UNIVERSITÄT PASSAU
Fakultät für Mathematik und Informatik

Automatische Methoden zur Parallelisierung im Polyedermodell

Diplomarbeit

Verfasser: Christian Wieninger

Aufgabensteller:
Prof. Christian Lengauer Ph. D.
Lehrstuhl für Informatik
Universität Passau

Betreuer:
Martin Griebl

Passau, den 15. Mai 1995

Vorab möchte ich mich bei Herrn Prof. Christian Lengauer Ph. D. bedanken, der stets für alle Fragen offen war und zur Lösung etwaiger Probleme beitrug. Mein besonderer Dank gilt meinem Betreuer, Herrn Martin Griebl, für seine unermüdliche Hilfsbereitschaft und konstruktiven Vorschläge zu allen Bereichen dieser Arbeit. Auch Herrn Jean-François Collard bin ich für seine hilfreichen Erklärungen zu großem Dank verpflichtet. Außerdem danke ich allen Mitgliedern des LooPo-Teams: Nils Ellmenreich, Peter Faber, Harald Keimer, Klaus Köbler, Frank Schuler, Sabine Wetzel und Alex Wüst, für viele ergiebige Diskussionen und Vorschläge während der Implementierung der Methoden, die in dieser Arbeit behandelt werden. Auch allen anderen Angehörigen des Lehrstuhls sei für ihre stete Hilfsbereitschaft gedankt.

Zusammenfassung

Mit der Einführung paralleler Rechnerarchitekturen wurde auch eine neue Form von Compilern notwendig, die sequentielle Programme parallelisieren, um die erhöhte Rechenleistung dieser Systeme zu nutzen. Wesentliche Bestandteile dieser Parallelisierung sind die Berechnung einer neuen Ausführungsreihenfolge für die Abarbeitung der einzelnen Befehle und die Verteilung der einzelnen Befehlsausführungen auf ein Mehrprozessorsystem. Diese Arbeit behandelt für beide Bereiche implementierbare Methoden, die einerseits effizient arbeiten und andererseits geringe Anforderungen an die zu parallelisierenden Programme und deren Abhängigkeiten stellen. Für den ersten Bereich werden zwei Verfahren mit sehr unterschiedlichen Ansätzen vorgestellt. Für den Bereich der Verteilung wird zunächst eine Methode behandelt, die eine lineare Anordnung der Prozessoren voraussetzt. Dieses Verfahren wird dann auf mehrdimensionale Prozessoranordnungen verallgemeinert.

Inhaltsverzeichnis

1 Einführung

Mit der Einführung paralleler Rechnerarchitekturen wurde auch eine neue Form von Compilern notwendig, die sequentielle Programme parallelisieren, um die erhöhte Rechenleistung dieser Systeme zu nutzen. Der Prozeß dieser Parallelisierung besteht zunächst aus einer Abhängigkeitsanalyse, gefolgt von der Berechnung einer neuen Ausführungshalbordnung, die durch einen sogenannten *Schedule* repräsentiert wird. Für die Verteilung der Berechnungen auf ein Mehrprozessorsystem wird eine sogenannte *Allokation* konstruiert. Abschließend wird anhand von Schedule und Allokation das parallele Programm aufgebaut.

Es gibt mittlerweile eine große Anzahl von Verfahren für jede dieser Aufgaben. Die meisten dieser Verfahren besitzen jedoch ernste Einschränkungen bzgl. der zu parallelisierenden Programme und der darin auftretenden Abhängigkeiten.

Gegenstand der vorliegenden Arbeit ist es, für die Bereiche Schedule und Allokation Methoden vorzustellen, die einerseits effizient arbeiten und andererseits geringe Anforderungen an die zu parallelisierenden Programme stellen. Diese Methoden wurden im Rahmen des Projekts LooPo[1] an der Universität Passau als Bestandteil dieser Arbeit implementiert.

