Datenverteilung als Konfigurationsproblem

Jörn Eisenbiegler

23.5.1996

1 Einleitung

Das zentrale Problem beim Entwurf verteilter Programme bzw. dem Übersetzen von PRAM-Programmen auf parallele (insbesondere verteilte) Systeme ist die optimale Verteilung der Daten. Eine optimale Verteilung für ein nur bedingt realistisches Maschinenmodell, das nur Latenzzeiten berücksichtigt, zu finden ist jedoch bereits NP-hart ([Papadimitriou und Yannakakis, 1990]). Löwe und Zimmermann zeigen in [Löwe und Zimmermann, 1995b,a], daß es für eine große Klasse von Anwendungen möglich ist, in polynomieller Zeit eine suboptimale Lösung zu finden, die nur um maximal einen kleinen konstanten Faktor schlechter ist als die optimale Lösung. Dieser Ansatz benötigt jedoch polynomiell viel Zeit in Bezug auf die im zu übersetzenden Programm auszuführenden Anweisungen. Die Laufzeit des Verteilungsalgorithmus beträgt also $O(W_{\mathrm{PRAM}}^k)$, wobei $k \in \mathbb{N}$ fest und W_{PRAM} die Arbeit des PRAM-Programmes ist.

Ein anderer Ansatz für die Verteilung der Daten auf Prozessoren findet man bei Philippsen. Er verteilt die Daten statisch anhand verschiedener Arten von Präferenzen. Eine mögliche Umverteilung wird jedoch nicht untersucht [Philippsen, 1994].

In dieser Arbeit soll die Verteilung der Daten auf ein Konfigurationsproblem zurückgeführt werden. Dieses Konfigurationsproblem ist nun in der Zeit $O(|A| \cdot T_{\mathrm{PRAM}})$ optimal lösbar, wobei A für die (i.a. exponentiell große) Menge der Verteilungsvarianten und T_{PRAM} für die PRAM-Laufzeit des Programmes steht. Dieses Verfahren erlaubt es, sowohl in der Übertragungs- als auch in der Konfigurationsphase z.B. durch die Einschränkung der Verteilungsalternativen oder den Verzicht auf das Ausrollen von Schleifen Laufzeiteinsparungen auf Kosten der Optimalität vorzunehmen. Mit dieser Methode können auf einfache Weise Bibliotheken, die verschiedene Algorithmen für verschiedene Ein-/Ausgangsverteilungen zur Verfügung stellen, eingebunden werden.

2 Verteilung als Konfiguration

Es ist i.a. nicht optimal, eine Verteilung für die Datenfelder eines Programmes statisch für die gesamte Programmlaufzeit anzugeben. Zwischen allen Programmschritten muß eventuell eine neue Verteilung der Daten in Betracht gezogen werden. Ein dazu analoges Problem ist die optimale Konifguration von Programmen [Moldenhauer, 1996]. Hierbei wird eine optimale Auswahl der Repräsentation von Daten, d.h. Datenstrukturen und Algorithmen auf diesen, zu jedem Zetpunkt des Programmablaufes gesucht.

Für die Lösung des Konfiguartionsproblemes wird ein gerichteter azyklischer Graph betrachtet. Für jeden Programmschritt gibt es für jede mögliche Representation der Datenstrukturen einen Knoten und zwischen zwei Programmschritten für jeden möglichen Representationswechsel eine Kante. Die Knoten werden mit den Kosten für die Ausführung des Programmschrittes mit der jeweiligen Representation und die Kanten mit den Kosten für den Representationswechsel gewichtet. Die optimale Konfiguration des Programmes ist nun der küzesten Pfad durch diesen Graphen.

Betrachtet man die Verteilung der Datenfelder als Datenstruktur und die Zugriffsfunktionen als Algorithmus auf dieser Struktur, so kann jedes Datenverteilungsproblem als Konfigurationsproblem dargestellt werden. Dieses Kon-

figurationsproblem ist nun in der Zeit $O(|A| \cdot T_{\mathrm{PRAM}})$ optimal lösbar, wobei A für die Menge der Verteilungsvarianten und T_{PRAM} für die PRAM-Laufzeit des Programmes steht.

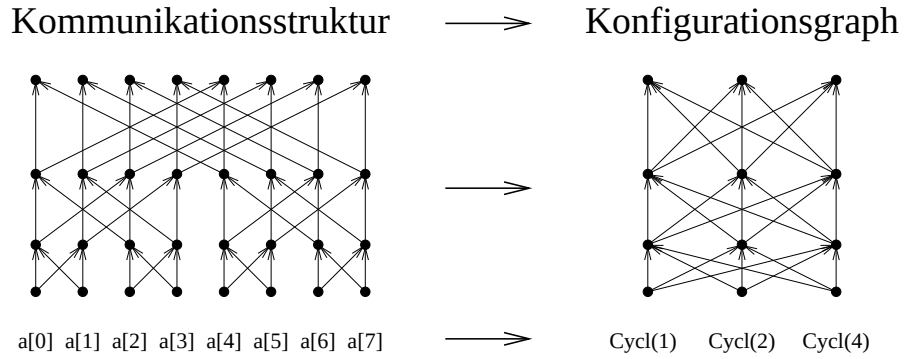

Abbildung 1: Übergang von der Verteilung einer Kommunikationsstruktur zur Berechnung des kürzesten Pfades durch einen Konfigurationsgraphen.

