größenordnungen betragen kann.

Ein Ansatz zur Reduzierung dieser extremen Simulationszeiten besteht in der Parallelisierung des Simulationsprozesses über einem Prozessormodell auf der Grundlage der modellinhärenten Parallelität. Damit ergibt sich im Vorfeld der parallelen Simulation die Notwendigkeit, das Prozessormodell in geeigneter Art und Weise zu partitionieren, um eine hohe Effizienz der parallelen Simulation zu gewährleisten[1].

Aus dem Studium und der Analyse bisheriger Forschung ergab sich, daß basierend auf der Beschreibungsebene Gatter/Register und des Parallelisierungskonzeptes (Abschnitt 2.3) der *Fan-In-Cone–Ansatz* als Partitionierungsgrundlage geeignet ist. Die Zusammenfassung von Elementen in solchen Cones stellt danach eine Zusammenfassung von Elementen der Gatter-/Register-Ebene dar (Abschnitt 3.2). Die Beziehungen zwischen den Cones lassen sich in Form eines speziellen Hypergraphen darstellen, der den Überlappungsgrad zwischen Cones formal beschreibt.

Ausgehend von einer allgemeinen *mathematischen Formulierung der Partitionierung* als eine zwischen zwei Mengen wirkende Abbildung werden die Partitionierung von Cone-Mengen charakterisierende Größen definiert, um Gütekriterien für Partitionen zu fixieren (Abschnitt 3.3). Auf Grund der zu erwartenden Prozessorgrößen mit Elementzahlen in den Größenordnungen 10^6 bis 10^7, erweist sich eine direkt optimierende Partitionierung in eine Anzahl von Clustern,

[1]Diese Arbeit wird von der Deutschen Forschungsgemeinschaft (DFG) unter dem Aktenzeichen *Sp 487/1-1* gefördert.

die im Bereich der Größenordnungen 10^1 bis 10^2 liegt, als schwierig oder nicht durchführbar. Deshalb gehen wir von einem *hierarchischen Partitionierungsmodell* aus, das schrittweise die Problemgröße reduziert (Abschnitt 4.4.1). Dieses Modell läßt sich mittels der eingeführten mathematischen Beschreibung des allgemeinen Partitionierungsproblems als Hintereinanderausführung der die Partitionierung realisierenden Abbildungen auffassen. In den unterschiedlichen Hierarchieebenen können parallel *mehrere verschiedene* Algorithmen eingesetzt und kombiniert werden (Abschnitt 4.4.2). Diese modulare Struktur im Hierarchieansatz erlaubt die Einbindung neuer Algorithmen und garantiert damit eine hohe Variabilität.

Verschiedene Algorithmen fokussieren dabei auf verschiedene mögliche Eigenschaften in der Cone-Struktur. Um die Vorteile einzelner Algorithmen bezüglich eines zu partitionierenden Modells miteinander verbinden zu können, wurde ein *Superpositionsprinzip für Partitionen* entwickelt (Abschnitt 4.4.3).

Ausgehend von ersten Untersuchungen an uns vorliegenden Prozessormodellen der Firma IBM wurden zwei neue Partitionierungsalgorithmen, der *Backward–Cone–Concentration—Algorithmus* (**n–BCC**) und der *Minimum–Overlap–Cone–Cluster—Algorithmus* (**MOCC**) entwickelt (Abschnitte 4.5.1 und 4.5.2). Erste experimentelle Ergebnisse werden in Abschnitt 4.6 vorgestellt[2].

[2]Die Implementation der Algorithmen sowie die Durchführung der Partitionierungen zur Gewinnung der experimentellen Ergebnisse wurde durch die Herren R. Reilein und H. Hennings realisiert.

2 Verteilte Logiksimulation

2.1 Ansätze zur Logiksimulation

Die Simulation ist eine wichtige Methode der Verifikation von VLSI–Entwürfen. MARWEDEL zeigt das breite Spektrum der dabei auftretenden Abstraktionsebenen, beginnend mit Simulationen von Diffusionsprozessen in kristallinen Strukturen (Prozeßebene) bis zu Simulationen auf verschiedenen Prozessoren laufender kooperierender Prozesse (Systemebene) [Mar93].

Unser Interesse ist auf die Gate- und Registerebene gerichtet. Simulation auf diesen Ebenen wird als Logiksimulation bezeichnet. Im folgenden wird ein zugrundeliegendes Modell vereinfacht als Menge von Elementen zweier Sorten (Boxen, Signale) betrachtet. Die formale Einführung eines strukturellen Hardware-Modells erfolgt in Abschnitt 3.1 . Boxen repräsentieren Elemente hochintegrierter Schaltkreise mit logischer Funktion (z.B. Gatter), mit speichernder Funktion (z.B. Latches) oder mit Kommunikationsfunktion bezüglich des Schaltkreisumfelds (z.B. Input– und Output-Pins). Sie sind mit wenigstens einem Ein- und Ausgang versehen. Signale repräsentieren die die Elemente verbindenden Leiterbahnen. Sie sind als Träger logischer Werte anzusehen. Der Einfachheit der folgenden Darstellung halber wird angenommen, daß jedes Signal eine nicht näher spezifizierte Information über seine Bindung an Boxeingänge und Boxausgänge enthält. Zu jeder Box gehört eine Abbildung (Schaltfunktion), welche Eingangstupeln logischer Werte Ausgangstupel logischer Werte zuordnet. Im Rahmen einer Simulation werden diese Abbildungen auf der Basis diskreter Zeitmodelle (bei der betrachteten Abstraktionsebene in der Regel *zero-delay–, unit-delay–, fixed delay*–Modelle [Mei93, MTSDA93]) entsprechend einem zugrundeliegenden Box–Scheduling realisiert. Diese Realisierung wird als Box–Evaluation bezeichnet. Das jeweils gewählte Zeitmodell bestimmt insbesondere die Zuordnung von als Verzögerung interpretierbaren Zeitintervallen zu Box–Evaluationen. Systemzustände während einer Simulation werden durch eindeutige Abbildungen von der Menge der speichernden Boxen und Signale in die Zweiermenge der logischen Werte gegeben. Box–Evaluationen ziehen in der Regel Zustandsänderungen nach sich. Abbildung 1 zeigt ein Basisschema der Logiksimulation in Anlehnung an [Mei93]. Ausgehend von einer Initialisierungsphase stellt der Simulationsprozeß eine zyklische

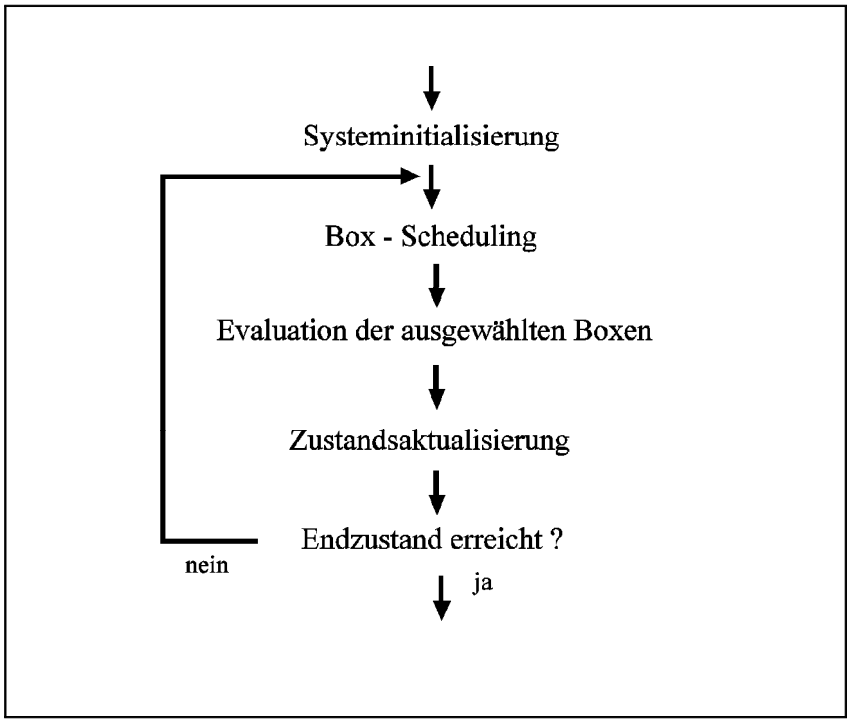


Abbildung 1: Basisschema der Logiksimulation

Abfolge von Box-Scheduling, Box-Evaluationen und Zustandsaktualisierungen dar.

Werden in jedem Simulationsschritt generell *sämtliche* Boxen zur Evaluation ausgewählt, spricht man von *zeitgesteuerter Simulation* (*time–driven, compiled mode simulation*). Dieser Simulationsart steht die *ereignisgesteuerte Simulation* (*event-driven simulation*) gegenüber [Pos89]. *Ereignisse* sind in diesem Zusammenhang als Änderungen von, den Signalen zugeordneten, logischen Werten zu verstehen. Voraussetzung für die Evaluation einer Box ist ein Ereignis bezüglich wenigstens eines ihrer Eingangssignale. Die ereignisgesteuerte Strategie bietet den Vorteil einer vergleichsweise geringeren Anzahl der während einer Simulation zu evaluierenden Boxen. Demgegenüber besteht der Vorteil der erstgenannten Strategie im Wegfall des für die Ereignisverwaltung erforderlichen Overheads.

Unsere Arbeit steht in Bezug zur rein funktionellen Logiksimulation. Für uns ist mit dem *clock–cycle*–Algorithmus ein spezieller *zeitgesteuerter* Simulationsalgorithmus von Interesse . Dabei werden in zugrundeliegenden Modellen synchroner Hardware *zero–delay*–Boxen (logische Boxen) und *unit–delay*–Boxen (durch Clock–Impuls gesteuerte Latches, globale Inputs/Outputs) unterschieden. Bei den *zero–delay*–Boxen werden aus Evaluationen resultierende Signalwertänderungen noch im gleichen Simulationsschritt

(zum gleichen Simulationszeitpunkt) wirksam; bei den *unit–delay*–Boxen geschieht das mit Verzögerung um eine Einheit der Simulationszeit. Ein Simulationsschritt entspricht der Phase zwischen zwei aufeinander folgenden Clock-Impulsen (Zyklus) und beinhaltet die Evaluation sämtlicher Boxen.

Der *clock–cycle*–Algorithmus geht von einer (aufgrund fehlender Rückkopplungen in der zugrundeliegenden kombinatorischen Logik möglichen) Einteilung der Boxen in Niveaumengen aus. Niveau 0 wird von den Latches und globale Inputs verkörpernden *unit-delay*–Boxen gebildet. Die verbleibenden Boxen werden derart aufeinander folgenden Niveaus j zugeordnet, daß die Eingangssignale jeder Box des Niveaus j nur mit Boxen aus Niveaus kleiner als j verbunden sind.

Die Simulation eines Zyklus läuft folgendermaßen ab:
Zunächst werden die an den Eingängen der Boxen des Niveaus 0 anliegenden logischen Werte auf die entsprechenden von diesen Boxen ausgehenden Signale weitergeleitet. Danach werden, in vor der Simulation fixierter Weise, alle Boxen des Niveaus 1 evaluiert und die betreffenden Ausgangssignale mit logischen Werten versehen. Entsprechend erfolgt die Evaluation der restlichen Niveaus mit wachsender Niveauzahl. Am Zyklusende liegen schließlich aktualisierte Werte an den Eingängen der Boxen des Levels 0 (für die Weiterverwendung in eventuell folgenden Simulationsschritten) und an den die globalen Ausgänge verkörpernden Boxen an.

Dieser vorgestellte Algorithmus gehört zur Klasse der '*cycle–based*' Algorithmen, die für die funktionelle Logiksimulation auf Gate- und Registerebene gegenüber ereignisgesteuerten Verfahren signifikante Laufzeitvorteile aufweisen [TU94].

2.2 Der Logik–Simulator *TEXSIM*

Der Simulator *TEXSIM* (IBM) arbeitet auf Basis des *clock-cycle* Algorithmus und ist einer der führenden heute kommerziell verfügbaren funktionellen Logiksimulatoren. In San Diego wurde er unter dem Namen *MaxSim* auf der *Design Automation Conference 94* (DAC'94) vorgestellt. *TEXSIM* ist in ein leistungsfähiges Umfeld von Design-Tools eingebettet. Die Erzeugung entsprechender Simulationsmodelle geht von VLSI–Spezifikationen in Hardwarebeschreibungssprachen *DSL/1* und *BDL/S*[3] aus, welche zunächst in strukturelle Zwischenmodelle (Protos) übersetzt werden. Zur Analyse und Bearbeitung der Protos dient eine Design Automation Data Base *DA_DB* (IBM),

[3] *DSL/1* und *BDL/S* sind firmeninterne Entwicklungen (IBM), die bezüglich ihres Funktionsumfangs in etwa mit den Sprachen *VHDL*, *Verilog* und *EDIF* vergleichbar sind.

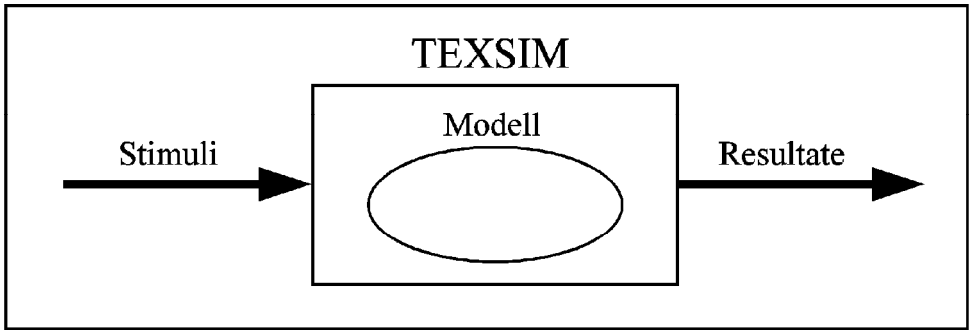


Abbildung 2: Sequentielle TEXSIM–Simulation

in deren Kontext die Bildung der eigentlichen Simulationsmodelle erfolgt. Sowohl *TEXSIM* als auch *DA_DB* stehen uns zur Verfügung.

Über ein als *TEXSIM*–Erweiterung anzusehendes Monitorprogramm sind *TEXSIM*–Simulationen mittels C– und REXX–Programmen steuerbar und Testcases in den Simulationsprozeß einbeziehbar. Die Simulationsresultate können sehr unterschiedlichen Umfang besitzen; logische Werte sind ebenso möglich wie Protokolle umfangreicher Signalwertverläufe. Eine schematische Darstellung der (momentan noch sequentiellen) *TEXSIM*–Simulation ist in Abbildung 2 wiedergegeben.

Mit *TEXSIM* zu realisierende Simulationsprozesse im Rahmen aktueller Prozessorentwicklungen lassen sich wie folgt charakterisieren :

- Die Modelle verkörpern in der Regel komplette Prozessorstrukturen.
- Die Stimuli (Testcases) stellen Mikro- bzw. Maschinencodesequenzen dar.
- Es erfolgt eine große Anzahl zeitintensiver Simulationsläufe mit umfangreichen Codesequenzen über ein und demselben Modell.

Man kann davon ausgehen, daß für relevante Prozessormodelle die Simulationslaufzeit das bis zu 10^7–fache der simulierten CPU–Zeit beträgt und bei heutigen Entwürfen den Umfang von *Monaten* erreichen kann. Damit ergibt sich hinsichtlich der Realisierbarkeit interessierender komplexer Simulationsläufe im Rahmen vertretbarer Entwicklungszeiten die Forderung einer signifikanten Reduzierung der Simulationszeit. Mit diesem Ziel arbeiten wir an der Parallelisierung der Simulation auf *TEXSIM*–Basis.

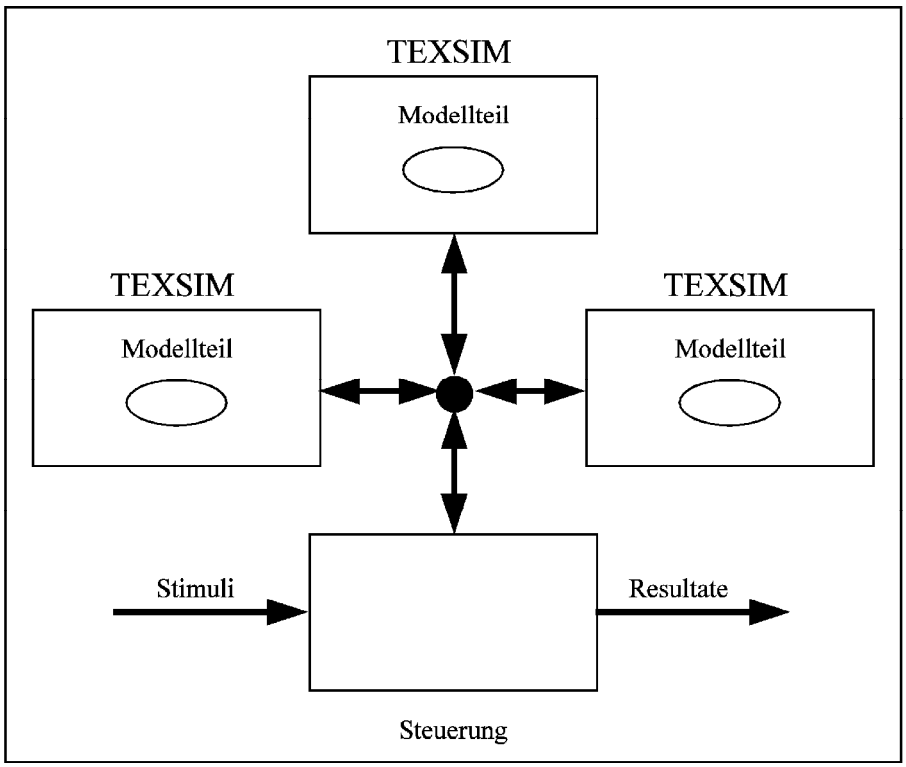


Abbildung 3: Parallele TEXSIM–Simulation

2.3 Parallelisierung der Logiksimulation

Wir gehen bei unserem Parallelisierungsansatz von der modellinhärenten Parallelität aus. Die angestrebte parallele Simulation ist durch auf lose gekoppelten Prozessoren über Teilen eines Ausgangsmodells arbeitende kooperierende *TEXSIM*–Instanzen gekennzeichnet, welche in ihrem Kern (Realisierung des *clock-cycle* Algorithmus) nicht modifiziert werden (Abb. 3) [Bry88]. Als mögliche Zielhardware sind über einen *High Performance Switch* verbundene *RISC/6000*–Prozessoren (*SP2*–Systeme von IBM) vorgesehen. Eine Ausführung auf alternativen Hardware-Plattformen, z.B. Cray T3D und Parsytec, soll grundsätzlich möglich sein.