Wir stellen nach der Einführung einiger Notationen in Abschnitt 3 die formalen Grundlagen zur Darstellung von Programmen und den darin auftretenden Abhängigkeiten vor. Die beiden folgenden Abschnitte bilden den Schwerpunkt dieser Arbeit. In Abschnitt 4 werden zwei Verfahren zur Berechnung von Schedules erläutert [1, 6]. In Abschnitt 5 gehen wir auf eine Methode zur Berechnung von Allokationen ein [8]. Im letzten Abschnitt wird schließlich die Konstruktion des parallelen Programms bei Vorgabe eines Schedules und einer Allokation angesprochen [8].

2 Notation

Zur Darstellung von Quantoren in formalen Ausdrücken benutzen wir die Notation von Dijkstra und Scholten [5]. Die Quantifizierung über einer Va-

[1] WWW-Referenz http://brahms.fmi.uni-passau.de/cl/loopo/index.html

riablen x hat damit die allgemeine Form:

$$(Qx \;:\; R.x \;:\; P.x)$$

Q repräsentiert den Quantifikator, R stellt den Wertebereich von x dar, und P ist ein Ausdruck in Abhängigkeit von x. Für formale Herleitungen in Beweisen benutzen wir folgende Darstellungsweise:

formaler Ausdruck 1
op { Kommentar zur Erklärung der Gültigkeit von $op \in \{ \Rightarrow, \Leftarrow, \Leftrightarrow \}$ }
formaler Ausdruck 2

Zur Darstellung der lexikografischen Ordnung auf einem k-dimensionalen $\mathbb{R}$-Vektorraum benutzen wir den Operator $\prec_l$ mit folgender Festlegung:

$$\begin{gathered}(\forall\ \vec{\imath},\vec{\jmath} : \vec{\imath},\vec{\jmath} \in \mathbb{R}^k \;:\; (\forall\ l \;:\; 1 \leq l \leq k \;:\; \vec{\imath} \prec_l \vec{\jmath} \Leftrightarrow \\ i_l < j_l \wedge (\forall\ m \;:\; 1 \leq m < l \;:\; i_m = j_m)))\end{gathered}$$

Wir verallgemeinern den Operator $\prec_l$ durch den Operator $\prec$ mittels:

$$(\forall\ \vec{\imath},\vec{\jmath} : \vec{\imath},\vec{\jmath} \in \mathbb{R}^k \;:\; \vec{\imath} \prec \vec{\jmath} \Leftrightarrow (\exists\ l \;:\; 1 \leq l \leq k \;:\; \vec{\imath} \prec_l \vec{\jmath}))$$

Die Operatoren $\succ_l$ und $\succ$ werden analog festgelegt.

Um formal auszudrücken, daß eine Zuweisung S in einem Eingabe-Programm syntaktisch vor einer Zuweisung T steht, schreiben wir $S <_L T$.

3 Grundlagen

3.1 Formale Darstellung von Eingabe-Programmen und Abhängigkeiten

Zunächst wollen wir in Anlehnung an die Darstellung in [6] und [1] festlegen, welche Programme als Eingabe für die vorzustellenden Verfahren akzeptiert werden sollen. Die erforderlichen Einschränkungen an diese Programme werden dabei so gering wie möglich gehalten und bei Bedarf später verschärft. Wir betrachten Fragmente von imperativen Programmen mit folgenden Eigenschaften:

- for-Schleifen und Zuweisungen sind die einzigen erlaubten Statements eines Programms und dürfen in beliebiger Kombination angeordnet werden. Die Schrittweite der for-Schleifen beträgt konstant +1. Es werden also auch sogenannte nicht perfekt verschachtelte Programme akzeptiert. Ein Programm heißt *perfekt verschachtelt*, wenn es die folgende Form besitzt:

  for $i_1 := l_1$ **to** u_1 **do**
      for $i_2 := l_2(i_1)$ **to** $u_2(i_1)$ **do**
          for $i_3 := l_3(i_1, i_2)$ **to** $u_3(i_1, i_2)$ **do**
              $\vdots$
              for $i_n := l_n(i_1, \ldots, i_{n-1})$ **to** $u_n(i_1, \ldots, i_{n-1})$ **do**
                  {Schleifenkörper B}
              end
              $\vdots$
          end
      end
  end

  Der Schleifenkörper B besteht ausschließlich aus Zuweisungen. Eine Iteration eines Schleifenkörpers bezeichnen wir mit $B(\vec{\imath})$, wobei $\vec{\imath} = (i_1, \ldots, i_n)$ ist.