Auf den ersten Blick ergibt sich durch die Übertragung des Verteilungsproblems auf das Konfigurationsproblem kein wesentlicher Vorteil, da die Anzahl der möglichen Verteilungen A eines Datenfeldes F mit $|F|$ Elementen auf P Prozessoren gleich $P^{|F|}$ und damit exponentiell groß ist. Gelingt es jedoch, diese Anzahl auf ein „vernünftiges“ Maß zu reduzieren, wird die Durchführung des Konfigurationsalgorithmus praktikabel (siehe Abschnitt 3). Bei dieser Auswahl der Verteilungen können verschiedene Arten von Wissen benutzt werden:

- allgemeine heuristische Erfahrungswerte
- heuristische oder exakte Berechnugen aus dem Programmtext
- Angaben des Programmierers

Durch die Wahl der dem Konfigurationsalgorithmus zur Verfügung gestellten Varianten wird die Laufzeit des Konfigurationsalgorithmus gesteuert.

Zusätzlich bietet sich die Möglichkeit, die Konfigurationszeit durch die für Konfigurationen üblichen Verfahren zur Behandlung von Schleifen weiter zu reduzieren:

- Periodensuche beim Aufrollen von Schleifen
- Ersetzen von Schleifenrümpfen durch alle möglichen Fixpunkte
- Konfigurierung von n Schleifendurchläufen mit gleicher Kommunikationsstruktur in $\log(n)$ Schritten, via Matrizenpotenzierung.

Dem Konfigurationsalgorithmus können auch unterschiedliche Umstrukturierungen bezüglich der Ausführungsreihenfolge oder Vorschläge zur redundanten Berechnung präsentiert werden. Diese Vorgehensweise ist nicht von einer bestimmten Systemarchitektur (z.B. nachrichten-/speichergekoppelt) oder der Verwendung eines bestimmten Modells zur Kostenschätzung (z.B. LogP, LogP-Mesh, NUMA) abhängig.

Durch den Übergang zu einem Konfigurationsproblem erhält man die Möglichkeit, die in Programmen vorhandene Symmetrie auszunutzen, indem man z.B. Perioden in der optimalen Datenverteilung in Schleifen, die evtl. unveränderte Kommunikationsstruktur innerhalb von Schleifen oder die Gleichartigkeit von Speicherzugriffen erkennt und benutzt.

3 Auswahl der Verteilungsalternativen

Das oben beschriebene Verfahren ist nur dann praktisch anwendbar, wenn die Menge der Verteilungsalternativen klein genug gehalten werden kann. In die Auswahl der Alternativen, die dem Konfigurationsalgorithmus zur Verfügung gestellt werden, muß sowohl das Maschinenmodell als auch der Programmtext eingehen. Im folgenden soll nur auf das LogP-Maschinenmodell und Programmtexte in einer um synchrone parallele Schleifen erweiterten „üblichen“ Programmiersprache eingegangen weden. Diese Art von an das PRAM-Modell angelehnten Programmiersprachen eignet sich besonders für die Programmierung von parallelen numerischen Anwendungen.

Verteilungsalternativen können dem Konfigurationsalgorithmus global, d.h für jeden beliebigen Programmschritt, oder bei genauerer Kenntnis der Situation lokal, d.h. für eine bestimme Folge von Programmschritten zur Verfügung gestellt werden. Wird eine Verteilungsalternative aufgrund einer einzelnen Zuweisung oder einer Gruppe von Zuweisungen im Programmtext vorgeschlagen, ist es evtl. sinnvoll sie nur in diesem Bereich des Programmes vorzuschlagen. Erweitert man den Vorschlagsbereich etwas gegenüber dem Bereich, aus dem man den Vorschlag gewinnt, kann das Konfigurationssystem einen früheren oder späteren Wechsel auswählen.

Um die Migration zwischen zwei Verteilungsalternativen zu erleichern, sollten auch verschiedene Zwischenstufen zwischen den eigentlichen „Hauptalternativen“ vorgeschlagen werden, so daß der Wechsel zwischen den Alternativen in mehreren Schritten erfolgen kann, oder für mehrere Zuweisungen eine einheitliche Zwischenstufe ausgewählt werden kann.