Wir arbeiten an der Entwicklung, Analyse und Implementierung von Partitionierungsalgorithmen für Modelle im Vorfeld der parallelen Logiksimulation auf *TEXSIM*–Basis. Dabei ist die Relevanz dieser Arbeiten nicht auf den speziellen uns zur Verfügung stehenden Simulator beschränkt; die Ergebnisse sind für die Parallelisierung anderer auf Basis des *clock-cycle*–Algorithmus arbeitender Simulatoren übertragbar.

Die mit der Modellpartitionierung erfolgende Fixierung von, durch einzelne Simulatorinstanzen auf parallelen Prozessoren zu bearbeitenden, Modellteilen bestimmt einen aus dem Schneiden von Signalpfaden resultierenden Kommunikations- und Synchronisationsoverhead sowie eine Lastverteilung bezüglich der beteiligten Prozessoren. Diese Faktoren beeinflussen wesentlich den *speed-up* einer auf die Partitionierung folgenden parallelen Simulation. Ziel der Partitionierung ist damit eine möglichst optimale Gestaltung dieser Faktoren hinsichtlich erreichbarer Verkürzungen der Simulationslaufzeit. Angesichts der in der Regel großen Zahl über ein und demselben Modell erfolgenden zeitintensiven Simulationsläufe (siehe Abschnitt 2.2) kann ein im Vergleich zu einem einzelnen Simulationslauf hoher Rechenaufwand für die Partitonierung durchaus gerechtfertigt sein.

Wir gehen davon aus, daß die in eine parallele Simulation einbezogenen Simulatorinstanzen über einem Modellteil die Simulation eines Zyklus unverändert gegenüber sequentiellen Läufen über dem Gesamtmodell ausführen. Deshalb müssen innerhalb der Modellteile Boxen derart zusammengefaßt werden, daß eine Interprozessorkommunikation höchstens an den Zyklusgrenzen notwendig wird. Diese Forderung ist bei der Wahl sogenannter *fan-in–Cones* als Grundbausteine für Partitionskomponenten erfüllt. Daraus begründet sich unsere cone-basierte Partitionierungsstrategie.

3 Mathematische Beschreibung des Modells – Definitionen und Begriffsbildungen

3.1 Beschreibung des Hardware-Modells

Als Grundlage für alle folgenden Definitionen und Begriffsbildungen wird ein strukturelles Modell für logisches Design auf der Gatter- und Register-Ebene vorgestellt. Die mit diesem Modell beschreibbaren Schaltkreise sind synchron getaktet, d.h. asynchrone kombinatorische Rückkopplungen sind nicht erlaubt.

Auf dieser Grundlage läßt sich das Hardware–Modell M aus folgenden Mengen aufbauen (s. Abb. 4):

M_I ... globale Eingänge, M_O ... globale Ausgänge, M_E ... logische Elemente,

M_L ... speichernde Elemente, M_S ... Signale.

Die Elemente der Menge $M_I \cup M_O$ modellieren die Verbindungen der Hardware nach außen. Die Menge M_E faßt alle Elemente in der betrachteten Hardware zusammen, die Träger logischer Funktionen sind. Die Elemente von M_E können dabei durchaus komplexe Gebilde mit komplizierterer logischer Funktion im Rahmen der o.g. Gatter- und Register-Ebene sein. Demgegenüber stehen die Elemente von M_L, die Speicherfunktion besitzen und zyklusbegrenzend im Rahmen des in Abschnitt 2.1 erläuterten *clock-cycle*–Algorithmus wirken. Alle Elemente von M_I, M_O, M_E und M_L (s. Abb. 4) sind paarweise disjunkt und werden in der Menge der Boxen $M_B = M_I \cup M_O \cup M_E \cup M_L$ zusammengefaßt. Eine Verbindung der Elemente von M_B untereinander kann durch Signale, die in der Menge M_S zusammengefaßt sind und in der zugrundeliegenden Hardware durch Leiterbahnen repräsentiert werden, erfolgen. Die Signale als Elemente von M_S verbinden mindestens 2 Elemente aus M_B miteinander, symbolisieren also häufig komplizierte Leiterbahnstrukturen, wobei die Pfeilrichtung der Richtung des Signalflusses zur Übertragung logischer Werte entspricht (vgl. Abb. 5).

Neben der hier nicht weiterverfolgten Möglichkeit auf der Grundlage der oben eingeführten Mengen einen Hypergraphen einzuführen, bietet sich die Definition eines

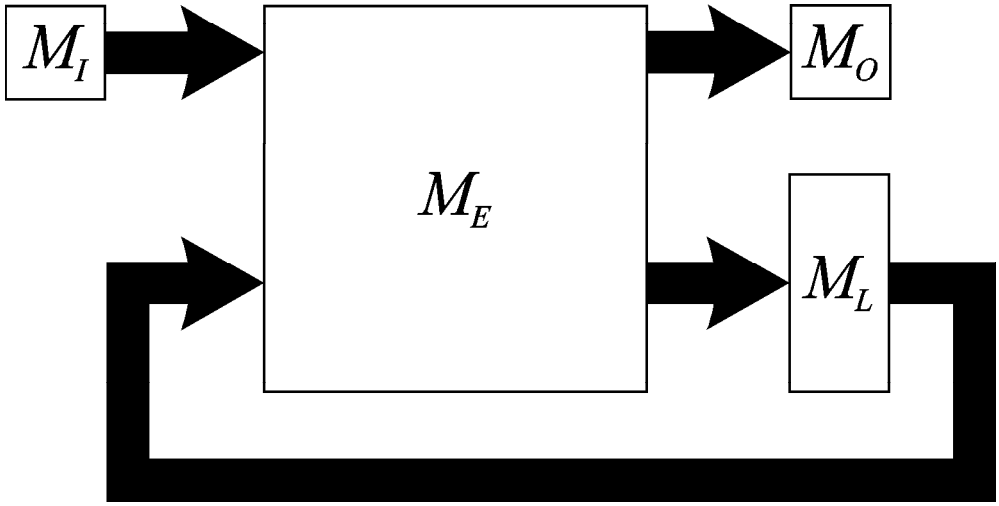


Abbildung 4: Hardware-Modell (schematisch) – Grobstruktur

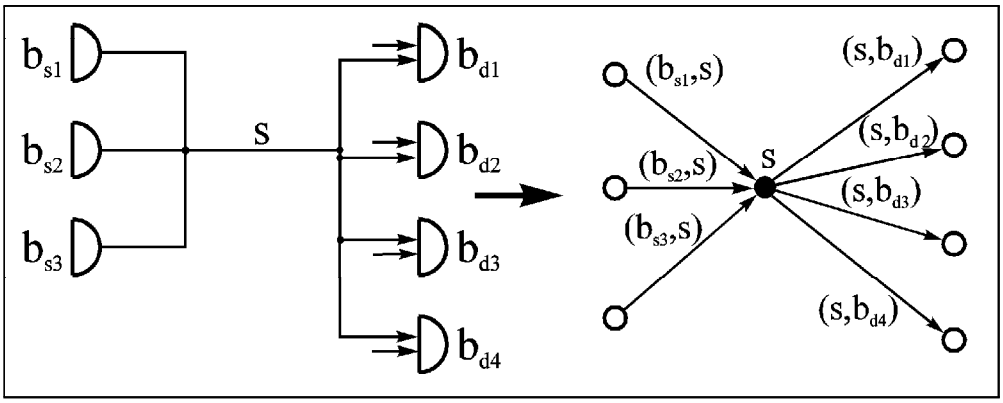


Abbildung 5: Schematische Schaltkreisdarstellung (links) und Modelldarstellung (rechts)

gerichteten bipartiten Graphen an. Dazu führen wir die Relation $M_{\mathcal{R}} \subseteq (M_B \times M_S) \cup (M_S \times M_B)$ ein, die zum Ausdruck bringt, daß sowohl eine Box $b_1 \in M_B$ logische Werte einem Signal $s_1 \in M_S$ übergeben kann – $(b_1, s_1) \in M_{\mathcal{R}}$, als auch eine Box b_2 logische Werte von einem Signal s_2 erhalten kann – $(s_2, b_2) \in M_{\mathcal{R}}$. Wie in Abb. 5 dargestellt, kann ein Signal $s \in M_S$ sowohl über mehrere Quellen $b_{q1}, b_{q2}, \ldots \in M_B$ als auch über mehrere Senken $b_{s1}, b_{s2}, \ldots \in M_B$ verfügen.

Mit der Menge der Nachfolger $\mathcal{N}_G^+(x) = \{y|(x,y) \in \mathcal{R}\}$ und der Vorgänger $\mathcal{N}_G^-(x) = \{y|(y,x) \in \mathcal{R}\}$ für beliebige gerichtete Graphen $G = (X, \mathcal{R})$ und $x \in X$ definieren wir das Hardware-Modell wie folgt:

Definition 3.1 *: Es seien M_I, M_O, M_E, M_L und M_S paarweise disjunkte, nichtleere Mengen und M_B, $M_{\mathcal{R}}$ wie oben definiert. $M = (M_I, M_O, M_E, M_L, M_S, M_{\mathcal{R}})$ wird genau dann als **Hardware-Modell** bezeichnet, wenn der entsprechende gerichtete bipartite Graph $G(M) = (M_B, M_S, M_{\mathcal{R}})$ [Len90, Spo95] folgenden Bedingungen genügt:*

1. $\left\{x|x \in M_B \cup M_S \wedge \mathcal{N}_{G(M)}^-(x) = \emptyset\right\} = M_I$,
2. $\left\{x|x \in M_B \cup M_S \wedge \mathcal{N}_{G(M)}^+(x) = \emptyset\right\} = M_O$,
3. *jeder Kreis in $G(M)$ enthält ein Element von M_L.*

M_I ist die Menge aller Quellen von $G(M)$ und M_O die Menge aller Senken von $G(M)$. Bedingung 3 bringt den synchronen getakteten Charakter der zugrundeliegenden Hardware dadurch zum Ausdruck, daß es keine Kreise gibt, die ausschließlich Elemente von $M_E \cup M_S$ enthalten.

3.2 Der Cone—Begriff

Um eine Senkung der Simulationszeit zu erreichen, wird gemäß Abschnitt 2.3 das Hardware-Modell mit dem Ziel partitioniert, die dabei erhaltenen Modellteile auf mehreren Prozessoren getrennt simulieren zu können. Die damit verbundene Auftrennung der Signale bzw. der Elemente aus der Menge der Relationen $M_{\mathcal{R}}$ (in der Sprache der Graphen: der Schnitt der entsprechenden Kanten des äquivalenten bipartiten Graphen $G(M)$) führt in der Regel zu einem beachtlichen Kommunikationsbedarf zwischen den einzelnen Prozessoren. Der

verwendete *clock-cycle*–Algorithmus (s. Abschnitt 2.1) verlangt eine Beschränkung bei der Aufschneidung von Signalen aus M_S auf die Zyklusgrenzen. Daraus ergibt sich, daß die Aufteilung der Elemente aus M_B auf die Prozessoren nicht beliebig erfolgen kann.

Im folgenden werden die zur Partitionierung des Hardware-Modells notwendigen Definitionen aufgestellt:

Definition 3.2 *: Der* ***fan-in–Cone*** $co_I(x)$ *eines Elementes* $x \in M_O \cup M_E \cup M_L$ *ist rekursiv durch folgende Bedingungen definiert:*

1. $x \in co_I(x)$,
2. $y \in M_E \wedge \mathcal{N}^+_{G(M)}(\mathcal{N}^+_{G(M)}(y)) \cap co_I(x) \neq \emptyset \rightarrow y \in co_I(x)$.

Analog dazu erfolgt die Definition eines fan-out–Cones $co_O(x)$:

Definition 3.3 *: Der* ***fan-out–Cone*** $co_O(x)$ *eines Elementes* $x \in M_I \cup M_E \cup M_L$ *ist durch folgende Bedingungen definiert:*

1. $x \in co_O(x)$,
2. $y \in M_E \wedge \mathcal{N}^-_{G(M)}(\mathcal{N}^-_{G(M)}(y)) \cap co_O(x) \neq \emptyset \rightarrow y \in co_O(x)$.

Auf dieser Grundlage können Cones (fan-in–Cones) $co(x)$ im engeren Sinne definiert werden, die im folgenden die Grundbausteine für eine Aufteilung des Hardware-Modells bilden [SUM87, MTSDA93, Man92]:

Definition 3.4 *: Ein* ***Cone*** $co(x)$ *ist ein fan-in–Cone* $co_I(x)$ *mit* $x \in M_O \cup M_L$.

Ein Cone $co(x)$ faßt somit neben dem Kopfelement $x \in M_O \cup M_L$ selbst alle die Boxen aus M_E zusammen, die für die Auswertung des Kopfelementes relevante Beiträge in Form logischer Werte liefern. Ausgehend von der Definition des Cones beinhaltet die Schnittmenge, gebildet aus $co(x)$ und der Vereinigungsmenge $M_O \cup M_L$ der globalen Ausgänge und der speichernden Elemente, genau das Kopfelement des Cones: $co(x) \cap (M_O \cup M_L) = \{x\}$.

Für ein gegebenes Modell M kann man eindeutig die Menge aller zugehörigen Cones $co(x)$ bilden:

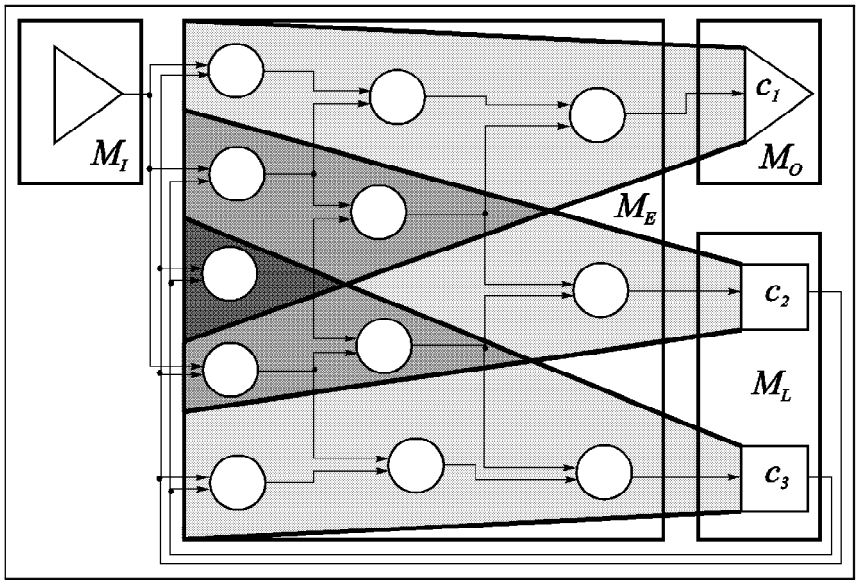


Abbildung 6: Hardware-Modell mit Cones (schattiert)

Definition 3.5 *: Die zum Modell M gehörige* ***Cone-Menge*** *ist definiert durch* $Co(M) = \{c | \exists x(x \in M_O \cup M_L \wedge c = co(x)\}$

Die Elemente der Cone-Menge bilden die vollständige Menge der Grundbausteine von uns betrachteter Zerlegungen des Hardware-Modells. Die Anzahl der zu M gehörigen Cones c der Cone-Menge $Co(M)$ beträgt:

$$|Co(M)| = |M_L| + |M_O| = m_c \ . \tag{3.1}$$

Für relevante Modelle liegt das Verhältnis der Anzahl der Cones m_c zu der Anzahl der Boxen m_E ungefähr bei $1 : 10$.

In Abb. 6 ist ein einfaches Hardware-Modell M mit seinen Komponenten M_I , M_O , M_E und M_L dargestellt, deren Elemente mit Signalen untereinander verbunden sind. Weiterhin sind durch starke Linien die schattierten Bereiche eingegrenzt worden, die ausgehend von den 3 möglichen Kopfelementen alle Elemente der entsprechenden Cones c_1 , c_2 , c_3 überdecken. Verschiedene Cones c_i können sich überlappen (vgl. Abb. 6 – stärker schattierte Bereiche), d.h. es gibt Boxen, die zu 2 oder mehreren Cones gehören – $c_j \cap c_l \neq \emptyset,\ j \neq l$;

$$\sum_{i=1}^{m_c} |c_i| \geq \left| \bigcup_{i=1}^{m_c} c_i \right| = |M_L| + |M_O| + |M_E| \tag{3.2}$$

(Abb. 6 – 4 Boxen sind in 2 Cones und eine Box ist in allen 3 Cones enthalten). Aufgrund der sich verzweigenden Signale bestimmen die logischen Ausgangswerte dieser von mehreren Cones erfaßten Boxen die Eingangswerte mehrerer Kopfelemente der Cones c_i gleichzeitig. Verteilt man nun die entsprechenden Cones auf verschiedene parallele Prozessoren, müssen diese Boxen zwei- oder mehrfach ausgewertet werden.

Mit der Aufteilung des Modells auf der Basis der Cones, die die Kommunikation im Rahmen des betrachteten *clock-cycle*–Algorithmus zwischen den Prozessoren vereinfacht und an die Zyklusgrenzen verbannt, ergibt sich der Nachteil der Mehrfachauswertung von Boxen. Um die Aufteilung des Hardware-Modells und die daraus resultierende Mehrfachauswertung von Boxen bewerten und letztere nachfolgend minimieren zu können, ist die Definition folgender Größen hilfreich:

Definition 3.6 ***: boxbezogener Cone-Überlappungsgrad*** $u : M_E \rightarrow \aleph$*:*
Jeder Box aus M_E der logischen Elemente wird durch die Abbildung u eine natürliche Zahl zugeordnet ($\aleph$ ... Menge der natürlichen Zahlen), die angibt, von wievielen Cones diese Box erfaßt wird.