- Die einzigen Datenstrukturen sind Skalare und Arrays beliebiger Dimension.

- Schleifengrenzen und Indizes von Array-Zugriffen sind affin lineare Ausdrücke in den Indizes umgebender Schleifen und in Strukturparametern. Mit *Strukturparametern* sollen Konstanten bezeichnet werden, für die ein Bezeichner vergeben wurde. Sie beschreiben oft die Problemgröße (z. B. Invertierung einer $n \times n$-Matrix mit n als Strukturparameter). Für die Schleifengrenzen gibt es zusätzlich die Möglichkeit, ein Maximum bzw. Minimum affiner Ausdrücke als untere bzw. obere Schranke festzulegen. Die Koeffizienten der Ausdrücke sind rational.

Ein Beispiel für ein Programm, das die Einschränkungen dieser Notation erfüllt, ist folgendes Fragment:

```
c[0] := 1;                                          {S_1}
for i := 0 to min(3, m/4) do
    for j := max(2 * i, 2 * n) to 20 do
        a[i, j - 2 * i] := a[i, 0];                 {S_2}
        b[i] := a[j, m];                            {S_3}
    end
    c[i] := a[i, n] + c[i];                         {S_4}
end
```

Ein Programm, das diesen Einschränkungen genügt, wird im folgenden als ein *Schleifenprogramm*[2] bezeichnet und gewöhnlich mit L benannt. Die Menge der Zuweisungen eines Schleifenprogramms L bezeichnen wir mit Ω_L.

Ein Schleifenprogramm besitzt einen statischen Kontrollfluß. Dadurch ist es möglich, jede Ausführung eines Befehls zur Compilezeit durch Angabe der Zuweisung und des Indexvektors zu kennzeichnen.

Definition 3.1.1 (Indexvektor und Indexraum) Gegeben sei ein Schleifenprogramm L mit Zuweisungsmenge Ω_L. Für eine Zuweisung $S \in \Omega_L$ seien $i_1, \ldots, i_m$ die umgebenden m Schleifenindizes und l_j bzw. u_j die untere bzw. obere Grenze eines Index i_j, $1 \leq j \leq m$. Dann bezeichnet

$$D_S = \{\, (i_1, \ldots, i_m) : (i_1, \ldots, i_m) \in \mathbb{Z}^m \wedge (\forall\, j \,:\, 1 \leq j \leq m \,:\, l_j \leq i_j \leq u_j) \,\}$$

den *Indexraum* von S. Ein Vektor $\vec{\imath} \in D_S$ heißt *Indexvektor* von S. Mit

$$D_{\Omega_L} = \{D_S : S \in \Omega_L\}$$

wird die Menge aller Indexräume eines Schleifenprogramms bezeichnet. p_S gibt die Dimension eines Indexraums D_S an.

Geometrisch betrachtet stellt ein Indexraum ein Polyeder $\{x \,:\, Ax \geq b\}$ im $\mathbb{Z}^m$ dar, wobei das Ungleichungssystem $Ax \geq b$ wie in der Definition angegeben durch die Schleifengrenzen festgelegt ist.

[2]Der in der Theorie geläufige Begriff *Schleifensatz* bezeichnet Eingabe-Programme, in denen stets eine umgebende Schleife existiert. Da in unserer Festlegung aber auch Zuweisungen ohne umgebende Schleifen und auch aufeinanderfolgende Schleifen erlaubt sind, verwenden wir den Begriff Schleifenprogramm.