3.1 Allgemeine Heuristiken

Die allermeisten parallelen numerischen Algorithmen zeichnen sich durch einen hohen Grad an Symmetrie aus. Es ist daher zu erwarten, daß sich symmetrische Datenverteilungen besonders eignen. Parallele Programiersprachen, die eine explizite Datenverteilung vom Programmierer erwarten (z.B. HPF [Richardson, 1995]), stellen dafür i.a. Kombinationen aus blockweisen und zyklischen Verteilungen zur Verfügung. Die Anzahl solcher Kombinationen ist quadratisch in der Feldgröße beschränkt. Die Konfigurationszeit für n Datenelemente beträgt bei Nutzung ausschließlich solcher Verteilungen somit $O(n^2 * T_{\mathrm{PRAM}})$.

3.2 Berechnung durch Schablonenauswertung

Besonders in numerischen Anwendungen werden Felder häufig elementweise durch gleiche Operationen Parallel neu berechnet (z.B. Jacobi-Iteration, Simulationen, FEM-Methoden). Hudak und Abraham untersuchten solche Situationen in sequentiell wiederholten parallelen Schleifen (sequentiel iterated parallel

loops, [Hudak und Abraham, 1990]). Ein solches Schleifenkonstrukt zeigt Abbildung 7.

Die Menge $\{(\delta_{\nu,1}, \ldots, \delta_{\nu,d}) | \nu = 1, \ldots, k\}$ der Vektoren, die auf rechten Seite der Zuweisung zum aktuellen Feldindex addiert werden, bezeichnen Hudak und Abraham als Schablone (stencil). Für diese Situationen geben sie eine Regel für das optimale Verhältnis der Kantenlängen beim blockweisen Verteilen der Daten an. Im folgenden soll eine nähere Betrachtung dieser Schleifenart vorgenommen werden. Selbst, wenn eine solche parallele Zuweisung nicht in einer sequentiellen Schleife steht, sind die im folgenden berechneten Verteilungen eine gute Heuristik für die Verteilungsalternativen.

3.2.1 Eindimensionale Felder

```
1    for j:= 1 to Steps do
2      forall i:=n to m do
3        a[i]:=Φ(a[i + δ_1], a[i + δ_2], ..., a[i + δ_k]);
4      end;
5    end;
```

Abbildung 2: Sequentiell iterierte parallele Schleife über einem eindimensionalen Feld.

In diesem Abschnitt wird zunächst der Fall eines eindimensionalen Feldes betrachtet. Eine sequentiell iterierte Schleife über ein solche Feld ist in Abbildung 2 zu sehen. Für das weitere Vorgehen werden zunächst folgende Definitionen benötigt:

Definition 1 *Zwei Feldelemente a[i] und a[j] sind bezüglich* $\{\delta_1, \delta_2, \ldots, \delta_k\}$ einfach abhängig*, wenn es ein h gibt mit* $i + \delta_h = j$ *oder* $j + \delta_h = i$.

Zwei Feldelemente a[i] und a[j] sind bezüglich $\{\delta_1, \delta_2, \ldots, \delta_k\}$ abhängig*, wenn es Feldelemente a[*h_l*]* $0 < l \leq n$ *gibt, so daß a[i]=a[*h_1*] und a[*h_n*]=a[j] ist, sowie alle a[*h_l*] von a[*h_{l+1}*] einfach abhängig sind.*

Zwie Teilfelder mit den Indexmengen A und B sind voneinander unabhängig*, wenn es keine zwei Feldelemente a[i] und a[j] mit* $i \in A$ *und* $j \in B$ *gibt die voneinander abhängig sind.*

Um zu testen, ob zwei Feldelemente a[i] und a[j] voneinander abhängig sind muß geprüft werden, ob durch mehrfache Addition oder Subtraktion von Elementen aus $\{\delta_1, \delta_2, \ldots, \delta_k\}$ die Differenz zwischen i und j überbrückt werden kann. Dies ist gleichbedeutend mit der Frage nach einer ganzzahligen Lösung der (liearen diophantischen) Gleichung

$$x_1 \cdot \delta_1 + x_2 \cdot \delta_2 + \ldots + x_k \cdot \delta_k = j - i. \tag{1}$$

Nach Diophant ist diese Gleichung genau dann lösbar, wenn $\Delta = \mathrm{ggT}(\delta_1, \ldots, \delta_k)$ Teiler von $j - i$ ist [Diophant, ca.250; Bundschuh, 1992]. Von einem gegebenen Feldelement a[i] sind somit höchstens diejenigen Feldelemente a[j] abhängig, deren Indices in derselben Δ-Restklasse wie i liegen. Dies führt direkt zu

Theorem 1 *Sei* $\Delta = ggT(\delta_1, \delta_2, \ldots, \delta_k)$*. Dann besteht die Berechnung in Abbildung 2 aus* mindestens Δ *voneinander unabhängigen Teilfeldern mit den Indexmengen* $I_\nu = \{i \mid n \leq i \leq m \wedge i \equiv \nu \mod \Delta\}$, $0 \leq \nu < \Delta$.