Zur weiteren Beschreibung vereinbaren wir:

Definition 3.7 *: Mit $\mathcal{C}$ sei eine beliebig gewählte nichtleere Teilmenge von $Co(M)$ bezeichnet, d.h. $\mathcal{C} = \{c_1, \ldots, c_l\} \subseteq Co(M),\ 1 \leq l \leq m_c$.*

Weiterhin ist es sinnvoll, Boxen zusammenzufassen, die zu genau den gleichen Cones gehören:

Definition 3.8 *: Sei $\mathcal{C}$ wie in Def. 3.7 definiert. Das* ***Überlappungsgebiet*** *$ovr(\mathcal{C})$ ist die Menge derjenigen Boxen, die genau von denjenigen Cones c_i erfaßt werden, die in $\mathcal{C}$ enthalten sind:*

$$ovr(\mathcal{C}) = \bigcap_{c \in \mathcal{C}} c \setminus \bigcup_{c \in Co(M) \setminus \mathcal{C}} c\ . \tag{3.3}$$

Aus diesen Definitionen leiten sich folgende Eigenschaften ab:

1. Die Menge der Boxen $M_E \cup M_L \cup M_O$ ist eindeutig und disjunkt in Überlappungsgebiete $ovr(\mathcal{C})$ aufteilbar. Sei

   $$P^* = P\left(Co\left(M\right)\right) \setminus \{\emptyset\} \tag{3.4}$$

   die Potenzmenge von $Co\left(M\right)$ ohne die leere Menge, so gilt:

(a)

$$M_E \cup M_L \cup M_O = \bigcup_{\mathcal{C} \in P^*} ovr(\mathcal{C}) \tag{3.5}$$

(b)

$$|M_E| + |M_L| + |M_O| = \sum_{\mathcal{C} \in P^*} |ovr(\mathcal{C})| \ . \tag{3.6}$$

Bem.: Ein Großteil der Überlappungsgebiete $ovr(\mathcal{C})$ für beliebige zugelassene $\mathcal{C} \subseteq Co(M)$ sind i.a. leere Mengen (vgl. Abb. 6: $|ovr(\mathcal{C})| = 0$ für $\mathcal{C} = \{c_1, c_3\}$).

2. Ist $\mathcal{C}$ nur eine Einermenge $\{c\}$, so *entartet* das Überlappungsgebiet $ovr(\mathcal{C})$ zur Menge der Boxen, die genau zum Cone c gehören.

3. Die Anzahl der Elemente von $\mathcal{C}$:

$$u_{\mathcal{C}} = |\mathcal{C}| \tag{3.7}$$

bringt den *Überlappungsgrad des Überlappungsgebietes* $ovr(\mathcal{C})$ zum Ausdruck, d.h. von wievielen Cones die Boxen des Überlappungsgebietes $ovr(\mathcal{C})$ erfaßt werden (man beachte den Unterschied zum boxbezogenen Überlappungsgrad u aus Definition 3.6 - dort wird jeder einzelnen Box aus M_E dieser Wert u zugeordnet).

4. Die Menge der Boxen eines Cones c ist aus allen Überlappungsgebieten $ovr(\mathcal{C})$ eindeutig zusammensetzbar, deren Mengen von Cones $\mathcal{C}$ den Cone c enthalten:

(a)

$$c = \bigcup_{(\mathcal{C} \in P^* \wedge c \in \mathcal{C})} ovr(\mathcal{C}) \ , \tag{3.8}$$

(b)

$$|c| = \sum_{(\mathcal{C} \in P^* \wedge c \in \mathcal{C})} |ovr(\mathcal{C})| \ . \tag{3.9}$$

Auf der Basis der Überlappungsstruktur der Cones ist es möglich, einen Hypergraphen zu konstruieren, dessen Knoten mit den Cones (bzw. alternativ dazu: mit den Überlappungsgebieten des Grades 1) und dessen Hyperkanten mit den Cone-Mengen identifiziert werden, deren Überlappungsgebiete einen Überlappungsgrad größer als 1 haben. Überlappungsgebiete, die keine Box enthalten, liefern keine Hyperkante. Diese Überlegungen spiegeln sich in der folgenden Definition wider (vgl. auch [Len90]):

Definition 3.9 *: Der* ***Überlappungs-Hypergraph*** $GU = (V, E)$ *besteht aus der endlichen Menge von Knoten* $V = Co(M)$ *und einer Menge von Hyperkanten* $E \subseteq P^*$, P^* *aus (3.4), wobei* $\mathcal{C} \in E$ *genau dann gilt, wenn*

1. $|\mathcal{C}| > 1$ *und*
2. $|ovr(\mathcal{C})| > 0$.

Sowohl die Knoten als auch die Hyperkanten werden mit Hilfe einer Abbildung $\nu : V \rightarrow \aleph$ und $\mu : E \rightarrow \aleph$ bewertet. Dabei erfolgt die Knotenbewertung ν eines Knotens $c_i \in V$ mit der Anzahl der Boxen, die von genau dem Cone c_i erfaßt werden: $\nu_i = \nu(c_i) = |ovr(\{c_i\})|$ und die Bewertung einer Hyperkante $\mathcal{C}_j \in E$ mit der Anzahl der Boxen, die von genau den Cones c_i erfaßt werden, die in $\mathcal{C}_j$ enthalten sind: $\mu_j = \mu(\mathcal{C}_j) = |ovr(\mathcal{C}_j)|$.

Neben dieser Definition des bewerteten ungerichteten Hypergraphen GU sind andere Varianten denkbar, z.B.:

1. $GU' \equiv GU$ mit einer anderen Knotenbewertung $\nu'_i = \nu'(c_i) = |c_i|$ und $\mu' = \mu$,
2. MANJIKIAN führte einen gerichteten Hypergraphen GU'' ein, dessen Knoten und Kanten wiederum denen von GU entsprechen [Man92]. Eine beliebige Hyperkante wird dann in Richtung genau eines Knotens nach innen inzident. Die Kantenbewertung μ'' erfolgt unverändert, die Knotenbewertung ν'' erfolgt in Anlehnung an die Knotenbewertung ν. Die Knotenbewertung wird durch die Anzahl der Boxen ergänzt, welche der Bewertung aller der Hyperkanten entspricht, die mit diesem betrachteten Knoten nach innen inzident sind.

In Abb. 7 ist ein Beispiel für einen Überlappungs-Hypergraphen GU dargestellt, wie er sich aus der Abb. 6 ableiten läßt.

3.3 Partitionierung auf Grundlage des Cone–Begriffs

Wir wollen zunächst zur Begriffsklärung eine allgemeine Formulierung des Partitionierungsbegriffs einführen. Dazu betrachten wir zwei nichtleere Mengen U und V.

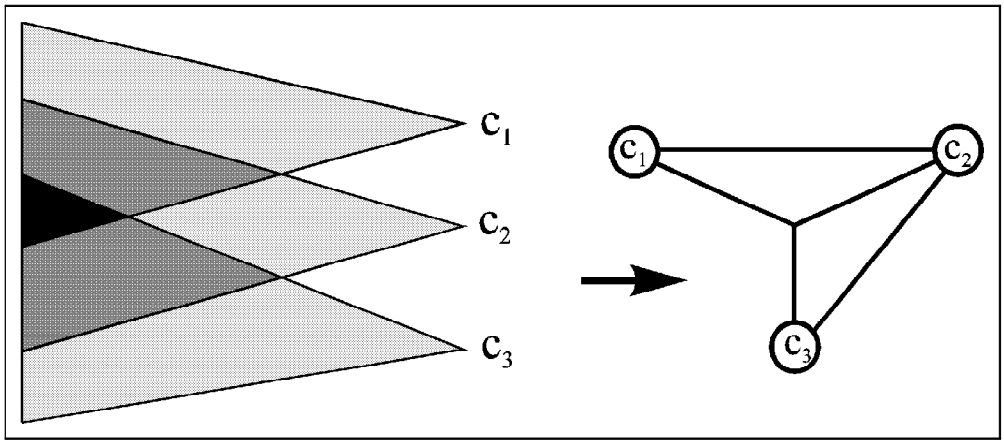


Abbildung 7: Darstellung des Überlappungs-Hypergraphen GU zu Abb. 6

Definition 3.10 *: Eine* ***Partitionierung*** *von U bezüglich V ist eine eindeutige Abbildung $\Phi : U \rightarrow V$, die jedem $u \in U$ genau ein $v \in V$ zuordnet.*

Anschaulich verkörpert ein als Funktionswert der Abbildung Φ auftretendes $v \in V$ ein Objekt, welches als Repräsentant einer Partitionskomponente aufgefaßt werden kann, die gerade die Elemente $u \in U$ zusammenfaßt, welche vermöge Φ auf v abgebildet werden. Damit definieren wir den Begriff der Partition wie folgt:

Definition 3.11 *: Eine* ***Partition*** *Ψ_Φ von U bezüglich der Partitionierung $\Phi : U \rightarrow V$ ist gegeben durch: $\Psi_\Phi = \{\Phi^{-1}(v) \mid v \in cod\ \Phi\}$, wobei $cod\ \Phi$ der Wertebereich (Co-Domain) von Φ ist.*

Remark 3.12 *Falls die Partionierung $\Phi : U \rightarrow V$ nicht surjektiv ist, existieren $v \in V$, die keine Partitionskomponente repräsentieren.*

Remark 3.13 *Die Partition Ψ_Φ ist eine Zerlegung von U im mathematischen Sinn.*

Im vorgegebenen Rahmen wird U zunächst durch die Menge der Cones $Co(M)$ und V durch die Menge der zur Verfügung stehenden m_b Prozessoren gebildet[4]. Die Prozessormenge sei als Blockmenge $\mathcal{B}$ bezeichnet, mit den Blöcken (Prozessoren) b_i als Elementen.

[4]Später werden auch andere Strukturen betrachtet.

Um Laufzeiten für die parallele Simulation abschätzen zu können, ist es erforderlich, die *Last* W_i für jeden Prozessor b_i; $i = 1, \ldots, m_b$ zu berechnen. Für jede Box wird die zur Auswertung ihrer Funktion erforderliche Zeit zunächst mit der Einheitszeit 1 angesetzt[5]. Damit ergibt sich die Last W_i aus der Bestimmung der Anzahl der Boxen im Block b_i. Die Partition Ψ_Φ als eindeutige Zerlegung von $Co\,(M)$ liefert diese Aussage im Zusammenhang mit der Kenntnis der Überlappungsgebiete (bzw. des Überlappungs-Hypergraphen) $ovr(\mathcal{C})$ des Hardware-Modells M. Die Bestimmung der Last W_i eines beliebigen Blockes b_i erfolgt aus der Zuordnung der Cones zu diesem Block: $\Phi^{-1}\,(b_i) = \left\{c_1^i, c_2^i, \ldots, c_{m_i}^i\right\}$ mit $m_c = \sum_{i=1}^{m_b} m_i$, wobei jedem c_j^i eineindeutig ein $c \in Co(M)$ zugeordnet ist:

$$W_i = \left| \bigcup_{(\mathcal{C} \in P^* \wedge \mathcal{C} \cap \Phi^{-1}(b_i) \neq \emptyset)} ovr\,(\mathcal{C}) \right| = \sum_{(\mathcal{C} \in P^* \wedge \mathcal{C} \cap \Phi^{-1}(b_i) \neq \emptyset)} |ovr(\mathcal{C})| \tag{3.10}$$

Folgende Grenzfälle sind in dieser Formel enthalten:

1. $m_b = 1$ (sequentielle Simulation – vgl. Gl. 3.6):
   $W_{seq} = \sum_{\mathcal{C} \in P^*} |ovr(\mathcal{C})| = |M_E| + |M_L| + |M_O|$
2. $m_b = m_c$ (Grenzfall der parallelen Simulation, wobei jedem Block genau ein Cone zugeordnet wird – vgl. Gl. 3.9):

$W_i = \sum_{(\mathcal{C} \in P^* \wedge c_i \in \mathcal{C})} |ovr(\mathcal{C})| = |c_i|.$

Führt man ausgehend vom Beispiel der Abb. 6 eine Partitionierung derart durch, daß man die 3 vorhandenen Cones auf 2 Blöcke aufteilt, z.B. $\Phi^{-1}\,(b_1) = \{c_1, c_2\}$ und $\Phi^{-1}\,(b_2) = \{c_3\}$, ergibt sich folgendes für die Blocklasten:

$W_1 = |ovr(\{c_1\})| + |ovr(\{c_2\})| + |ovr(\{c_1, c_2\})| + |ovr(\{c_2, c_3\})| + |ovr(\{c_1, c_2, c_3\})| = 11,$

$W_2 = |ovr(\{c_3\})| + |ovr(\{c_2, c_3\})| + |ovr(\{c_1, c_2, c_3\})| = 7$

und

$W_{seq} = |M_E| + |M_L| + |M_O| = 15$

$\Longrightarrow W_1 + W_2 > W_{seq}.$

[5] Eine Verallgemeinerung bezüglich spezieller logischer Funktionen auf unterschiedliche Auswertungszeiten ist möglich.

4 Modellpartitionierung

4.1 Ziel der Partitionierung

Ziel der Partitionierung eines Prozessormodells ist es, im Vergleich zur sequentiellen Simulation mit Hilfe der parallelen Simulation eine möglichst signifikante *Reduzierung der Simulationszeit* zu erreichen. Im folgenden beziehen wir uns jeweils auf die Zeit, die zur Simulation eines Zyklus im Rahmen des *clock-cycle*–Algorithmus (vgl. Abschnitt 2.1) benötigt wird. Quantitativ erfaßt wird dieses Ziel durch den *speed-up*–Faktor

$$S(m_b) = \frac{T_{seq}}{T_{par}(m_b)} \tag{4.1}$$

und durch die *Effizienz*

$$E(m_b) = \frac{S(m_b)}{m_b} \, . \tag{4.2}$$

Die sequentielle Simulationszeit T_{seq} identifizieren wir im Rahmen unseres Modells mit der sequentiellen Rechenlast W_{seq} (W_{seq} – s. erste Folgerung von Gleichung (3.10)) . Die parallele Simulationszeit $T_{par}(m_b)$ wird im weiteren noch genauer spezifiziert.

Das Partitionierungsziel wird im wesentlichen durch die beiden Faktoren *Kommunikations-* bzw. *Synchronisationsoverhead* und *Verteilung der Rechenlasten* bestimmt. Davon ausgehend kann man die parallele Simulationszeit $T_{par}(m_b)$ aus den beiden additiven Bestandteilen der parallelen Rechenlast $W_{par}\,(m_b) = \max_{1 \le i \le m_b} \{W_i\}$ (s. Gl. (3.10)) sowie der Kommunikations- und Synchronisationszeit t_{sync} zusammensetzen.

Eine obere Schranke für den *speed-up*–Faktor erreicht man unter Berücksichtigung dieser Faktoren bei einer gleichmäßig verteilten Rechenlast – $W_{par}^{\min} = W_i = W_{seq}/m_b$ – und einem minimalen Kommunikationsoverhead – $t_{sync}^{\min}$:

$$S_{\max}(m_b) = \frac{m_b}{1 + \frac{t_{sync}^{\min}}{W_{seq}} m_b} \, . \tag{4.3}$$

Praktisch sind sowohl die parallele Rechenlast $W_{par}\,(m_b)$ als auch der Kommunikationsoverhead t_{sync} Funktionen der Anzahl der Prozessoren m_b, was

dazu führt, daß der *speed-up*–Faktor mit wachsender Prozessoranzahl eine Sättigung erreichen und sogar wieder absinken kann [Man92].

Auf der Grundlage des *clock–cycle*—Algorithmus haben wir eine erste Stufe der Partitionierung durch Zusammenfassung von Boxen in Cones vorgenommen (vgl. Abschnitt 3.2). Die zwischen den Prozessoren notwendige Kommunikation wird durch die Beschränkung derselben auf die Zyklusgrenzen reduziert. Ausgehend von einer effektiven Realisierung der Interprozessorkommunikation und dem extrem hohen Verhältnis der Anzahl von Boxen zur Anzahl von Prozessoren geben wir zunächst der Minimierung der parallelen Rechenlast $W_{par}(m_b)$ die Priorität. Allerdings zieht die Verwendung der Cone-Menge $Co(M)$ als Ausgangsbasis für die folgende Partitionierung in die Blockmenge $\mathcal{B}$ das zusätzliche Problem der *Mehrfachauswertung* von Boxen nach sich, welche zu einer Erhöhung der mittleren parallelen Rechenlast

$$\overline{W}_{par} = \frac{1}{m_b} \sum_{i=1}^{m_b} W_i \tag{4.4}$$

gegenüber der minimalen parallelen Rechenlast $W_{par}^{\min} = W_{seq}/m_b$ führt.

Ausgehend von den diesen Betrachtungen liegen damit 2 Hauptkriterien mit ihrer jeweiligen Zielrichtung vor, die zu beeinflussen sind und der Optimierung der Partitionen dienen:

1. gleichmäßige Lastverteilung,
2. minimale Mehrfachauswertung.

4.2 Zielfunktion zur Bewertung von Partitionen

Ausgehend von den beiden aus dem Partitionierungsziel abgeleiteten Optimalitätskriterien gilt es nun, Zielfunktionen[6] abzuleiten, die der Bewertung von Partitionen im Rahmen der Optimierung dienen.

MANJIKIAN , der ebenfalls eine Partitionierung auf der Grundlage von Cones vorgenommen hat [Man92], verwendet zur Bewertung der Partitionen folgende *Zielfunktion*:

$$\Omega_{Man} = \sum_{i=1}^{m_b} \left| W_i - \frac{W_{seq}}{m_b} \right| \ . \tag{4.5}$$

[6] äquivalente Bezeichnungen sind Güte- und Optimierungsfunktion

Im Idealfall, d.h. bei gleichmäßiger Lastverteilung und verschwindender Mehrfachauswertung, nimmt die Zielfunktion den Wert 0 an.

Dem Vorteil der Kompaktheit dieser Zielfunktion steht der Nachteil gegenüber, daß die beiden Optimierungskriterien in fest gefügter Art und Weise in diese Funktion eingehen und nicht auf verschiedene Hardware-Modelle und unterschiedliche Partitionierungsalgorithmen individuell in ihrem gegenseitigen Verhältnis angepaßt werden können.