Definition 3.1.2 (Instanz einer Zuweisung) Sei L Schleifenprogramm, $S \in \Omega_L$ eine Zuweisung in L und $\vec{\imath} \in D_S$ ein Indexvektor. Dann heißt das Paar $(S, \vec{\imath})$ *Instanz* der Zuweisung S mit Indexvektor $\vec{\imath}$.

Damit sind sämtliche Konstrukte definiert, die wir zur formalen Darstellung des Eingabe-Programms benötigen. Als nächsten Punkt wollen wir nun eine Darstellung für auftretende Abhängigkeiten festlegen [1].

Definition 3.1.3 (Abhängigkeit) Gegeben seien ein Schleifenprogramm L, zwei nicht notwendig verschiedene Zuweisungen $S, T \in \Omega_L$ und zwei Indexvektoren $\vec{\imath} \in D_S$, $\vec{\jmath} \in D_T$. Dann heißt die Instanz $(S, \vec{\imath})$ *abhängig* von der Instanz $(T, \vec{\jmath})$, i. Z. $(S, \vec{\imath})\delta(T, \vec{\jmath})$, falls gilt:

1. $(S, \vec{\imath})$ und $(T, \vec{\jmath})$ greifen beide auf dieselbe Speicherzelle M zu, wobei mindestens einer dieser Zugriffe schreibend ist, und

2. $(S, \vec{\imath})$ wird in L vor $(T, \vec{\jmath})$ ausgeführt.

Sind zwei Instanzen $(S, \vec{\imath})$ und $(T, \vec{\jmath})$ abhängig so heißen auch die Zuweisungen S und T abhängig, i. Z. $S\delta T$.

Mit dieser Darstellung von Abhängigkeiten wird ein breites Spektrum abgedeckt. Es werden auch *indirekte* Abhängigkeiten erfaßt. Für direkte Abhängigkeiten fordert man zusätzlich, daß zwischen den beiden Zugriffen durch S und T kein anderer Schreibzugriff auf die Speicherzelle erfolgt. Eine Spezialfall von Abhängigkeiten, der für Methoden zur Parallelisierung von Programmen im Polyedermodell häufig auftaucht, sind uniforme Abhängigkeiten.

Definition 3.1.4 (uniforme Abhängigkeit) Für ein Schleifenprogramm L mit der Zuweisungsmenge Ω_L und $S, T \in \Omega_L$ heißt eine direkte Abhängigkeit $S\delta T$ *uniform*, falls $D_S = D_T$ und ein lexikografisch positiver Vektor $\vec{d} \in \mathbb{Z}^{p_S}$ existiert, sodaß die Differenz zwischen den Indexvektoren der abhängigen Instanzen von S und T konstant $\vec{d}$ ist. Formal:

$$(\exists\, \vec{d} \,:\, \vec{d} \in \mathbb{Z}^{p_S} \wedge \vec{d} \succ \vec{0} \,:\, (\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath}, \vec{\jmath} \in D_S \wedge (S, \vec{\imath})\delta(T, \vec{\jmath}) \,:\, \vec{d} = \vec{\jmath} - \vec{\imath}))$$

Der Vektor $\vec{d}$ heißt dann der *Distanzvektor* der Abhängigkeit.