Ist der Ausschnitt des Indexbereiches $[n \ldots m]$ jedoch zu klein, kann es sein, daß der Zusammenhang der in Theorem 1 definierten Indexmengen nicht innerhalb von $[n \ldots m]$ hergestellt werden kann. Ist z.B. $k = 2$ und $\delta_1 = 7, \delta_2 = 9$ (und damit $\Delta = 1$), so gibt es für den Indexbereich $[10, \ldots, 20]$ mindestens zwei voneinander unabhängige Indexmengen. In Abbildung 3 ist eine dieser Mengen grau unterlegt.

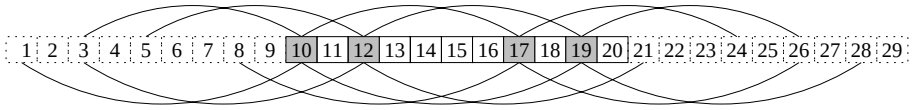

Abbildung 3: Abhängigkeiten für $k = 2$ und $\delta_1 = 7, \delta_2 = 9$ im Indexbereich $[10, \ldots, 20]$

Theorem 2 *Sei* $\Delta = ggT(\delta_1, \delta_2, \ldots, \delta_k)$ *und zusätzlich* $m - n \geq 2 \cdot \max\{\delta_\nu \mid 1 \leq \nu \leq k\}$. *Dann besteht die Berechnung in Abbildung 2 aus* maximal Δ *voneinander unabhängigen Teilfeldern mit den Indexmengen* $I_\nu = \{i \mid n \leq i \leq m \wedge i \equiv \nu \mod \Delta\}$, $0 \leq \nu < \Delta$.

Beweis Die Berechnung eines Wertes a[i] ist abhängig von der Berechnug des Wertes a[j], wenn es möglich ist, durch mehrfache Addition oder Subtraktion von Werten aus $\{\delta_1, \ldots, \delta_k\}$ vom Wert von i zum Wert von j zu gelangen, ohne daß mit den Zwischenwerten der Indexbereich des Feldes verlassen wird.

Ist der Indexbereich groß genug, dann ist dies gleichbedeutend mit der Frage nach einer ganzzahligen Lösung der Gleichung 1. Es muß nun nur noch gezeigt werden, daß für den „Weg“ der Indexbereich $\{n, \ldots, m\}$ nicht verlassen werden muß. Sei $(x_1, \ldots, x_k)$ eine Lösung der Gleichung 1, $\delta'_\nu = \text{sign}(x_\nu) \cdot \delta_\nu$. Nun muß in beliebiger Reihenfolge $|x_1|$ mal δ'_1, $|x_2|$ mal δ'_2, $\ldots$ und $|x_k|$ mal δ'_k addiert werden. Dabei wird der Bereich $\{n, \ldots, m\}$ nicht verlassen, wenn immer dasjenige δ'_h addiert wird, dessen Betrag am größten ist und bei dessen Addition der Bereich nicht verlassen wird.

Angenommen das Verfahren führt nicht zum Erfolg. Sei z die Zwischensumme bei Abbruch des Verfahrens. Dann gibt es noch y_i nicht addierte δ'_i mit $1 \leq i \leq k$ und für alle $y_i > 0$ gilt: $z + \delta'_i < n$ oder $z + \delta'_i > m$. Haben nun alle noch zu addierenden δ'_i dasselbe Vorzeichen, so kann der Endwert j nicht mehr erreicht werden. Im Widerspruch zur Annahme, daß $(x_1, \ldots, x_k)$ eine Lösung ist. Haben zwie der verbleibenden δ'_i verschiedenes Vorzeichen, so führt ihre Addition auf verschiedenen Seiten aus dem gewünschten Bereich heraus. Dann wäre aber $m - n < 2 \cdot \max\{\delta_\nu \mid 1 \leq \nu \leq k\}$. ◇

Für das in Abbildung 2 angegebene Schleifenkonstrukt ist also eine zyklische Verteilung mit dem Zyklenabstand $\Delta = \text{ggT}(\delta_1, \delta_2, \ldots, \delta_k)$ optimal, wenn sich die Δ unabhängigen Teilfelder auf die P Prozessoren verteilen lassen, d.h. P Δ teilt. Teilt P Δ nicht, müssen die überzähligen Teilfelder zusammengefaßt (wodurch sich die Werte der Schablone durch Δ teilen) und auf mehrere Prozessoren aufgeteilt werden (z.B. mit einer der folgenden Methoden).