Dieser Nachteil wird durch folgende neue parametrisierte Zielfunktion beseitigt:

$$\Omega_\alpha = \alpha_1 \left(\overline{W}_{par} - \frac{W_{seq}}{m_b} \right) + \alpha_2 \sqrt{\sum_{i=1}^{m_b} \frac{\left(W_i - \overline{W}_{par} \right)^2}{m_b}} \tag{4.6}$$

mit $\alpha_1 + \alpha_2 = 1$ und $\alpha = (\alpha_1, \alpha_2)$.

Der erste Summand verschwindet genau dann, wenn keine Mehrfachauswertung von Boxen erforderlich ist, d.h. wenn die mittlere parallele Rechenlast mit dem m_b-ten Teil der sequentiellen Rechenlast übereinstimmt. Dagegen geht der zweite Summand gegen Null, wenn die Streuung der parallelen Lasten W_i verschwindet, d.h. wenn die parallelen Lasten gleichmäßig auf die Prozessoren verteilt werden. Im Idealfall gleichmäßiger Lastverteilung und ausbleibender Mehrfachauswertung verschwindet die Zielfunktion. Die Wahl des Verhältnisses der Parameter α_1 und α_2 erlaubt eine differenzierte Wichtung der beiden Optimalitätskriterien. So wird bei einem Partitionierungsalgorithmus, der *a priori* eine gleichmäßige Lastverteilung gegenüber der Mehrfachauswertung bevorzugt, das Verhältnis $\alpha_1/\alpha_2 \gg 1$ gewählt, so daß bei der Zielfunktion die Minimierung der Mehrfachauswertung im Vordergrund steht. Der modulare Aufbau der Zielfunktion in der Form $\Omega_\alpha = \sum_{i=1}^{n} \alpha_i \Omega_i$ mit $\sum_{i=1}^{n} \alpha_i = 1$ und $\alpha = (\alpha_1, \ldots, \alpha_n)$, erlaubt es außerdem, weitere Kriterien Ω_i aufzunehmen und zu wichten.

Eine weitere Möglichkeit zur Wahl der Zielfunktion besteht darin, den Nenner des *speed-up*–Faktors, die parallele Rechenlast $W_{par}\,(m_b)$, zu minimieren:

$$\Omega_{W_{\max}} = \max_{1 \le i \le m_b} W_i \; . \tag{4.7}$$

Diese Zielfunktion geht im Idealfall gegen den Wert W_{seq}/m_b. Hier ist man, wie auch bei der Zielfunktion von MANJIKIAN (s. Gl. (4.5)), nicht in der Lage, die beiden Optimalitätskriterien getrennt zu beeinflussen. So werden z.B. verschiedene Partitionen

mit einer identischen Box- und Cone-Menge aber unterschiedlichen Rechenlasten gleich bewertet:

1. Partition: $W_1 = 700,\ W_2 = 500,\ W_3 = 500,\ W_4 = 400$
2. Partition: $W_1 = 700,\ W_2 = 700,\ W_3 = 700,\ W_4 = 500$
3. Partition: $W_1 = 700,\ W_2 = 500,\ W_3 = 300,\ W_4 = 100$.

Offensichtlich ist die 3. Partition diejenige mit der kleinsten mittleren parallelen Rechenlast und damit auch die mit der geringsten Mehrfachauswertung. Gleichzeitig zeigt diese Partition aber auch die stärkste Streuung und damit die ungleichmäßigste Lastverteilung. Bei dieser Partition kann man hoffen, durch Umverteilung (Verschiebung) von Cones der Blöcke b_1 und b_2 auf die Blöcke b_3 und b_4 eine Verbesserung zu erreichen, die aber u.U. eine erhöhte Mehrfachauswertung bedeuten kann. Dagegen verfügt die 2. Partition über die gleichmäßigste Lastverteilung, wobei aber ein hoher Anteil von Boxen mehrfach ausgewertet werden muß. Hier kann man erwarten, daß durch Austausch von Cones zwischen verschiedenen Blöcken eine Reduktion der Mehrfachauswertung auftritt. Allerdings können die beteiligten Cones unterschiedlich viele Boxen besitzen, womit eventuell die gleichmäßige Lastverteilung wieder verloren geht. Die erste Partition liegt in der Beurteilung zwischen der zweiten und der dritten. Die Bewertung gemäß der dritten Zielfunktion macht aber zwischen diesen Partitionen keinen Unterschied. Hat man das globale Minimum der Zielfunktion erreicht, spielt das keine Rolle. Man muß sich für eine von diesen drei Partitionen entscheiden, die alle 3 eine gleiche parallele Rechenlast haben und bei gleichem Kommunikationsoverhead t_{sync} identische *speed-up*–Werte liefern. Im Optimierungsprozeß selbst können diese Unterschiede in den Partitionen aber entscheidend für die weitere Gestaltung der Optimierung sein, wie oben anhand des Beispiels erläutert wurde.

4.3 Bisherige Partitionierungsalgorithmen

Partitionierungsalgorithmen haben das Ziel, Partitionen zu erstellen, die nach bestimmten Kriterien (s. Abschnitte 4.1 und 4.2) optimiert sind. Für die Modellpartitionierung im Vorfeld der parallelen Logiksimulation gibt es eine Reihe von Algorithmen, die sich mehr oder weniger für unsere konkrete Problemstellung (sehr komplexe Modelle mit einer großen Anzahl von Boxen; Cones als Grundbausteine –

Überlappung der Cones) eignen. Im folgenden werden einige Algorithmen mit ihren Grundideen vorgestellt und auf ihre Anwendbarkeit für unsere Modelle untersucht.

In [Spo95] erhält man einen Überblick zu Partitionierungsalgorithmen bezüglich der parallelen Logiksimulation.

Einfache Partitionierungsverfahren gehen so vor, daß man einmalig eine Partition erzeugt und diese im Gegensatz zu den iterativen Algorithmen dann nicht mehr verändert wird. Das kann gemäß einer zufälligen und gleichmäßigen Verteilung der Elemente auf die Modellteile geschehen, die Elemente können aber auch z.B. gemäß existierender innerer Verknüpfungen ausgewählt werden. Diese Verfahren liefern i.a. initiale Partitionen, die mit Hilfe iterativer Algorithmen weiterverarbeitet werden können. Mit diesen einfachen Verfahren kann man insbesondere große Modelle schnell vorverarbeiten. Auch für Cones als Grundbausteine der Partition eignen sich diese initialen Partitionierungen ([MTSDA93]). MUELLER–THUNS bildet auf der Grundlage von Cone-Ketten Partitionen, deren Cones über die speichernden Elemente aus M_L miteinander verknüpft sind. Damit zielt er auf einen möglichst geringen Umfang von Interprozessorkommunikationsbeziehungen. Die Cone-Menge $Co(M)$ wird dabei gleichmäßig auf die Partitionskomponenten aufgeteilt, was nicht unbedingt zum Lastausgleich führen muß (die Cones $c \in Co(M)$ können extrem unterschiedlich viele Elemente haben) und auch keinerlei Einfluß auf die Mehrfachauswertung nimmt.

Die einfachen Partitionierungsverfahren lassen sich auch in die sog. *bottom-up* – Vorgehensweise einpassen. Hier werden über 2 oder mehrere Stufen die Elemente zu größeren Einheiten bis hin zu den gewünschten Modellpartitionen zusammengefaßt (*Clusterverfahren*). Die in den jeweiligen Stufen verwendeten Partitionierungsalgorithmen können sehr verschieden sein – z.B. einfache Verfahren mit zufälliger Verteilung der Elemente oder Verfahren mit einer Verteilung der Elemente entsprechend innerer Signalpfade bzw. gemäß dicht vernetzter Schaltungsteile. Aber auch komplexere Algorithmen, die sich meist iterativ einer suboptimalen Verteilung der Bausteine in der jeweiligen Stufe nähern, finden Verwendung. Entsprechend differenzierte Zwischenobjekte werden im Verlauf des Clusterverfahrens gebildet (Cones [SUM87, Man92, MTSDA93], Cluster [Hil81], Petals [SB93]). Diese *bottom-up* – Strategie kommt unseren Hardware-Modellen weitgehend entgegen. Aufgrund der Größe der Modelle mit ca. $10^6 - 10^7$ Elementen ist eine Partitionierung über $5 - 6$ Größenordnung in einer Stufe für viele Algorithmen praktisch nicht durchführbar.

Eine wichtige Gruppe von Algorithmen sind die *iterativen Verfahren* (*Iterative Improvement Heuristics*). Sie sind (oft nur heuristische) Optimierungsverfahren und

minimieren die Zahl der Verbindunssignale zwischen den Modellteilen (*Min-Cut – Verfahren*) oder maximieren die Zahl der innerhalb der Modellteile existierenden Signalpfade. Eine Einbeziehung des Lastausgleiches ist ebenfalls möglich (*Ratio-Cut –Verfahren*). Im Rahmen einer *bottom-up*–Vorgehensweise finden solche Verfahren bevorzugt in der finalen Phase der Partitionierung Verwendung. Diese Algorithmen lassen sich aber auch sowohl bei kleineren Modellen direkt auf der Elementstufe (vgl. gerichteter bipartiter Graph in Abschnitt 3.1) als auch für unsere Modelle auf der Basis der Cones (Überlappungs–Hypergraph in Abschnitt 3.2) anwenden. Im ersten Fall können die geschnittenen Kanten des Graphen tatsächlich mit geschnittenen Signalpfaden identifiziert werden, die zu zusätzlicher Kommunikation zwischen den Prozessoren führen und damit minimiert werden müssen. Im zweiten Fall entspricht das Schneiden von Hyperkanten vom Überlappungs–Hypergraphen beim Partitionieren einer zu minimierenden Mehrfachauswertung von Boxen (vgl. Abschnitt 4.1 und 4.2). In der Praxis bewährte iterative Verfahren [Len90] sind die Verfahren nach KERNIGHAN & LIN, die eine Partition durch fortwährenden Austausch zweier Elemente zwischen Partitionskomponenten optimieren [KL70], und nach FIDUCCIA & MATTHEYSES, die einzelne Elemente zwischen den Komponenten verschieben [FM82]. Beide Verfahren sind ursprünglich Bipartitionierungsverfahren. Sie wurden auch für den Einsatz zur Multipartitionierung (mehr als zwei Modellteile) modifiziert [KL70, San89]. Dabei werden Elementbewegungen (Austausch oder Verschiebung) zwischen jeweils zwei oder allen Modellteilen gleichzeitig in Betracht gezogen. Jeder möglichen Elementbewegung wird ein 'Gewinn' zugeordnet, der auf der Zielfunktion basiert. Es werden dann diejenigen Elemente verschoben oder ausgetauscht, die den größten Gewinn oder auch den geringsten Verlust versprechen. Um eine Entscheidung bei gleichem Gewinn herbeizuführen, können Gewinne höherer Ordnung eingeführt werden. Die bewegten Elemente werden dann markiert und festgehalten und der Zyklus wird fortgesetzt bis jedes Element markiert ist. Diese Zyklen werden so oft wiederholt, bis keine Verbesserungen der Optimierungsfunktion mehr zu verzeichnen sind.

Die Praktikabilität *randomisierter Algorithmen* (zur Definition s. [Kar90]) wird in [BDBK+90] diskutiert. Der Einsatz stochastischer Verfahren wie z.B. *Simulated Annealing* und *Genetische Algorithmen* zur Modellpartitionierung im Kontext der parallelen Logiksimulation wird in [Len90, Spo95] aufgrund sehr hoher Laufzeiten als nicht praktikabel eingeschätzt.

Ähnlich ist die Situation bei den *analytischen Verfahren*. Aufgrund unserer Modellgröße sind diese Verfahren durch ihren hohen Speicherbedarf nicht einsetzbar.

4.4 Hierarchische Partitionierung

4.4.1 Beschreibung des hierarchischen Ansatzes

Wie in Abschnitt 1 dargelegt, wird in naher Zukunft die Anzahl der logischen Elemente je Modell in der Größenordnung $10^6 - 10^7$ liegen. Das Ziel der Partitionierung ist, diese Elemente unter den oben beschriebenen Optimalitätskriterien auf verschiedene Blöcke $b_i \in \mathcal{B}$ (Prozessoren) zu verteilen. Dabei liegt die Anzahl der Blöcke $|\mathcal{B}| = m_b$ in der Größenordnung $10^1 - 10^2$, d.h. bei der Partitionierung ist eine Barriere von bis zu 10^6 Größenordnungen zu überwinden. Unter diesen Bedingungen scheint eine direkte Partionierung schwierig bzw. praktisch undurchführbar. Aus diesem Grund gehen wir von einem *hierarchischen Partitionierungsmodell* aus.

Hierarchische Lösungsstrategien werden bei der Bearbeitung extrem großer Datenmengen bzw. komplexer Probleme erfolgreich angewendet: z.B. bei der automatischen Farbbildverarbeitung [VMB+94], bei der Satellitenbildverarbeitung [GS93, Vil95], in der Robotik [RMS92] oder bei der Zeitreihenvorhersage in hochdimensionalen Datenräumen [DEFH93]. Diese Idee soll auf die hier zu bearbeitende Problemstellung übertragen werden.

Dazu betrachten wir den Partitionierungsansatz aus Def. 3.10 . Wie in Abschnitt 3.3 beschrieben ist, gilt für den Cone-Ansatz und den Variablen aus Def. 3.10 $U = Co\,(M)$ und $V = \mathcal{B}$, d.h.

$$\Phi : Co\,(M) \to \mathcal{B}\ . \tag{4.8}$$

Dieser direkten Partitionierung stellen wir einen hierarchischen Ansatz gegenüber:

Definition 4.1 *: Bei der* ***k-stufigen hierarchischen Partitionierung*** *von U bezüglich V in Anlehnung an Def. 3.10 mit der zusätzlichen Bedingung, daß $|V| \leq |U|$, wird die Abbildung Φ durch $\Phi_H : U \to V$ mit*

$$\Phi_H = \Phi_k \circ \Phi_{k-1} \circ \ldots \circ \Phi_1 \tag{4.9}$$

ersetzt, wobei

$$\Phi_j : V_j \to V_{j+1} \tag{4.10}$$

mit $V_1 = U$ bzw. $V_{k+1} = V$ und $|V_1| \geq |V_2| \geq \ldots \geq |V_{k+1}|$.

Im allgemeinen stellt Φ_H nur eine Approximation von Φ dar. Ziel dieses Hierarchieansatzes ist es, sukzessive die Größenordnungen der Anzahl der Elemente

in den V_j zu vermindern. Im konkret vorliegenden Problem (4.8) gehen wir zunächst von einem *zwei*-stufigen Ansatz aus, d.h. $V_1 = U = Co(M)$, $V_{k+1} = V = \mathcal{B}$ und $V_2 = \mathcal{S}$ als Menge von *Super-Cones* s_j. Seien weiter die Abbildungen Φ_1 als f und Φ_2 als g bezeichnet. Dann hat der zwei-stufige Ansatz die Form

$$\begin{gathered} Co(M) \xrightarrow{f} \mathcal{S} \xrightarrow{g} \mathcal{B} , \\ \Phi_H = g \circ f . \end{gathered} \tag{4.11}$$

Die Partitionierung von $Co(M)$ stellt jedoch nur eine abstrakte Beschreibungsstufe der Partitionierung der den Cones zugrundeliegenden Boxen aus $\mathcal{M} = M_E \cup M_L \cup M_O$ dar (vgl. Abschnitt 3.2). Damit kann man die Zusammenfassung von Boxen in Cones ebenfalls als eine Partitionierungsstufe Φ_0 ansehen und den zweistufigen Ansatz (4.11) erweitern:

$$\begin{gathered} \mathcal{M} \xrightarrow{\Phi_0} Co(M) \xrightarrow{f} \mathcal{S} \xrightarrow{g} \mathcal{B} , \\ \Phi_H^* = g \circ f \circ \Phi_0 . \end{gathered} \tag{4.12}$$

Bei diesem erweiterten hierarchischen Ansatz bleibt die Wahl der konkreten Partitionierungsalgorithmen für die Realisierung von f und g offen, so daß sie entsprechend der Eigenschaften des bipartiten Graphen M aus Def. 3.1, des Hypergraphen GU aus Def. 3.9 und den Optimierungskriterien aus Abschnitt 4.2 gewählt werden können. Dabei können z.B. für g Algorithmen verwendet werden, die in der ersten Partitionierungsstufe f auf Grund der Problemgröße nicht in Betracht kommen. Die initiale Partitionierung Φ_0 ist jedoch auf Grund der Cone-Definition 3.4 fixiert.

Es ist klar, daß selbst bei optimaler Wahl von g und f in ihren jeweiligen Partitionierungsstufen i.a. die Verkettung von g und f keine optimale Lösung liefert, da sie nur lokal in den Ebenen wirken. Die Bewertung solcher suboptimaler Heuristiken ist oft schwierig und problemgebunden [HHK95]. Andererseits stellt dieser Hierarchieansatz eine Möglichkeit dar, sehr große Modelle unter gewissen Optimalitätskriterien zu partitionieren. In dem hier vorgestellten zweistufigen Ansatz, bzw. dessen Erweiterung, sind die vorrangigen Optimierungsziele für die jeweiligen Partitionierungsstufen in Abb. 8 zusammengefaßt.