Ist $\max_{i=1, \ldots, k}(|\delta_i|)$ klein gegenüber n so ist nach [Löwe et al., 1996] eine äquidistante blockweise Verteilung über alle Prozessoren in Kombination mit redundanten Berechnungen asymptotisch maximal um den Faktor 4 vom Optimum entfernt. Diese blockweise Verteilung kann somit auch für diejenigen

unabhängigen Teilfelder verwendet werden, die auf mehr als einen Prozessor verteilt werden müssen.

Der erste dieser beiden Fälle kommt für große δ_i in Betracht, der zweite für kleine. Kommen sowohl relativ große als auch relative kleine Werte in $\{\delta_1, \delta_2, \ \ldots \ , \delta_k\}$ vor und lassen sich die δ_i gut in Klassen einteilen, führt eine Faltung des Feldes zur Verkleinerung der δ_i. Abbildung 4 zeigt dies am Beispiel der Schablone $\{1, 2, 7, 8\}$.

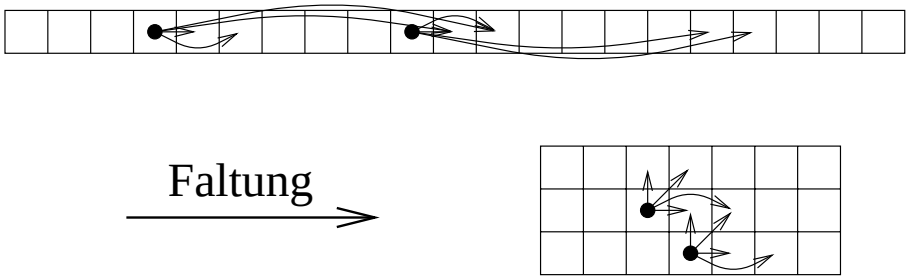

Abbildung 4: Faltung eines eindimensionales Feldes für die Schablone $\{1, 2, 7, 8\}$. Die Schablone für das gefaltete, nun zweidimensionale, Feld ist $\{(1, 0), (1, 2), (0, 1), (1, 1)\}$.

3.2.2 Bedingungen

Werden die Zuweisungen in Abhängigkeit von einer Bedingung ausgeführt, erhält man ein Programmstück wie in Abbildung 5. Ersetz man die Funktionen Φ^t und Φ^f durch die Funktion Ψ mit

$$\begin{aligned} &\Psi(a[i+\delta_1^t], \ \ldots \ , a[i+\delta_k^t], a[i+\delta_1^f], \ \ldots \ , a[i+\delta_k^f]) \\ &= \begin{cases} \Phi^t(a[i+\delta_1^t], \ \ldots \ , a[i+\delta_k^t]) & \text{falls} \quad Expr = True \\ \Phi^f(a[i+\delta_1^f], \ \ldots \ , a[i+\delta_k^f]) & \text{falls} \quad Expr = False \end{cases} \end{aligned}$$

ergibt sich das zu Abbildung 5 äquivalente Programm in Abbildung 6.[1]

Das Feld a zerfällt nach Theorem 1 in mindestens $ggT(a[i+\delta_1], \ \ldots \ , a[i+\delta_k], a[i+\delta_1^f], \ \ldots \ , a[i+\delta_k^f])$ Teilfelder. Eine Aussage über die maximale Anzahl von Teilfeldern ist jedoch nicht mehr möglich. Die Frage, ob ein durch Theorem 1 erhaltenes Teilfeld weiter unterteilt werden kann, kann nur durch weitere Untersuchung der Bedingung mit Hilfe eines entsprechenden Algebra-Systems oder durch exemplarische Ausführung des Programms beantwortet werden.

3.2.3 Mehrdimensionale Felder

Jedes mehrdimensionale Feld kann in ein eindimensionales Feld transformiert werden. Dies ist für die Betrachtung der Abhängigkeiten jedoch nicht optimal. Abbildung 7 zeigt eine sequentiell iterierte parallele Schleife über ein d-dimensionales Feld mit der Schablone $\{(\delta_{\nu,1}, \ \ldots \ , \delta_{\nu,d}) | \nu = 1, \ \ldots \ , k\}$.

[1] Alternative dazu können Bedingungen auch durch den Konfigurationsalgorithmus behandelt werden.

```
1     for j:= 1 to Steps do
2       forall i:=n to m do
3         if Expr then
4           a[i]:=Φ^t(a[i + δ^t_1], a[i + δ^t_2], ... , a[i + δ^t_k]);
5         else
6           a[i]:=Φ^f(a[i + δ^f_1], a[i + δ^f_2], ... , a[i + δ^f_k]);
7         end;
8       end;
9     end;
```

Abbildung 5: Sequentiell iterierte parallele Schleife über einem eindimensionalen Feld mit Bedingung.

```
1     for j:= 1 to Steps do
2       forall i:=n to m do
3         a[i]:=Ψ(a[i + δ_1], ... , a[i + δ_k], a[i + δ^f_1], ... , a[i + δ^f_k]);
4       end;
5     end;
```