4.4.2 Kombination von Algorithmen

Der Erfolg von Partitionierungsalgorithmen ist wesentlich von den Eigenschaften der zu partitionierenden Datenmenge abhängig. Insbesondere trifft das auf so komplizierte

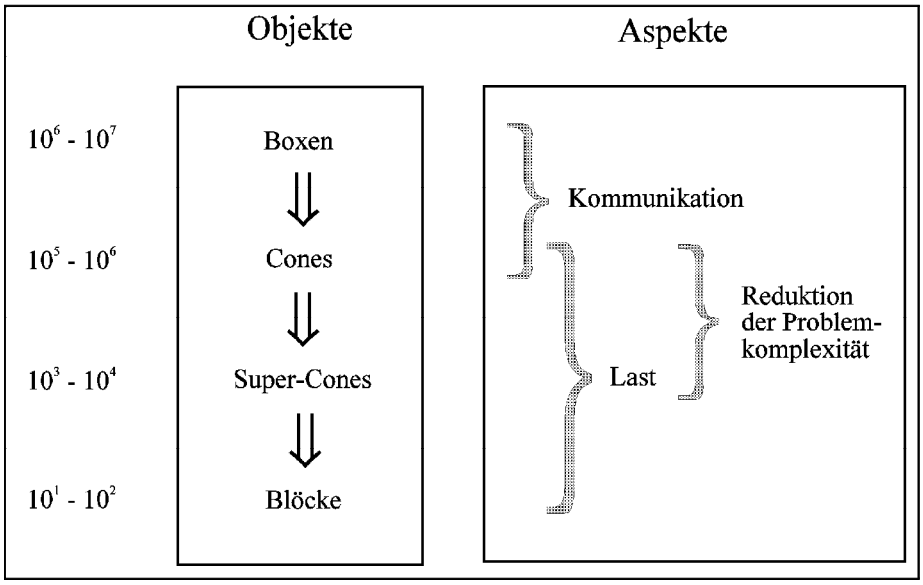


Abbildung 8: Hierarchiestufen im erweiterten Zwei-Stufen-Ansatz. Links ist jeweils die Anzahl der zu partitionierenden Objekte in der betrachteten Partitionierungstufe angegeben, rechts das hauptsächliche Optimierungsziel.

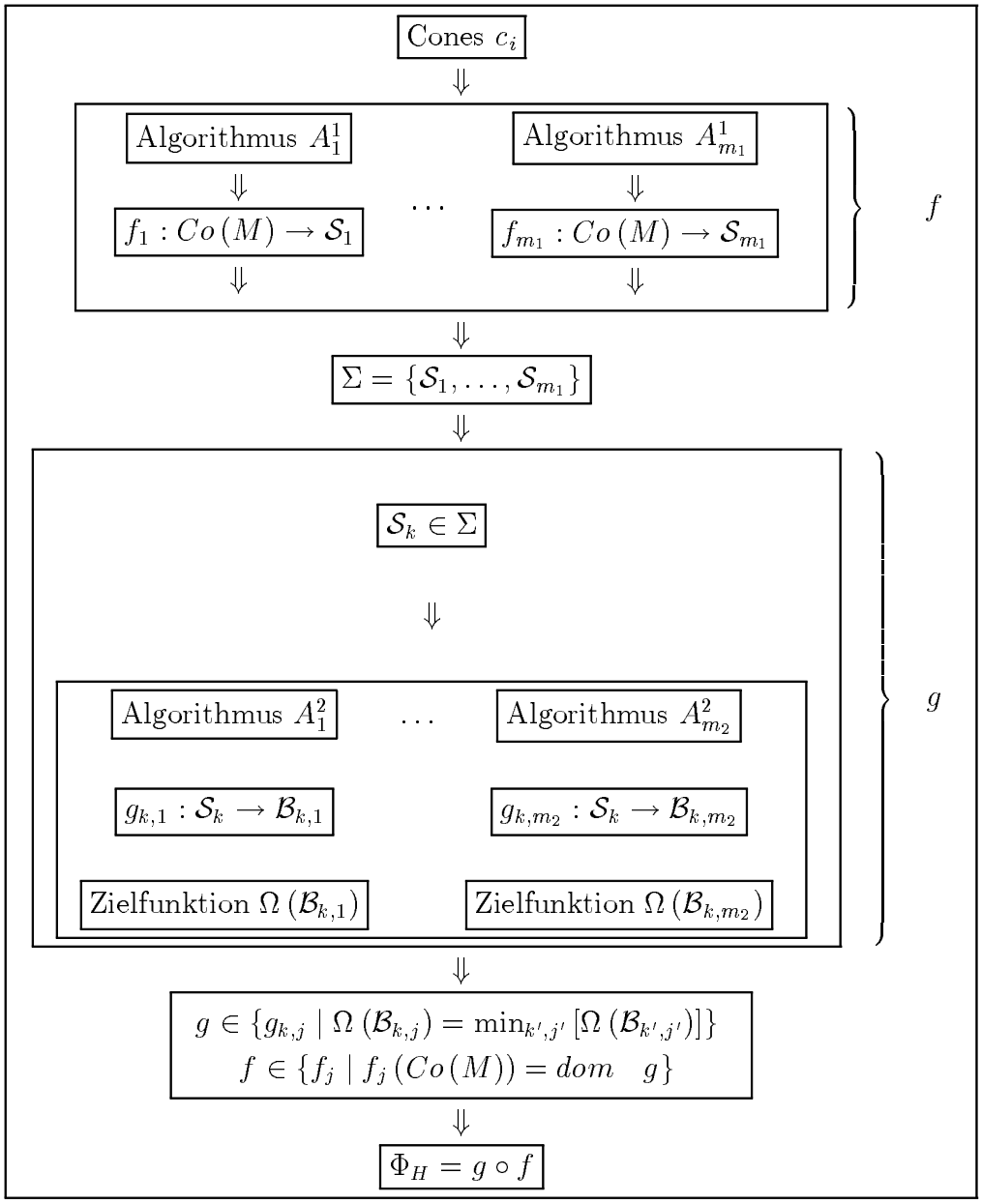
Gebilde wie den Hypergraphen GU aus Def. 3.9 zu. Deshalb ist es schwierig oder gar unmöglich, *a priori* zu entscheiden, welcher Algorithmus das Problem am besten löst. LENGAUER kommt in [Len90] zu dem Schluß, daß oft nur heuristische Methoden zum Ziel führen. Damit kommt der Wahl der Algorithmen für die Partitionierungsstufen eine entscheidende Bedeutung zu.

Wir verfolgen hier einen *parallelen* Ansatz. Verschiedene oder unterschiedlich parametrisierte Partitionierungsalgorithmen A_i^j, $i = 1 \ldots m_j$ arbeiten gleichzeitig auf der j-ten Partitionierungsstufe und erzeugen die Partitionen $\Psi_{A_i^j}$. Diese werden jeweils in der nachfolgenden Partitionierungstufe wieder von verschiedenen Algorithmen A_l^{j+} , $l = 1 \ldots m_{j+1}$ partitioniert und erzeugen Partitionen $\Psi_{A_i^j, A_l^{j+1}}$. Die Algorithmenmenge in jeder Hierarchiestufe kann sich dabei durchaus unterscheiden. Im Ergebnis aller k Hierarchiestufen entstehen Partitionen $\Psi_{A_{i_0}^0, A_{i_1}^1, \ldots, A_{i_k}^k}$ über der Cone-Menge $Co(M)$. Diese können an Hand der Optimalitätskriterien aus den Abschnitten 4.1 und 4.2 bewertet werden.

Die hier vorgestellte Idee des Wettbewerbs zwischen Algorithmen wurde in [JJNH91] bzw. in [KMP95] bei der Verwendung autonomer lernender Agenten (Subsysteme) zur Zeitreihenvorhersage dynamischer Systeme betrachtet. In

Verbindung mit einem hierarchischen Ansatz wurde diese Strategie in [JJ94] von JORDAN et al. eingeführt. Wir übertragen diese Idee hier auf den Wettbewerb zwischen Algorithmen zur Überwindung des Mangels an *a–priori*–Wissen.

Wenden wir uns wieder dem zweistufigen Ansatz (4.11) zu. In der ersten Hierarchiestufe werden Cones zu Super-Cone–Mengen $\mathcal{S}_i$ als Ergebnis des i-ten Algorithmus zusammengefaßt, die wiederum als Ausgangsmengen für die zweite Partitionierung dienen. Dieser Ansatz ist im folgenden Schema noch einmal zusammengefaßt:



(4.13)

4.4.3 Superpositionsprinzip für Partitionen

Der parallele Ansatz aus Abschnitt 4.4.2 läßt einen 'Wettbewerb' verschiedener Algorithmen und ihrer Kombinationen (Verkettungen) zu. Es ist jedoch weiter wünschenswert, daß direkt die Eigenschaften der Algorithmen in *einer* Hierarchiestufe kombiniert werden. Hier soll ein Konzept eingeführt werden, das eine solche Kombination zuläßt.

Wir betrachten wieder das zweistufige Partitionierungsproblem (4.11) und hier insbesondere die erste Partitionierungstufe mit den Partitionierungen $f_i : Co(M) \rightarrow \mathcal{S}_i$. Es ist plausibel anzunehmen, daß sich die den verschiedenen f_i zugrundeliegenden Heuristiken in unterschiedlichen Partitionen $\Psi_i = f_i^{-1}(\mathcal{S}_i)$ widerspiegeln, d.h. entsprechend der Heuristiken werden verschiedene Cones in Super-Cones s_i^j mit $j = 1 \ldots n_i$ zusammengefaßt. Die Eigenschaften der Algorithmen übertragen sich also auf die Super–Cones s_i^j. Wir führen nun das *Superpositionsprinzip für Partitionen* ein, welches gestattet, verschiedene Partitionen zu kombinieren und so die entsprechenden Heuristiken der zugrunde liegenden Algorithmen zu verschmelzen.

Definition 4.2 : *Sei* $\Pi = \{\Psi_1, \ldots, \Psi_k\}$ *ein System von Partitionen über der Grundmenge* U*. Die Elemente der* Ψ_i *seien mit* s_i^j*,* $j = 1 \ldots n_i$*, bezeichnet. Eine Menge* Ψ^**, heißt* ***Superposition*** *der Partitionen* $\Pi = \{\Psi_1, \ldots, \Psi_k\}$*, wenn sie den folgenden Bedingungen genügt:*

1. ***Nullmengenausschluß:*** $\emptyset \notin \Psi^*$
2. ***Erzeugendensystem:*** *für jedes* $s_i^j \in \Psi_i$ $(i = 1 \ldots k,\ j = 1 \ldots n_i)$ *existieren* $s_{l_1}^*, \ldots, s_{l_n}^* \in \Psi^*$ *so, daß* $s_i^j = s_{l_1}^* \cup \ldots \cup s_{l_m}^*$
3. ***Partitionsbedingung:*** Ψ^* *ist eine Partition von* U
4. ***Minimalität:*** *Sei* $\Psi^* = \{s_1^*, \ldots, s_n^*\}$ *ein Erzeugendensystem. Sei* $s_i^* \in \Psi^*$ *beliebig gewählt und* $\Psi^- = \Psi^* \setminus \{s_i^*\}$*. Dann ist* Ψ^- *nicht mehr erzeugend.*

Wir zeigen im folgenden für eine aus einem System von Partitionen direkt konstruierbare Menge die Superpositionseigenschaft:

Theorem 4.3 : *Sei* $\Pi = \{\Psi_1, \ldots, \Psi_k\}$ *ein System von Partitionen über der Grundmenge* U*. Die Elemente der* Ψ_i *seien mit* s_i^j*,* $j = 1 \ldots n_i$*, bezeichnet. Sei jetzt*

Ψ^* *als*

$$\Psi^* = \left\{ s^*_{j_1 \ldots j_k} \mid s^*_{j_1 \ldots j_k} = \bigcap_{i=1 \ldots k} s_i^{j_i} \text{ und } j_i = 1 \ldots n_i \right\} \setminus \{\emptyset\} \tag{4.14}$$

gegeben. Dann gilt: Ψ^* *ist eine Superposition im Sinne der Definition 4.2.*

Zum Beweis des Theorems 4.3 stellen wir zunächst das folgende Lemma auf:

Lemma 4.4 *: Für* $s_i^* \in \Psi^*$ *und* $s_j^* \in \Psi^*$ *mit* $i \neq j$ *gilt:* $s_i^* \cap s_i^* = \emptyset$*, d.h. die Elemente von* Ψ^* *sind paarweise disjunkt.*

Beweis des Lemmas:

Es seien $s^*_{i_1 \ldots i_k} \in \Psi^*$ und $s^*_{j_1 \ldots j_k} \in \Psi^*$ gegeben mit $(i_1, \ldots, i_k) \neq (j_1, \ldots, j_k)$. Dann existiert ein l so, daß $i_l \neq j_l$. Weiter ist $s^*_{i_1 \ldots i_k} = s_1^{i_1} \cap \ldots \cap s_l^{i_l} \cap \ldots \cap s_k^{i_k}$ und $s^*_{j_1 \ldots j_k} = s_1^{j_1} \cap \ldots \cap s_l^{j_l} \cap \ldots \cap s_k^{j_k}$. Für den Durchschnitt $s^*_{i_1 \ldots i_k} \cap s^*_{j_1 \ldots j_k}$ ergibt sich damit

$$\begin{aligned} s^*_{i_1 \ldots i_k} \cap s^*_{j_1 \ldots j_k} &= s_1^{i_1} \cap \ldots \cap s_l^{i_l} \cap \ldots \cap s_k^{i_k} \cap s_1^{j_1} \cap \ldots \cap s_l^{j_l} \cap \ldots \cap s_k^{j_k} \\ &= s_1^{i_1} \cap s_1^{j_1} \cap \ldots \cap s_l^{i_l} \cap s_l^{j_l} \cap \ldots \cap s_k^{j_k} \cap s_k^{j_k} \quad . \end{aligned}$$

Weiter gilt nach Definition von Ψ^*: $s_l^{i_l} \in \Psi_l$ und $s_l^{j_l} \in \Psi_l$. Da aber Ψ_l eine Partition von U ist, haben wir $s_l^{i_l} \cap s_l^{j_l} = \emptyset$. Damit ist das Lemma 4.4 bewiesen. □

Wir kommen nun zum Beweis des Theorems 4.3.

Beweis des Theorems:

(I) **Nullmengenausschluß:**

Gemäß der Definition von Ψ^* in (4.14) gilt $\emptyset \notin \Psi^*$.

(II) **Erzeugendensystem:**

Sei ein beliebiges $s_i^j \in \Psi_i$ gegeben. Wir bilden die Mengen S_l mit $l \neq i$ gemäß der Vorschrift: wenn $s_i^j \cap s_l^{j'} = \tilde{s}_l^{j'}$ und $\tilde{s}_l^{j'} \neq \emptyset$ so $\tilde{s}_l^{j'} \in S_l$. Dann gilt für alle $l : \cup \tilde{s}_l^{j'} = s_i^j$. Wir betrachten die Menge $S^* = \left\{ s^*_{j'_1 \ldots j'_k} \mid s^*_{j'_1 \ldots j'_k} = \bigcap_{\substack{l=1 \ldots k \\ i \neq l}} \tilde{s}_l^{j'_l} \quad , \quad \tilde{s}_l^{j'_l} \in S_l \right\} \setminus \{\emptyset\}$. Auf Grund der Bildung der $\tilde{s}_l^{j'_l} \in S_l$ als Schnittmengen mit $s_i^j \in \Psi_i$ ist es klar, daß $s^*_{j'_1 \ldots j'_k} \in \Psi^*$ gilt. Weiter haben wir $\bigcup_{s^*_{j'_1 \ldots j'_k} \in S*} s^*_{j'_1 \ldots j'_k} \subseteq s_i^j$. Es bleibt noch zu zeigen $\bigcup_{s^*_{j'_1 \ldots j'_k} \in S*} s^*_{j'_1 \ldots j'_k} \supseteq s_i^j$:

Sei ein Element $u \in s_i^j$ fixiert. Dann existiert in jeder Menge S_l genau ein Element $\tilde{s}_l^{j^*}$ so, daß $u \in \tilde{s}_l^{j^*}$, d.h. $u \in \bigcap\limits_{\substack{l=1\ldots k \\ i \neq l}} \tilde{s}_l^{j^*}$ und $\bigcap\limits_{\substack{l=1\ldots k \\ i \neq l}} \tilde{s}_l^{j^*} \in S^*$.

(III) **Partitionsbedingung:**

Nach (II) ist Ψ^* ein Erzeugendensystem für die $s_i^j \in \Psi_i$. Da die Ψ_i selbst Partitionen sind, gibt es für jedes $u \in U$ ein Element $s_{j^*}^* \in \Psi^*$ gibt mit $u \in s_{j^*}^*$. Die Disjunktheit der Elemente von Ψ^* ist in Lemma 4.4 bewiesen.

(IV) **Minimalität:**

Wir führen einen Widerspruchsbeweis: Sei $\Psi^- = \Psi^* \setminus \left\{ s_{i_1 \ldots i_k}^* \right\}$, $s_{i_1 \ldots i_k}^* \in \Psi^*$. Wir behaupten, daß $s_1^{i_1}$ auch in Ψ^- darstellbar ist.