Abbildung 6: Ersetzung der bedingten Zuweisung durch eine bedingte Funktion.

```
1     for j := 1 to Steps do
2       forall i_1 := n_1 to m_1 do
3         forall i_2 := n_2 to m_2 do
4           ...
5           forall i_d := n_d to m_d do
6             a[i_1, ... , i_d] := Φ(a[i_1 + δ_{1,1}, i_2 + δ_{1,2}, ... , i_d + δ_{1,d}],
7                                    a[i_1 + δ_{2,1}, i_2 + δ_{2,2}, ... , i_k + δ_{2,d}],
8                                    ...
9                                    a[i_1 + δ_{k,1}, i_2 + δ_{k,2}, ... , i_3 + δ_{k,d}]);
10          end;
11          ...
12        end;
13      end;
14    end;
```

Abbildung 7: Sequentiell iterierte parallele Schleife über einem mehrdimensionalen Feld.

Für den mehrdimensionalen Fall kann Theorem 1 auf jede Dimension einzeln angewendet werden. Die auf diese Weise in jeder Dimension erhaltenen unabhängigen Teilfelder sind zueinander orthogonal. Dies führt direkt zu:

Theorem 3 *Sei* $\Delta_\nu := ggT(\delta_{1,\nu}, \delta_{2,\nu}, \ \ldots \ , \delta_{k,\nu}), \nu = 1, \ \ldots \ , d$. *Dann besteht die Berechnung in Abbildung 7 aus* mindestens $\prod_{\nu=1}^{d} \Delta_\nu$ *voneinander unabhängigen Teilfeldern mit den Indexmengen* $I_{\mu_1, \ \ldots \ ,\mu_d} = \{(i_1, \ \ldots \ , i_d) \mid n_\nu \leq i_\nu \leq m_\nu \wedge i_\nu \equiv \mu_\nu \mod \Delta_\mu, \nu = 1, \ \ldots \ d\}$, $0 \leq \mu_\nu < \Delta_\nu$.

Die Anwendung von Theorem 1 auf jede einzelne Dimension ist jedoch nicht immer ausreichend. Abbildung 8 zeigt eines von 17 unabhängigen Teilfeldern für die Schablone $\{(4,3),(-3,2)\}$. Theorem 3 liefert uns aber nur $ggt(4,-3) \cdot ggt(3,2) = 1$ unabhängiges Teilfeld.

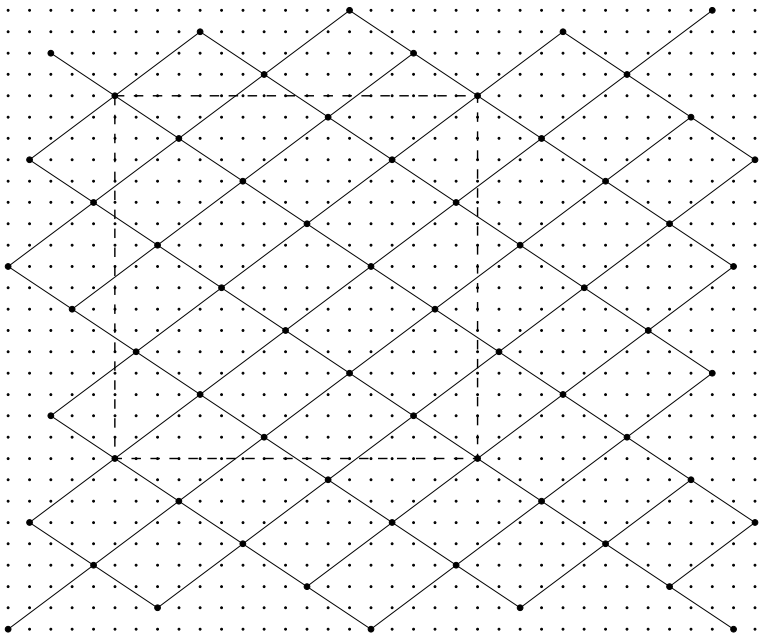

Abbildung 8: Unabhängiges Teilfeld für die Schablone $\{(4,3),(-3,2)\}$

Die Anzahl der unabhängigen Teilfelder in Abbildung 8 wird wegen der Regelmäßigkeit der Struktur durch den Abstand zweier voneinander abhängigen Punkte auf derselben Zeile bzw. Spalte bestimmt. Dieser Abstand kann für jede Dimension μ bestimmt werden, indem das Minimum m_μ des Betrages von

$$\delta_{1,\mu} \cdot x_1 + \delta_{2,\mu} \cdot x_2 + \ \ldots \ + \delta_{k,\mu} \cdot x_k \tag{2}$$

über alle Lösungen des linearen diophantischen Gleichungssystems

$$\begin{array}{rcl} \delta_{1,1} \cdot x_1 + \delta_{2,1} \cdot x_2 + \ \ldots \ + \delta_{k,1} \cdot x_k & = & 0 \\ \ldots & \ldots & \ldots \\ \delta_{1,\mu-1} \cdot x_1 + \delta_{2,\mu-1} \cdot x_2 + \ \ldots \ + \delta_{k,\mu-1} \cdot x_k & = & 0 \\ \delta_{1,\mu+1} \cdot x_1 + \delta_{2,\mu+1} \cdot x_2 + \ \ldots \ + \delta_{k,\mu+1} \cdot x_k & = & 0 \\ \ldots & \ldots & \ldots \\ \delta_{1,d} \cdot x_1 + \delta_{2,d} \cdot x_2 + \ \ldots \ + \delta_{k,d} \cdot x_k & = & 0 \end{array} \tag{3}$$

bestimmt wird. In der Richtung dieser Dimension gehört also nur jedes m_μ-te Feldelement zum selben unabhängigen Teilfeld. Jedes dieser m_μ ist somit eine untere Schranke für die Anzahl der unabhängigen Teilfelder. Somit folgt:

Theorem 4 *Die Berechnung in Abbildung 7 besteht aus* mindestens

$$\max_{\mu=1 \ \dots \ d} \{\min\{|(\delta_{i,\mu})_{i=1}^{k} \cdot \vec{x}| \mid \forall j = 1 \ \dots \ d, j \neq \mu : (\delta_{i,j})_{i=1}^{k} \cdot \vec{x} = 0, \vec{x} \in \mathbb{Z}^k\}\}$$

unabhängigen Teilfeldern.

Das dafür zu lösende Gleichungssytem 3 kann durch ganzahlige Gauss-elimination auf das Lösen von allgemeinen linearen diophantischen Gleichungen zurückgeführt werden. Einen Algorithmus dafür wurde zuerst von Aryabhata und Brahmagupta zwischen 500 und 600 n.Chr. aufgeschrieben und überliefert [Bundschuh, 1992]. Durch einsetzen des Lösungsraumes in den Term 2 erhält man einen von den freien Parametern des Lösungsraumes abhängigen Ausdruck. Der ggT der Faktoren dieser Parametern ergibt das gesuchte Minimum.

Diese Untersuchung macht keine Aussage über die maximale Anzahl von unabhängigen Teilfeldern und deren Aussehen. Das Verfahren kann aber nach dem Aufteilen in mehrere Felder wiederholt angewendet werden. Weitere Unterteilungen lassen sich durch eine eingehendere zahlentheoretische Untersuchung finden. Es stellt sich jedoch die Frage, inwieweit diese Fälle im praktischen Einsatz von Bedeutung sind.

3.2.4 Nicht-konstante Schablonen

Bisher wurde davon ausgegangen, daß die Schablone innerhalb der sequentiellen Schleife konstant bleibt. Ist dies nicht der Fall, müsen die Schablonen für jeden einzelnen sequentiellen Schleifendurchlauf bestimmt werden. Dies kann entweder durch Aufrollen der Schleife oder durch ein entsprechendes Algebra-System geschehen. Das Konfiguartionssystem muß die Schleifen aufrollen, solange Kenntnisse über den Ablauf der Schleife nicht anderwertig erlangt werden können.[2] Ist es dem Algebra-System zur Erkennung der Schablonen z.B. möglich, zu erkennen, daß bei der schnellen Fourier-Transformation die Werte in der Schablone exponentiell wachsen und daher ab einer bestimmten Iteration ein zyklische Verteilung über alle $P = 2^k$ Prozessoren anzuraten ist, braucht nur noch der Anfang der Schleife näher untersucht werden.

3.2.5 Mehrere Felder

Häufig wird in einer solchen parallelen Zuweisung nicht nur auf ein Datenfeld zugegriffen. Die günstigste Verteilung derjenigen Datenfelder, die nicht beschrieben werden, hängt von den Abständen zwischen denjenigen Feldelementen ab, die für eine einzelne Zuweisung gelesen werden, ihrer relativen Ausrichtung (alignment) zum zu schreibenden Elemement und der Verteilung des zu schreibenden Datenfeldes ab. Werden die nur gelesenen Datenfelder über längere Zeit nicht beschrieben, können ihre Elemente evtl. auch redundant auf mehreren Prozessoren gehalten werden.

[2] Beweis: Law of the excluded miracle.

3.3 Redundante Berechnungen

Papadimitriou und Yannakakis zeigen in [Papadimitriou und Yannakakis, 1990], daß es bei feingranularen Berechnungen besser sein kann, Teile der Zuweisungen mehrmals auf mehreren Prozessoren auszuführen. Diese redundante Berechnung kann bei der Datenverteilung durch Konfiguration auf zwei verschiedene Arten angeboten werden.