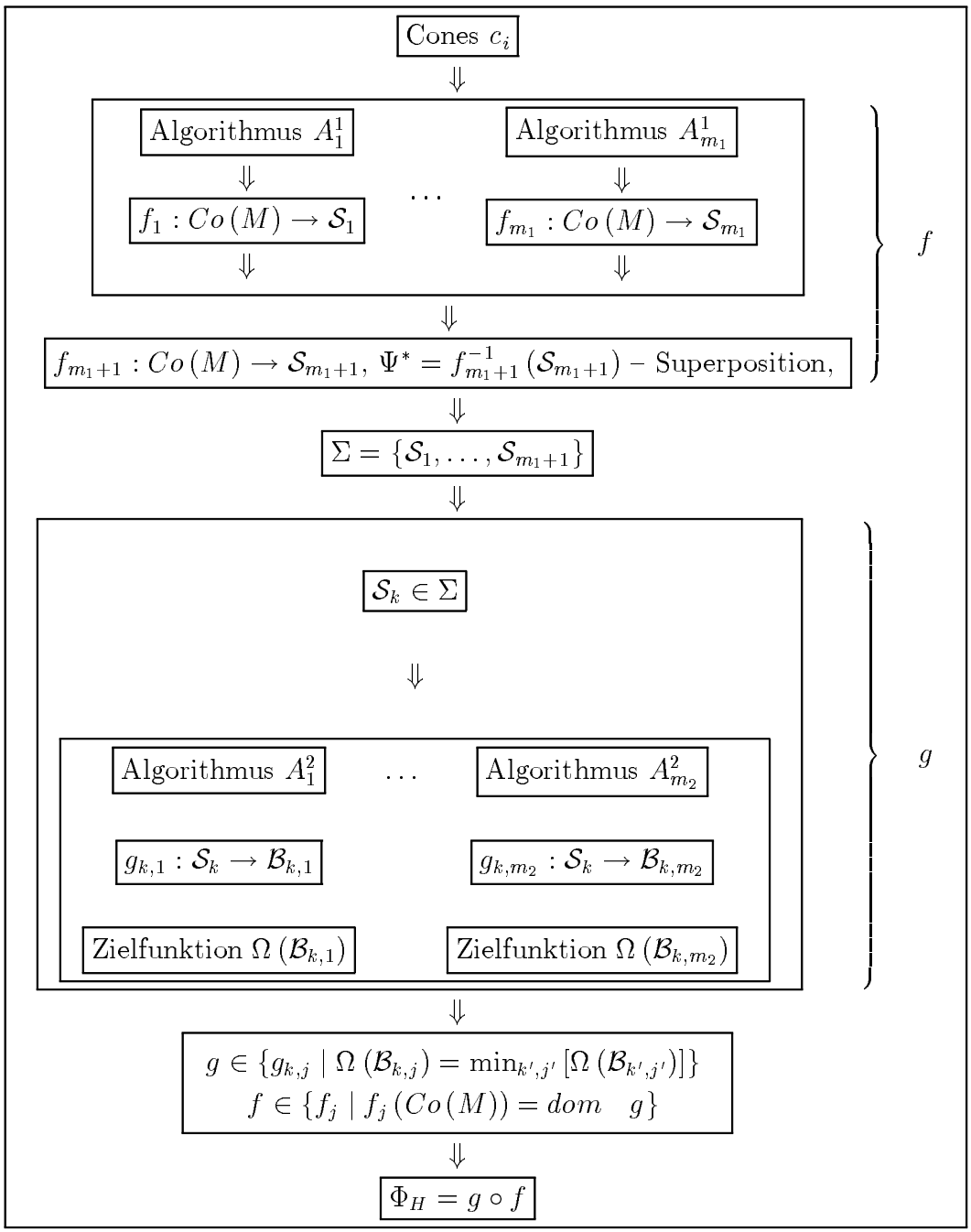
Wir nehmen an, daß $s_{l_1}^*, \ldots, s_{l_m}^*$ mit $s_{l_j}^* \in \Psi^-$ derart existieren, daß $\bigcup\limits_{j=1\ldots m} s_{l_j}^* = s_1^{i_1}$. Aus der Definition (4.14) folgt, daß $s_{i_1 \ldots i_k}^* \subseteq s_1^{i_1}$ und insbesondere daß $s_{i_1 \ldots i_k}^* \cap s_1^{i_1} \neq \emptyset$. Dann gilt aber $s_{i_1 \ldots i_k}^* \subseteq \bigcup\limits_{j=1\ldots m} s_{l_j}^*$ bzw. $s_{i_1 \ldots i_k}^* \cap \left(\bigcup\limits_{j=1\ldots m} s_{l_j}^* \right) \neq \emptyset$. Dann aber existiert ein j^* so, daß $s_{i_1 \ldots i_k}^* \cap s_{l_{j^*}}^* \neq \emptyset$. Das ist aber ein Widerspruch zu Lemma 4.4.

Damit ist auch das Theorem 4.3 bewiesen. □

Auf Grund des Theorems 4.3 läßt sich damit durch Bildung k-facher Durchschnitte aus den einzelnen Partitionen $\Psi_1, \ldots, \Psi_k$ eine Superposition gemäß (4.14) erzeugen, die die in den Partitionen verankerten Heuristiken der Partitionierungsalgorithmen in sich vereint.

Im parallelen Ansatz aus Abschnitt 4.4.2 kann diese Superposition als zusätzlich erzeugte Partition in einer Hierarchiestufe aufgenommen werden. Im Fall des

zweistufigen Ansatzes (4.11) erweitert sich damit das Schema (4.13) zu



(4.15)

Abschließend sei noch die folgende Bemerkung angefügt:

Remark 4.5 *Die triviale Partition der Cone-Menge $Co(M)$ in Einermengen stellt eine Superposition im Sinne der Def. 4.2 dar. Allerdings wird eine Superposition mit minimaler Elementanzahl, d.h. maximaler Granularität, angestrebt.*

4.5 Neu entwickelte Partitionierungsalgorithmen

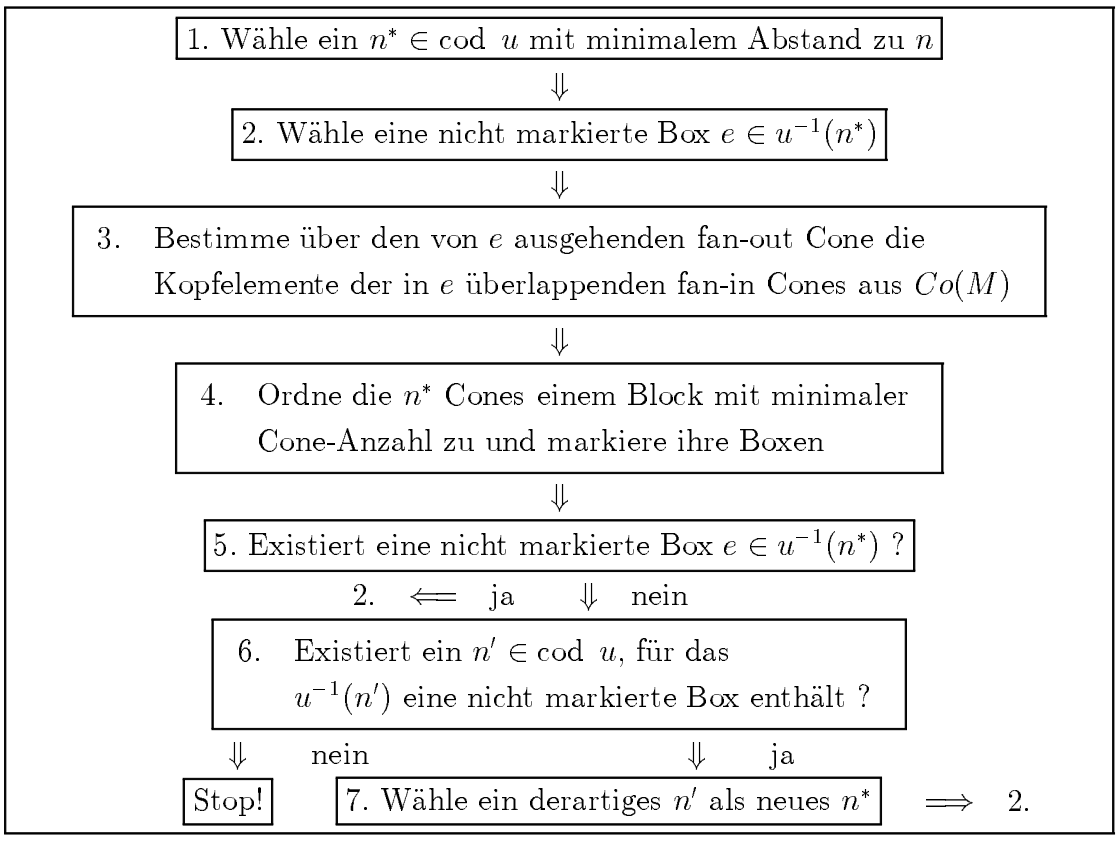
4.5.1 Der Backward – Cone – Concentration — Algorithmus (n-BCC)

Die im folgenden vorgestellte Heuristik wurde vorrangig in Hinblick auf die erste (von Cones ausgehende und zu Super-Cones führende) Stufe unseres hierarchischen Ansatzes (s. Abschnitt 4.4.1) entwickelt. Die Grundidee des Verfahrens besteht in der iterativen Zuweisung von Cone–Mengen zu Blöcken bei bevorzugter Wahl n einander überlappender Cones. n geht als Parameter (Referenz–Überlappungsgrad) in das Verfahren ein. Im Gegensatz zu dem in Abschnitt 4.5.2 dargestellten *MOCC–Algorithmus* wird beim *n-BCC–Algorithmus* die Anzahl von Boxen innerhalb betrachteter Cones bzw. Überlappungsgebiete nicht explizit berücksichtigt. Ähnlich dem auf der Fixierung von Cone-Ketten beruhenden Algorithmus von MUELLER-THUNS *et al.* [MTSDA93] wird eine gleichmäßige Verteilung der den Blöcken zugewiesenen Anzahl von Cones angestrebt. Ein wesentlicher Unterschied zum letztgenannten Verfahren besteht in der Wahl des Kriteriums der Zusammenfassung von Cones. Während MUELLER-THUNS *et al.* den Aspekt der Reduzierung erforderlicher Interprozessorkommunikation in den Vordergrund stellen, zielt *n-BCC* vorrangig auf die Verringerung der Überlappung der verschiedenen Blöcken zugewiesenen Cones.

Neben dem Referenz–Überlappungsgrad n gehen in den *n-BCC–Algorithmus* der für ein betreffendes Modell in einem Preprocessing-Schritt zu ermittelnde *boxbezogene Cone-Überlappungsgrad* $u : M_E \rightarrow \aleph$ aus Definition 3.6 und die Blockanzahl m_b als Parameter ein.

Der *n-BCC–Algorithmus* zählt ebenso wie der *MOCC–Algorithmus* (s. Abschnitt 4.5.2) zu den einfachen Partitionierungsverfahren (s. Abschnitt 4.3), die durch die Erstellung einer einzelnen initialen Partition charakterisiert sind. Es sei $\mathcal{B} = \{b_1, \ldots, b_{m_b}\}$ die betrachtete Blockmenge. Resultat des Algorithmus für ein vorgegebenes Modell M ist eine Partition Ψ_Φ von $Co(M)$ bezüglich einer (im Verlauf des Algorithmus schrittweise aufgebauten) Partitionierung $\Phi : Co(M) \longrightarrow \mathcal{B}$. Zu jedem Block b_j wird eine Conemenge $\mathcal{C}_j$ betrachtet mit $\mathcal{C}_j = \emptyset$ für $1 \leq j \leq m_b$ zu Beginn des Algorithmus. Desweiteren wird angenommen, daß initial keine der Boxen aus M_E markiert ist.

Algorithm 4.1 : ***Backward–Cone–Concentration—Algorithmus (n-BCC)***



Die schematische Algorithmendarstellung 4.1 wird im folgenden näher erläutert:

1. Zunächst wird geprüft, ob der Referenz–Überlappungsgrad n im Wertebereich von u vorkommt. Ist das der Fall, wird n als Arbeitswert n^* übernommen. Andernfalls wird als n^* das kleinste n' aus dem Wertebereich von u gewählt, für das $|n' - n|$ minimal wird.

2. Aus der Menge $u^{-1}(n^*) \subseteq M_E$ wird eine nicht markierte Box e bestimmt. Diese Wahl ist bei jedem Eintritt in Schritt 2 gewährleistet. Aufgrund der Definition von u (s. Definition 3.6) ist e in genau n^* Cones aus $Co(M)$ enthalten.

3. Mittels beschränktem *Depth First Search* werden im *fan-out-Cone* $co_O(e)$ die Kopfelemente der n^* in e überlappenden Cones fixiert. Eine schematische Darstellung dieses und des nachfolgenden Schrittes ist in Abbildung 9 wiedergegeben.

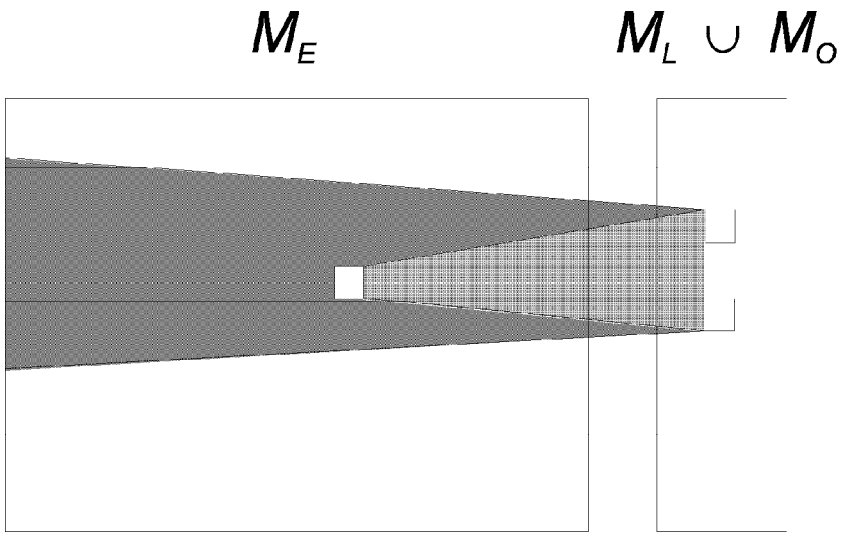


Abbildung 9: 2–BCC

4. Es wird ein j^* mit $1 \leq j^* \leq m_b$ gewählt, für welches $|\mathcal{C}_{j^*}|$ minimal ist. Danach erfolgt eine Aufnahme der Cones in die Menge $\mathcal{C}_{j^*}$, die durch die n^* in Schritt 3 fixierten Kopfelemente repräsentiert werden. Sämtliche Elemente dieser Cones, die in M_E liegen, werden mittels beschränktem *Depth First Search* ausgehend von den Kopfelementen markiert.

5. Falls noch eine bisher nicht markierte Box aus $u^{-1}(n^*)$ existiert, wird bei Schritt 2 fortgefahren, andernfalls bei Schritt 6.

6. Gibt es ein n' aus dem Wertebereich von u derart, daß $u^{-1}(n')$ eine nicht markierte Box enthält, wird bei Schritt 7 fortgesetzt. Andernfalls sind sämtliche nicht nur ein Kopfelement enthaltende Cones bereits den Mengen $\mathcal{C}_j$ zugewiesen. Falls einelementige Cones existieren, werden diese sukzessive zur Menge $\mathcal{C}_j$ mit der jeweils kleinsten Anzahl von Cones zugeordnet und die Abarbeitung des Algorithmus ist beendet.

7. Existiert ein $n' < n^*$ aus dem Wertebereich von u derart, daß $u^{-1}(n')$ eine nicht markierte Box enthält, wird das größte derartige n' als neues n^* gewählt. Andernfalls wird das kleinste n' mit $n' > n^*$, für welches die bezüglich u^{-1} formulierte Bedingung gilt, zum neuen Wert von n^*. Es wird bei Schritt 2 fortgesetzt.

Bemerkungen:

- Eine erste *n-BCC*–Implementierung liegt vor. Diesbezügliche experimentelle Ergebnisse sind in Abschnitt 4.6 dargelegt.
- Untersuchungen zur Wahl des Parameters n in Abhängigkeit vom zugrundeliegenden Modell M stehen am Anfang. Als möglicher Ansatzpunkt in dieser Richtung erscheint die Ausnutzung von Modelleigenschaften, die mit der Bitbreite parallel laufender Datenkanäle in Zusammenhang stehen.
- Im Rahmen der Schritte 2 (Wahl einer nicht markierten Box) und 7 (Wahl des aktuellen Referenz-Überlappungsgrades) sind Variationen der momentan implementierten Strategie vorgesehen.

4.5.2 Der Minimum – Overlap – Cone – Cluster — Algorithmus (MOCC)

Der *MOCC–Algorithmus* baut sukzessive eine Partition aus der jeweiligen Grundmenge zur Partitionierung unter Berücksichtigung modellspezifischer Informationen auf. Beim *MOCC–Algorithmus* werden Modellinformationen in Form des Überlappungs–Hypergraphen *GU* (s. Def. 3.9) verwendet. Die initiale Partition kann im weiteren durch iterative Verfahren bezüglich einer Zielfunktion Ω optimiert werden.

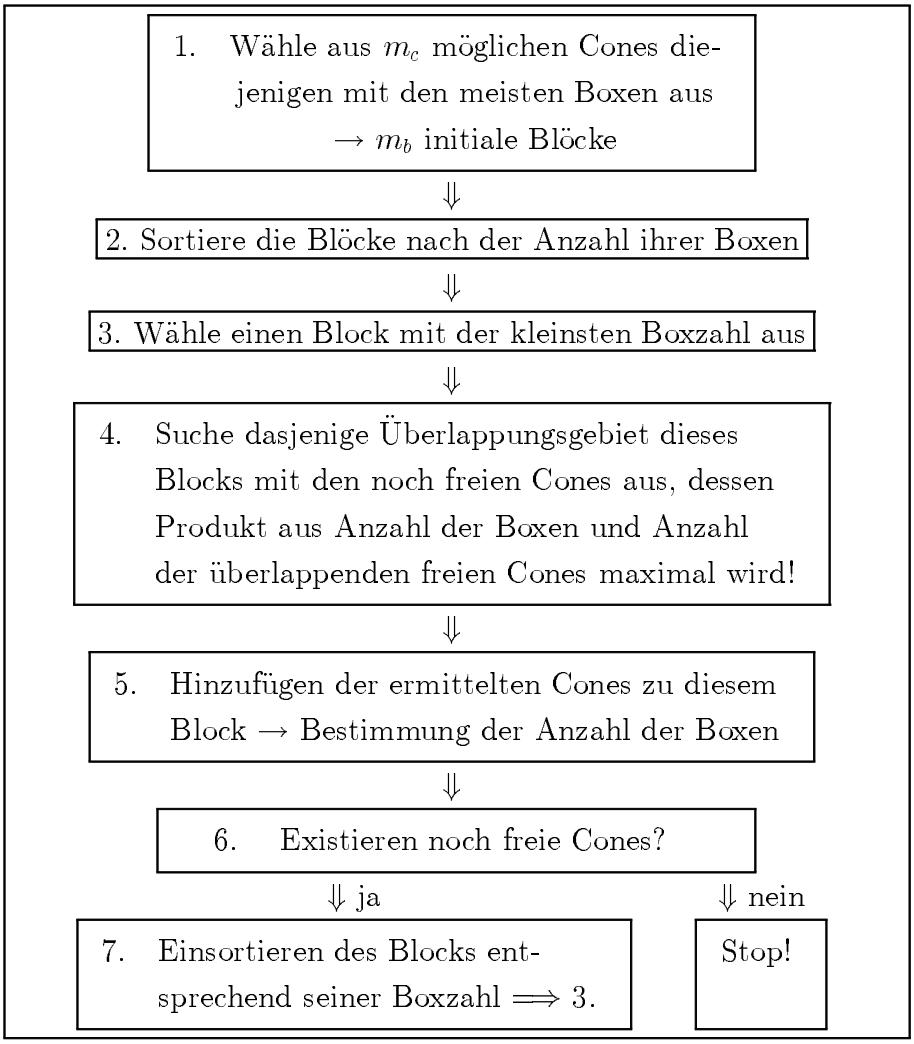
Der *MOCC–Algorithmus* ist im Unterschied zum *n-BCC–Algorithmus* primär für die *zweite Stufe in der Partitionierungshierarchie* (s. Def. 4.1), d.h. für die Partitionierung einer Super-Cone–Menge $V_2 = \mathcal{S}$, aber auch für die erste Stufe und damit für die Partitionierung der Cone-Menge $V_1 = U = Co(M)$, geeignet (vgl. Abschnitt 4.4.1).