Zum einen können dem Konfigurationssystem die angebotenen Verteilungen zusätzlich mit verschiedenen Graden der Überlappung angeboten werden. Die Wahl, wann wieviel Redundanz eingesetzt wird, trifft dann im Rahmen der Vorgabe das Konfigurationssystem. Zum anderen kann das Programm durch eine Voruntersuchung (z.B auch Schablonenauswertung, Abschnitt 3.2) in grobere Stücke zerteilt werden (Clustring siehe [Löwe et al., 1996]). Die Redundanz wird hierbei durch die Vorverarbeitung festgelegt.

Diese Verfahren können natürlich auch gemischt werden. Entscheidungen können von der Vorverarbeitung an die Konfiguration weitergeleitet werden, wenn die Optimalität durch die Vorverarbeitung nicht eindeutig bestimmbar ist. Ist es dem Vorverarbeitungsystem z.B. bekannt, daß an einer bestimmten Stelle auf eine bestimmte Art von Redundanz „im Prinzip“ eingesetzt werden soll, so kann sie dem Konfigurationsalgorithmus eine gewisse Bandbreite an Redundanztiefe vorschlagen.

4 Bibliotheken

Mit Hilfe des Konfigurationssystems können auf einfache Art und Weise Bibliotheken für häufig gebrauchte verteilte Algorithmen zur Verfügung gestellt werden. Neben der Möglichkeit den Quelltext der Bibliotheksroutine mit den oben genannten Verfahren zu verteilen, können jetzt auch verschiedene von Hand auf ein bestimmtes verteiltes System optimierte Varianten mit verschiedenen Ein- und Ausgangsverteilungen der Daten als „Black-Box“ zur Verfügung gestellt werden. Aus diesen Varianten sucht das Konfigurationssystem die passendste aus. Es ist auch möglich häufig benutzte Routinen in einem aufwendigeren Optimierungslauf für verschiedene Ein- und Ausgangsverteilungen getrennt zu verteilen und später auf diese Weise einzubeziehen.

5 Zusammenfassung und Ausblick

Durch die Übertragung des Problems der Datenverteilung in ein Konfigurationsproblem ist es möglich, verschiedene Arten von Wissen und verschiedene Arten der Analyse auszunützen und miteinander zu kombinieren. Es können sowohl heuristische Annahmen als auch exakte Berechnungen als auch Angaben durch den Programmierer eingebracht werden. Die Schablonenanalyse zeigt, daß es möglich ist die regulären Strukturen, die die meisten Programme benutzen zu erkennen und auszunutzen.

Über die Intensität der Analyse, die Anzahl der vorgestellten Verteilungsalternativen und die Methoden des Konfigurationsalgorithmus kann die Laufzeit der Datenverteilung in einem breiten Rahmen beeinflußt und den Bedürfnissen an die zu erwartende Qualität der Optimierung angepaßt werden. Zudem ist es

möglich „von Hand“ oder in einem getrennten Durchlauf aufwendiger optimierte Bibliotheksfunktionen mit in die Optimierung einzubeziehen.

Weitere Arbeiten sind in dem Bereich der Gewinnung bzw. Einschränkung der Verteilungsalternativen von Nöten. Hier muß insbesondere die Kombination verschiedener Felder und die Erzeugung von Zwischen- bzw. Mischvarianten eingehender untersucht werden. Auch fehlt eine praktische Validierung.

Literatur

Bundschuh P. *Einführung in die Zahlentheorie.* Springer-Lehrbuch. Springer-Verlag, 1992.

Diophant. *Αριθμητικά* (Arithmetika). Alexandria, ca.250.

Hudak D.E. und Abraham S.G. Compiler techniques for data partitioning of sequentially iterated parallel loops. *Proceedings of the International Conference on Supercomputing, ACM SIGARCH News*, 18(3):187–200, 1990.

Löwe W., Eisenbiegler J. und Zimmerman W. Optimization of parallel programms on machines with expensive communication. In *Proceedings of EUROPAR '96.* 1996.

Löwe W. und Zimmermann W. Programming data-parallel – executing process-parallel. In *Proceedings of the ZEUS'95 Workshop on Parallel Programming and Computation.* 1995a.

Löwe W. und Zimmermann W. Upper time-bounds for executing pram-programs on the logp-machine. In *Proceedings of the 9th ACM International Conference on Supercomputing.* 1995b.

Moldenhauer H. Graph-based system configuration. Technischer Bericht 1996-22, Universität Karlsruhe, Fakultät für Informatik, 1996.

Papadimitriou C. und Yannakakis M. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19(2):322 – 328, 1990.

Philippsen M. *Optimierungstechniken zur Übersetzung paralleler Programmiersprachen.* VDI-Verlag, 1994.

Richardson H. High performance fortran: history, overview and current developments. Technischer Bericht TMC-261, Thinking Machines Corporation, 14 Crosby Drive, Bedford, MA 01730, USA, 1995.