Zur einheitlichen Gestaltung der nachfolgenden Beschreibung des Algorithmus werden wir analog zum *n-BCC–Algorithmus* eine Partitionierung der Cone-Menge $Co(M)$ ($|Co(M)| = m_c$) in die Block-Menge $\mathcal{B}$ ($|\mathcal{B}| = m_b$) zugrunde legen.

Mit dem *MOCC–Algorithmus* verfolgen wir das *Ziel*, auf der Ebene der Überlappungsstruktur im Hardware-Modell (Überlappungs–Hypergraph *GU*) innerhalb einer Partition Blöcke derart aufzubauen, daß die Knoten im Hypergraphen zu Teilgraphen zusammengefaßt werden, die durch maximal viele Hyperkanten mit höchstem Gewicht miteinander verknüpft sind. Implizit entspricht das der Minimierung der Zahl der geschnittenen Hyperkanten unter Berücksichtigung ihres Gewichtes. Bei diesem Vorgehen werden die Cones zusammengefaßt, die über eine hohe Anzahl von

(durch die beteiligten Cones gemeinsam erfaßten) Boxen verfügen. Damit ist man in der Lage, Einfluß in Richtung einer möglichst minimalen Mehrfachauswertung von Boxen zu nehmen. Gleichzeitig wird im Verlauf des sukzessiven Auffüllens der Blöcke mit Cones eine gleichmäßige Verteilung der Boxen auf die Blöcke (Lastausgleich) angestrebt.

Algorithm 4.2 : ***Minimum–Overlap–Cone–Cluster—Algorithmus (MOCC)***



Im folgenden werden wir den *MOCC–Algorithmus* 4.2 näher erläutern:

1. Der erste Schritt im Algorithmus ist dadurch gekennzeichnet, daß aus der Cone-Menge $Co(M)$ genau m_b Cones ausgewählt werden. Die Auswahl geschieht nach

der Anzahl der vom Cone c erfaßten Boxen $|c|$. In diesem ersten Schritt werden die Cones ausgewählt, die über die meisten Boxen verfügen.

2. Im zweiten Schritt werden die Blöcke entsprechend der in ihnen gemäß der zugehörigen Cones zusammengefaßten Anzahl von Boxen, d.h. der Last W_i, $1 \leq i \leq m_b$ (s. Gleichung (3.10)) sortiert.

3. In diesem Schritt wird erstmalig der Zyklus des sukzessiven Aufbaus der Blöcke bis zur vollständigen Verteilung der Cones, d.h. der Bildung einer Partition Ψ_Φ, $\Phi : Co(M) \rightarrow \mathcal{B}$, erreicht. Vor dieser Stufe sind jedem Block b_i eindeutig über seiner Cone-Menge $\mathcal{C}_i$ Cones c_j^i zugeordnet. Die Blöcke sind entsprechend der Anzahl der ihnen zugehörigen Boxen (der Lasten W_i) sortiert. Es folgt die Auswahl eines Blockes b_j, für den gilt: $W_j = \min_{1 \leq i \leq m_b} W_i$.

4. Die Cone-Menge $Co(M)$ ist eindeutig in zwei disjunkte Mengen zerlegbar: $\mathcal{C}_B$ und $\mathcal{C}_{frei}$ mit $\mathcal{C}_B = \{c | \exists i (c \in \mathcal{C}_i \wedge 1 \leq i \leq m_b)\}$ und $\mathcal{C}_{frei} = Co(M) \setminus \mathcal{C}_B$. Ausgehend von dem in 3. ausgewählten Block b_j werden die Überlappungsgebiete $ovr(\mathcal{C})$ mit allen möglichen Cone-Mengen $\mathcal{C} \in P^*(Co(M))$ (s. Gl. (3.4)) in die Betrachtung einbezogen, für die gilt: $\mathcal{C} \cap \mathcal{C}_j \neq \emptyset \wedge \mathcal{C} \cap \mathcal{C}_{frei} \neq \emptyset \wedge |ovr(\mathcal{C})| \neq 0$. Im folgenden ist es sinnvoll, Überlappungsgebiete zu größeren Boxmengen $v_{\mathcal{C}'}$ mit $\emptyset \subset \mathcal{C}' \subseteq \mathcal{C}_{frei}$ zusammenzufassen: $v_{\mathcal{C}'} = \bigcup_{\mathcal{C} \cap \mathcal{C}_{frei} = \mathcal{C}' \wedge \mathcal{C} \cap \mathcal{C}_j \neq \emptyset} ovr(\mathcal{C})$, von denen uns im weiteren nur die $v_{\mathcal{C}'} \neq \emptyset$ interessieren.

   Eine mögliche Realisierung einer derartigen Überdeckungsstruktur am Beispiel eines Blockes b_j mit $j = 1$ ist in Abbildung 10 dargestellt. Drei der vier dargestellten Cones c_1, c_2 und c_3 überlappen mit dem, im dritten Schritt ermittelten, kleinsten Block b_1 und bilden fünf disjunkte nichtleere Überlappungsgebiete $v_{\mathcal{C}'}$.

   Unter diesen Überlappungsgebieten wird ein $v_{\mathcal{C}^*}$ ermittelt, für welches das Produkt aus der Anzahl der in diesem Überlappungsgebiet befindlichen Boxen $|v_{\mathcal{C}^*}|$ und der Anzahl der dieses Überlappungsgebiet überdeckenden freien Cones $|\mathcal{C}^*|$ maximal ist: $|v_{\mathcal{C}^*}|\,|\mathcal{C}^*| = \max_{\emptyset \subset \mathcal{C}' \subseteq \mathcal{C}_{frei}} (|v_{\mathcal{C}'}|\,|\mathcal{C}'|)$.

5. Ausgehend vom ermittelten zusammengefaßten Überlappungsgebiet $v_{\mathcal{C}^*}$ werden genau die freien Cones $c \in \mathcal{C}^*$ dem Block b_j hinzugefügt. Es wird die geänderte Last W_j des Blockes b_j ermittelt.

   Es sei im Beispiel der Abbildung 10 das zusammengefaßte Überlappungsgebiet $v_{\{c_1,c_2\}}$ dasjenige mit dem größten Wert $|v_{\mathcal{C}^*}|\,|\mathcal{C}^*| = \left|v_{\{c_1,c_2\}}\right|\,|\{c_1,c_2\}| = 2\left|v_{\{c_1,c_2\}}\right|$.

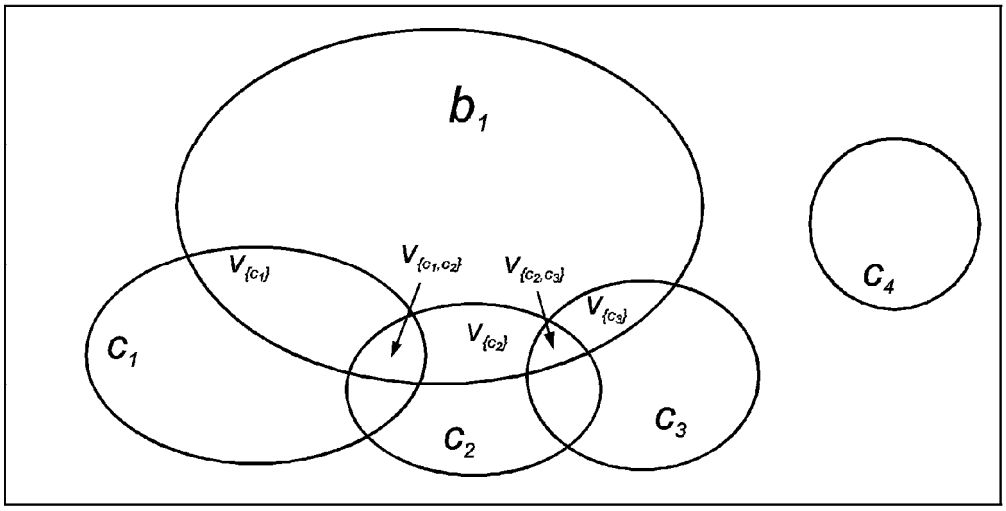


Abbildung 10: Beispiel für eine Überlappungsstruktur vor einer Zuordnung freier Cones c_i zum Block b_1

Davon ausgehend werden in diesem Schritt die Cones c_1 und c_2 zum Block b_1 dazugenommen (s. Abbildung 11) und die Last W_1 des Blockes neu berechnet.

6. Im sechsten Schritt wird getestet, ob es noch weitere freie Cones gibt ($|\mathcal{C}_{frei}| \neq 0$?). Existieren keine weiteren freien Cones, wird der Zyklus abgebrochen, die Partition Ψ_Φ , $\Phi : Co(M) \rightarrow \mathcal{B}$, ist vollständig.

7. Gibt es noch freie Cones ($|\mathcal{C}_{frei}| \neq 0$), wird der Block b_j entsprechend seiner neuen Last W_j in die Blockmenge $\mathcal{B}$ neu einsortiert und es erfolgt im Zyklus ein Sprung zum dritten Schritt.

Bemerkungen:

- Der erste Schritt im Algorithmus läßt andere Varianten zu, wie z.B. die zufällige Auswahl von m_b Cones und deren Verteilung auf die Blöcke.

- Im vierten Schritt ist eine Entscheidung für die Hinzunahme neuer Cones aufgrund anderer Kriterien als den oben beschriebenen möglich. So ist es denkbar, Überlappungsgebiete nicht zu größeren Einheiten zusammenzufassen, sondern einzeln einer Wertung zu unterziehen.

- Weiterhin ist es sinnvoll, im vierten Schritt eine obere Schranke für die Größe der Blöcke bezüglich der Lasten bei der Hinzunahme weiterer freier Cones zu

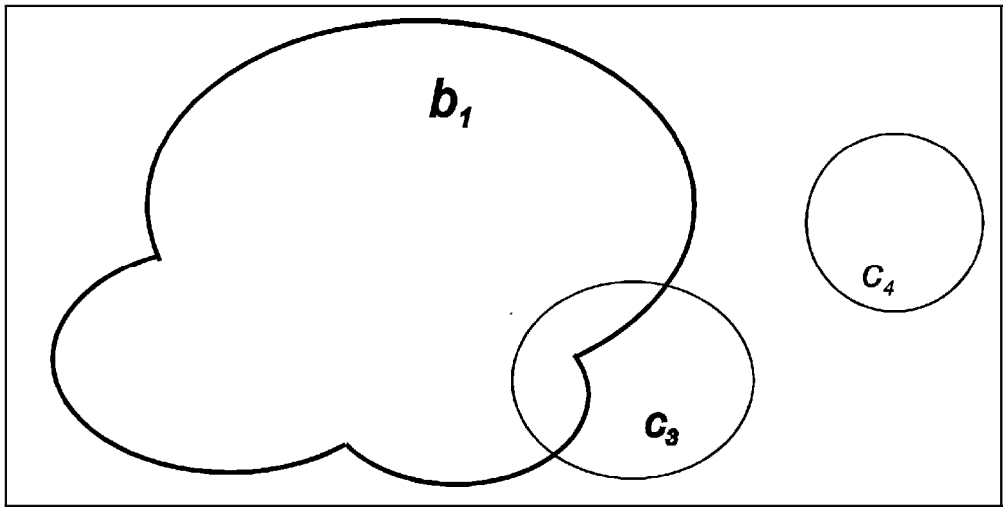


Abbildung 11: Bildung des erweiterten Blockes b_1 nach Auswahl des Überlappungsgebietes v_2 durch Hinzufügen der Cones c_1 und c_2

setzen. Das kann dazu führen, daß nicht alle der vom Algorithmus ermittelten freien Cones auch tatsächlich zum Block hinzugefügt werden.

- Tritt im vierten Schritt der Fall ein, daß kein freier Cone mit dem Block b_j überlappt, d.h. $\max_{\emptyset \subset \mathcal{C}' \subseteq \mathcal{C}_{frei}} (|v_{\mathcal{C}'}| \, |\mathcal{C}'|) = 0$, wird der größte freie Cone mit den meisten Boxen dem Block b_j zugeordnet.

4.6 Erste experimentelle Ergebnisse

Für die experimentellen Untersuchungen zur Modellstatistik und im weiteren zur Partitionierung standen bisher zwei Hardware-Modelle signifikanter Größe von IBM zur Verfügung, die wir im folgenden mit M_1 und M_2 bezeichnen.

Vor der Partitionierung wurden die Modelle hinsichtlich ihrer charakteristischen Eigenschaften und insbesondere bezüglich ihrer Cones untersucht. In der folgenden Tabelle werden für die beiden Modelle die grundlegenden Größen dargestellt:

Modell	M_1	M_2
$\lvert Co(M_i)\rvert$	2070	21 705
W_{seq}	16 398	180 211

Bei der Charakterisierung der Modelle bezüglich der Cone-Menge als Ausgangsmenge zur Partitionierung sind folgende Daten von Interesse:

- Anzahl (prozentualer Anteil) der Boxen, die von genau u Cones bzw. von maximal u Cones überdeckt werden:

u	genau 1	max. 1	genau 2	max. 2	genau 3	max. 3	genau 4	max. 4
M_1	58%	58%	8%	66%	4.4%	70.4%	2.3%	72.7%
M_2	44.8%	44.8%	5.4%	50.2%	3.1%	53.3%	4.7%	58%

- Ein wesentlicher Anteil der Boxen wird nur von einem Cone erfaßt, kann also genau einem Cone und damit auch genau einem Block bei der Partitionierung zugeordnet werden. Dieser Anteil von Boxen wird also in jedem Fall nur auf einem Prozessor ausgewertet. Für alle anderen Boxen muß mit einer zwei- bzw. mehrfachen Auswertung auf verschiedenen Prozessoren im parallelen Simulationslauf gerechnet werden.

- Die folgenden zwei Abbildungen 12 und 13 stellen für die beiden Modelle M_1 und M_2 die Anzahl von Boxen dar, die den gleichen boxbezogenen Cone-Überlappungsgrad (s. Def. 3.6) haben. Die Abbildung 12 bezieht sich auf das Modell M_1 mit verschiedenen Maßstäben bezüglich des Cone-Überlappungsgrades u. Abbildung 13 stellt die Anzahl der Boxen im Modell M_2 dar, die über den Cone-Überlappungsgrad u verfügen.Es zeigt sich, daß es eine hohe Zahl von Boxen gibt, die gleichzeitig von einer großen Menge von Cones überdeckt wird. Für bestimmte Werte von u gibt es signifikante Maxima bezüglich der Anzahl von Boxen. Das ist z.B. bei $u = 16$ der Fall, was mit der Bitbreite parallel laufender Datenkanäle in Zusammenhang stehen kann. Solche Unterstrukturen sollten möglichst in einem Block zusammengefaßt und nicht partitioniert werden.

- Alternativ zu den vorangegangenen Darstellungen sind die Abbildungen 14 und 15 mit der Anzahl von Cones, die eine bestimmte Zahl von Boxen pro Cone $|c|$ einschließen.

  Hieraus ist zu ersehen, daß es eine große Anzahl von Cones gibt, die nur über wenige Boxen verfügen und somit im wesentlichen unter dem Gesichtspunkt des

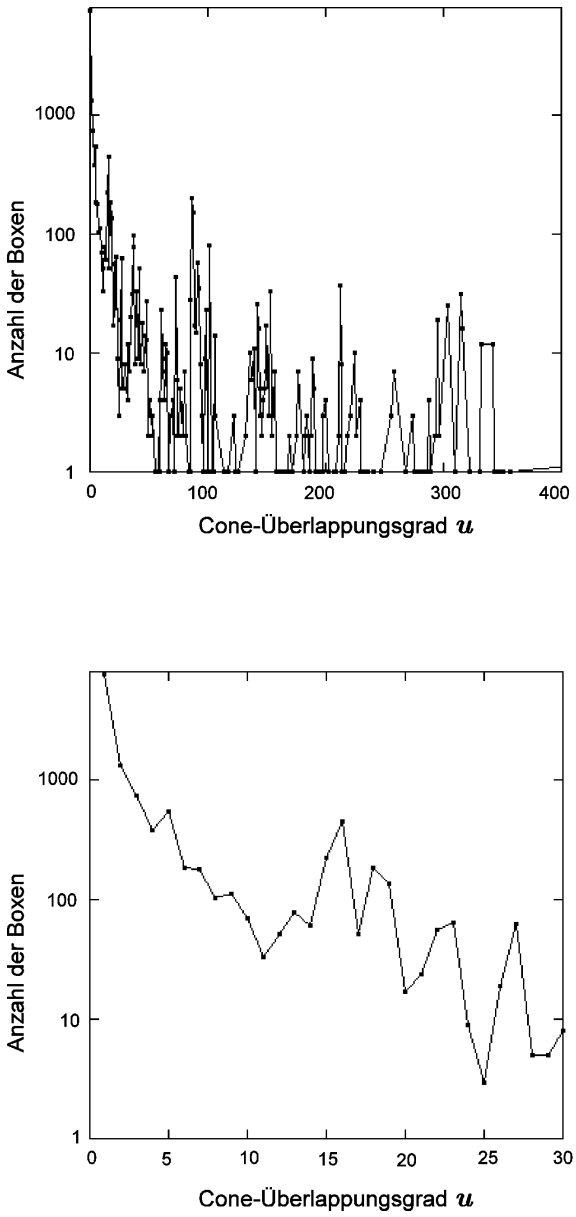


Abbildung 12: Anzahl der Boxen mit dem Cone-Überlappungsgrad u im Modell M_1 (im unteren Bildteil ist ein Ausschnitt dargestellt)

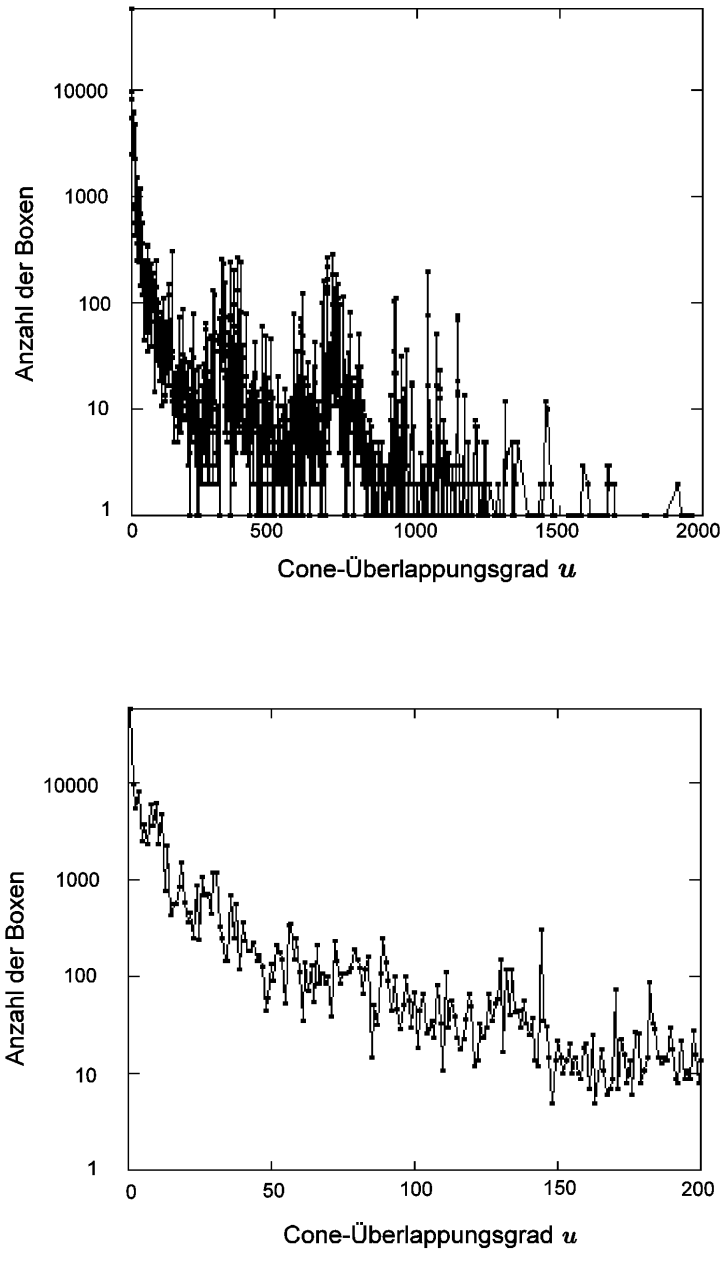


Abbildung 13: Anzahl der Boxen mit dem Cone-Überlappungsgrad u im Modell M_2 (im unteren Bildteil ist ein Ausschnitt dargestellt)

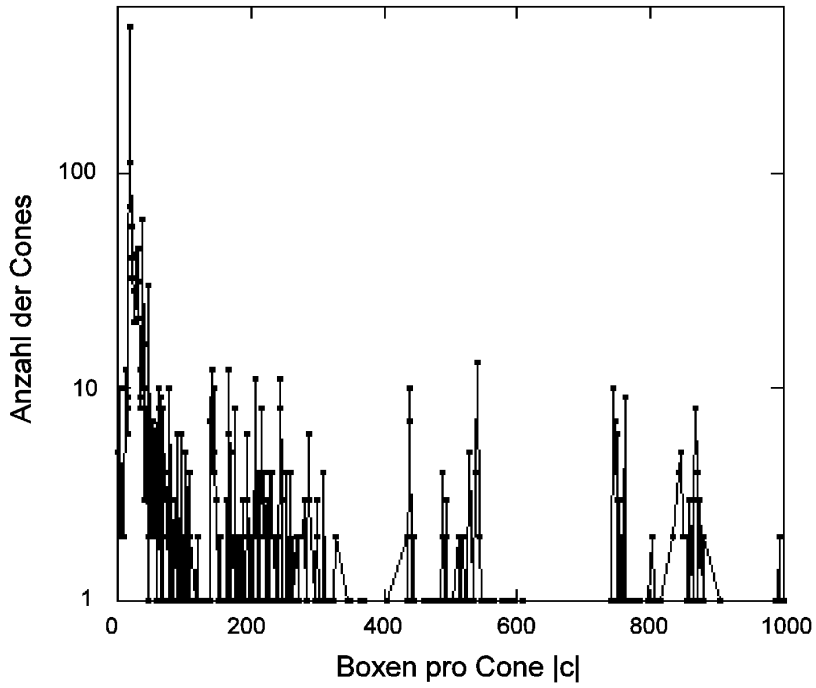


Abbildung 14: Anzahl der Cones mit einer bestimmten Anzahl von Boxen pro Cone $|c|$ für das Modell M_1

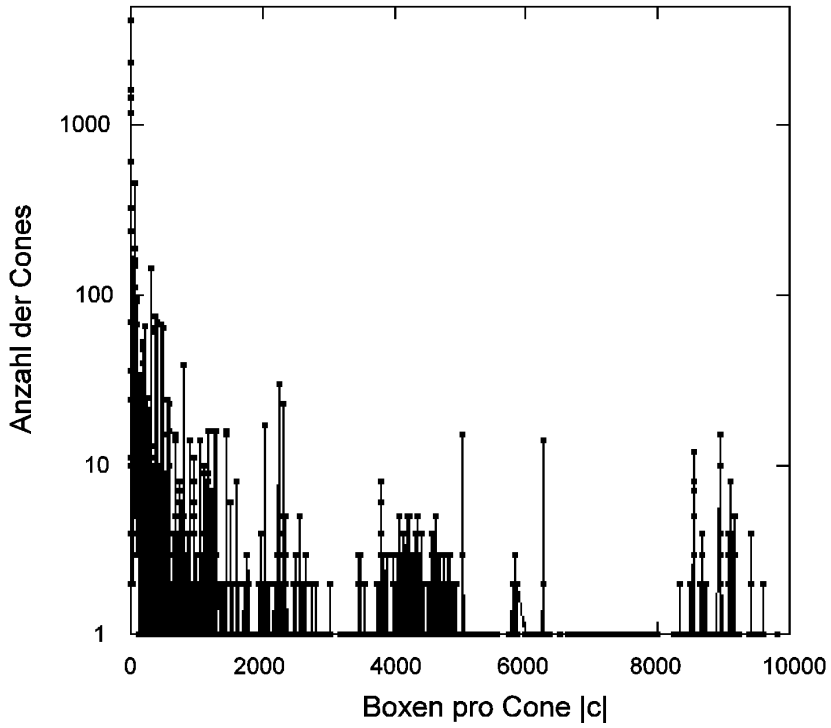


Abbildung 15: Anzahl der Cones mit einer bestimmten Anzahl von Boxen pro Cone $|c|$ für das Modell M_2

Lastausgleiches verteilt werden sollten. Dagegen erscheint es sinnvoll, Cones mit vielen Boxen bereits frühzeitig beim sukzessiven Auffüllen der Blöcke zu verteilen.

Im zweiten Teil dieses Abschnittes werden erste Ergebnisse für die Partitionierung der Modelle vorgestellt. Partitioniert wird bei den hier vorliegenden Elementzahlen ohne die Zwischenstufe der Super–Cones, d.h. es erfolgt eine direkte Partitionierung der Cone-Menge $CO(M_i)$ in die Blockmenge $\mathcal{B}$. Zwei Algorithmen sind implementiert und bereits angewendet worden: der Algorithmus A_1 von MUELLER-THUNS (MT–Algorithmus) und der n-BCC–Algorithmus A_2. Den n-BCC–Algorithmus versehen wir an der Stelle mit dem Argument n in der Form $A_2(n)$. Einer Partition des Algorithmus A_1 werden jeweils verschiedene Partitionen von $A_2(n)$ mit unterschiedlichem n gegenübergestellt. Wir variieren die Anzahl der Blöcke m_b. Weiterhin werden alle Werte relativ zur sequentiellen Last W_{seq} normiert, um von der konkreten Modellgröße zu abstrahieren.

Folgende Größen werden für die Partitionen ausgewertet:

1. normierte Blocklast (s. Gl. (3.10)):

$$W_i' = W_i / W_{seq}$$

2. Replikationsrate (Rate der Mehrfachauswertung von Boxen auf verschiedenen Blöcken):

$$r = \sum_{i=1}^{m_b} W_i' = m_b \, \overline{W}'_{par}$$

($\overline{W}'_{par} = \overline{W}_{par} / W_{seq}$ – s. Gl. (4.4))

3. normierte Standardabweichung (Maß für den Lastausgleich):

$$\sigma' = \sigma / W_{seq} = \frac{1}{W_{seq}} \sqrt{\frac{1}{m_b} \sum_{i=1}^{m_b} \left(W_i - \overline{W}_{par} \right)^2}$$

4. normierte Zielfunktion:

   (a) nach MANJIKIAN (vgl. Gl. (4.5)):

$$\Omega'_{Man} = \Omega_{Man} / W_{seq} = \sum_{i=1}^{m_b} \left| W_i' - \frac{1}{m_b} \right|$$

(b) parametrisiert (vgl. Gl. (4.6)): $\alpha_1 = \alpha_2 = 0.5$

$$\Omega'_\alpha = \Omega_\alpha / W_{seq} = \frac{1}{2}\left[\left(\overline{W}'_{par} - \frac{1}{m_b}\right) + \sigma'\right] = \frac{1}{2}\left[\left(\frac{r-1}{m_b}\right) + \sigma'\right]$$

(c) Maximallast (vgl Gl. (4.7)):

$$\Omega'_{W_{\max}} = \Omega_{W_{\max}} / W_{seq} = \max_{1 \le i \le m_b} W'_i$$

In den folgenden Tabellen werden Eigenschaften durch den n-BCC– bzw. MT–Algorithmus erzeugter Partitionen dargestellt:

1. Modell M_1; $m_b = 4$

Algorithmus	A_1	$A_2(2)$	$A_2(4)$	$A_2(8)$	$A_2(16)$	$A_2(32)$
W'_1	0.311	0.451	<u>0.482</u>	<u>0.481</u>	0.352	0.362
W'_2	0.384	0.431	0.473	0.422	<u>0.431</u>	<u>0.403</u>
W'_3	0.382	0.472	0.438	0.427	0.360	0.402
W'_4	<u>0.525</u>	<u>0.528</u>	0.425	0.423	0.409	0.395
r	1.598	1.886	1.818	1.748	1.570	1.563
σ'	0.078	0.030	0.023	0.025	0.032	0.019
Ω'_{Man}	0.598	0.886	0.818	0.748	0.570	0.563
Ω'_α	0.114	0.126	0.114	0.106	0.087	0.080
$\Omega'_{W_{\max}}$	0.525	0.528	0.482	0.481	0.431	0.403

2. Modell M_1; $m_b = 8$

Algorithmus	A_1	$A_2(2)$	$A_2(4)$	$A_2(8)$	$A_2(16)$	$A_2(32)$
W'_1	0.174	0.301	0.345	0.286	0.255	0.238
W'_2	0.218	0.333	0.318	<u>0.362</u>	0.248	0.265
W'_3	0.225	0.378	0.309	0.285	0.272	0.252
W'_4	0.243	0.340	0.291	0.309	<u>0.281</u>	0.231
W'_5	0.343	0.318	0.330	0.305	0.243	0.260
W'_6	0.201	0.308	0.342	0.318	0.273	0.244
W'_7	0.258	0.334	<u>0.372</u>	0.281	0.274	<u>0.292</u>
W'_8	<u>0.363</u>	<u>0.382</u>	0.328	0.322	0.278	0.239
r	2.042	2.719	2.632	2.478	2.133	2.038
σ'	0.060	0.030	0.024	0.025	0.013	0.019
Ω'_{Man}	1.042	1.719	1.632	1.478	1.133	1.038
Ω'_{α}	0.095	0.122	0.114	0.105	0.077	0.074
$\Omega'_{W_{\max}}$	0.363	0.382	0.372	0.362	0.281	0.292

3. Modell M_1; $m_b = 16$

Algorithmus	A_1	$A_2(2)$	$A_2(4)$	$A_2(8)$	$A_2(16)$	$A_2(32)$
r	2.682	3.971	3.781	3.518	2.912	2.743
σ'	0.036	0.023	0.016	0.019	0.017	0.026
Ω'_{Man}	1.682	2.971	2.781	2.518	1.912	1.743
Ω'_{α}	0.071	0.104	0.095	0.088	0.068	0.067
$\Omega'_{W_{\max}}$	0.243	0.282	0.268	0.255	0.214	0.204

- Beim n-BCC–Algorithmus sind für alle Blockzahlen mit wachsendem Referenz-Überlappungsgrad n zunehmend bessere Resultate zu verzeichnen (insbesondere hinsichtlich der Mehrfachauswertung).

- Der n-BCC–Algorithmus liefert für das untersuchte Modell für alle Testläufe ausgeglichenere Blocklasten als der MT–Algorithmus.

- Im Vergleich zum MT–Algorithmus sind die Werte für die Mehrfachauswertung beim n-BCC–Algorithmus für kleinere n schlechter, aber für größere n tendentiell besser.

- Die maximalen Blocklasten $W_{\max}$, die die *speed-up*–Werte bestimmen, sind aufgrund der ausgeglicheneren Blocklasten für den n-BCC–Algorithmus günstiger.

Literatur

[BDBK+90] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. Technical Report TR-90-023, Berkeley, CA, June 1990.

[Bry88] R. E. Bryant. Data parallel switch-level simulation. In *Proc. IEEE International Conference on Computer-Aided Design, ICCAD-88*, pages 354–357. IEEE–Press, 1988.

[DEFH93] R. Der, H. Englisch, M. Funke, and M. Herrmann. Time Series Prediction Using Hierarchical Self-Organized Feature Maps. *Neural Network World*, (3):699–703, 1993.

[FM82] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Automata Conference, ACM/IEEE*, pages 175–181, 1982.

[GS93] Markus H. Gross and F. Seibert. Visualization of multidimensional image data sets using a neural network. *Visual Computer*, 10:145–159, 1993.

[HHK95] L. Hagen, J.-H. Huang, and A. B. Kahng. Quantified Suboptimality of VLSI Layout Heuristics. In *Proc. of the Design Automation Conference DAC'95*, 1995.

[Hil81] W. Hilberg. Partitionierung mit Hilfe der Verbindungsmatrix. *ntz Archiv*, 3(3):57–62, 1981.

[JGD87] L. H. Jamieson, D. B. Gannon, and R. J. Douglass. *The Characteristic of Parallel Algorithms*. MIT Press, 1987.

[JJ94] M. I. Jordan and R. A. Jacobs. Hierarchical Mixture of Experts and the EM Algorithm. In P. Morasso, editor, *Proc. ICANN'94*, pages 479–486. Springer, 1994.

[JJNH91] R. A. Jacobs, M. A. Jordan, S. J. Nowlan, and G. E. Hinten. Adaptive Mixture of Local Experts. *Neural Computation*, (3):79 – 87, 1991.

[Kar90] Richard M. Karp. An introduction to randomized algorithms. Technical Report TR-90-024, Berkeley, CA, June 1990.

[Kar91] Richard M. Karp. Parallel combinatorial computing. Technical Report TR-91-006, Berkeley, CA, January 1991.

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(2):291–307, 1970.

[KMP95] J. Kohlmorgen, K.-R. Müller, and K. Pawelzik. Improving Short-Term Prediction with Competing Experts. In *Proc. ICANN'95*, Paris 1995, 1995.

[Len90] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Teubner-Verlag Stuttgart and JOHN WILEY & SONS, 1990.

[Man92] N. Manjikian. High performance parallel logic simulation on a network of workstations. Technical Report CCNG T-220, Department of Electrical and Computer Engineering and Computer Communications Network Group, University of Waterloo, 1992.

[Mar93] P. Marwedel. *Synthese und Simulation von VLSI–Systemen.* Carl Hanser Verlag, 1993.

[Mei93] G. Meister. A survey on parallel logic simulation. Report 14/1993, Universität Saarbrücken (Germany), SFB 124, 1993.

[MTSDA93] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham. VLSI Logic and Fault Simulation on General Purpose Parallel Computers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12:446–460, 1993.

[Pos89] H.-U. Post. *Entwurf und Technologie hochintegrierter Schaltungen.* Teubner-Verlag Stuttgart, 1989.

[RMS92] Helge Ritter, Thomas Martinetz, and Klaus Schulten. *Neural Computation and Self-Organizing Maps: An Introduction.* Addison-Wesley, Reading, MA, 1992.

[San89] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.

[SB93] C. Sporrer and H. Bauer. Corolla partitioning for distributed logic simulation of VLSI-circuits. In *Proc. Of the 1992 SCS Western Simulation Multiconference on Parallel and Distributed Simulation , PADS'92*, pages 205–209, 1993.

[Spo95] C. Sporrer. *Verfahren zur Schaltungspartitionierung für die parallele Logiksimulation.* Verlag Shaker Aachen, 1995.

[SUM87] S. P. Smith, B. Underwood, and M. R. Mercer. An analysis of several approaches to circuit partitioning for parallel logic simulation. In *Proceedings IEEE International Conference on Computer Design (ICCD)*, pages 664–667, 1987.

[TU94] C. C. Tung and C. Ussery. Face off : Cycle–based vs. event driven simulation. *Computer Design's ASIC DESIGN*, pages A14–A17, 1994.

[Vil95] Th. Villmann. *Topologieerhaltung in selbstorgansierenden neuronalen Merkmalskarten.* Diss., eingereicht, Universität Leipzig, 1995.

[VMB+94] A. Verikas, K. Malmqvist, M. Bachauskene, L. Bergman, and K. Nilsson. HIERARCHCAL neural networkfor COLOR classification. In *Proc. ICNN'94, Int. Conf. on Neural Networks*, pages 2938–2941, Piscataway, NJ, 1994. IEEE Service Center.