Im Zusammenhang mit uniformen Abhängigkeiten in einem perfekt verschachtelten Schleifenprogramm spricht man auch vom Level einer Abhängigkeit:

Definition 3.1.5 (Level uniformer Abhängigkeiten) Gegeben sei ein perfekt verschachteltes Schleifenprogramm $L = (L_1, \ldots, L_m)$ mit m Schleifen und der Distanzvektor $\vec{d} = (d_1, \ldots, d_m)$ einer uniformen Abhängigkeit in L. Da ein Distanzvektor stets ungleich dem Nullvektor ist, existiert ein von links gelesen erstes Element $d_l \neq 0$. Formal:

$$(\exists\, l \,:\, l \in \mathbb{N} \wedge 1 \leq l \leq m \,:\, l = (\min\ k \,:\, k \in \mathbb{N} \wedge 1 \leq k \leq m \wedge d_k \neq 0 \,:\, k))$$

Dieses l bezeichnet man als den *Level* von $\vec{d}$, i. Z. $level(\vec{d})$. Es gilt also

$$level(\vec{d}) = l \Leftrightarrow \vec{d} \succ_l \vec{0}$$

Definition 3.1.6 (Abhängigkeiten tragende Schleife) Gegeben sei ein perfekt verschachteltes Schleifenprogramm $L = (L_1, \ldots, L_m)$ mit m Schleifen und der Distanzvektor $\vec{d} = (d_1, \ldots, d_m)$ einer uniformen Abhängigkeit in L. Ist l der Level von $\vec{d}$, so sagt man, daß die Schleife L_l diese Abhängigkeit *trägt*, i. Z. $carries_d(L_l)$, ansonsten gilt $\neq carries_d(L_l)$. Formal:

$$(\forall\, l \,:\, 1 \leq l \leq m \,:\, carries_d(L_l) \;=\; (\,level(\vec{d}) = l\,))$$

Anmerkung 3.1.7 Uniforme Abhängigkeiten sind Abhängigkeiten zwischen den Iterationen eines Schleifenprogramms. Um von den Zuweisungen auf die Iterationen zu abstrahieren, benutzen wir die Notation $B(\vec{\imath})\delta B(\vec{\jmath})$ für zwei abhängige Iterationen $B(\vec{\imath})$ und $B(\vec{\jmath})$.

In Definition 3.1.3 wird unter 2. implizit von einer Abbildung

$$\Theta : (\Omega_L \times D_{S \in \Omega_L}) \rightarrow \mathbb{Q}$$

ausgegangen, die jeder Instanz jeder Zuweisung, also jedem Paar $(S, \vec{\imath}) \in \Omega_L \times D_{S \in \Omega_L}$ einen Ausführungszeitpunkt zuweist. Eine solche Abbildung bezeichnet man als *Schedule*, falls sie die Scheduleeigenschaft besitzt.

Definition 3.1.8 (Scheduleeigenschaft) Eine Abbildung

$$\Theta : (\Omega_L \times D_{S\in\Omega_L}) \to \mathbb{Q}$$

besitzt für ein Schleifenprogramm L genau dann die *Scheduleeigenschaft*, wenn für jedes abhängige Paar von Zuweisungen $S, T \in \Omega_L$ gilt:

$$(\forall\, \vec{\imath}, \vec{\jmath} : \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \wedge (S, \vec{\imath})\delta(T, \vec{\jmath}) \; : \; \Theta(S, \vec{\imath}) < \Theta(T, \vec{\jmath}))$$

Mit

$$(\forall\, S \; : \; S \in \Omega_L \; : \; \Theta_S(\vec{\imath}) := \Theta(S, \vec{\imath}))$$

wird der *Schedule der Zuweisung* S definiert.

Man beachte, daß Schedules von unterschiedlichen Zuweisungen i. a. unterschiedlich sind.

Anmerkung 3.1.9 Für eine uniforme Abhängigkeit mit Distanzvektor $\vec{d}$ kann eine affine Abbildung Θ der Form

$$\Theta(S, \vec{\imath}) = \vec{t}\vec{\imath} + c$$

dadurch auf die Scheduleeigenschaft bzgl. dieser Abhängigkeit getestet werden, daß man prüft, ob $\vec{t}\vec{d} > 0$ gilt. Dies folgt unmittelbar aus der Definition uniformer Abhängigkeiten und der Festlegung eines Schedules.

Im Zusammenhang mit einem Schedule definiert man als eine weitere Abbildung die Allokation.

Definition 3.1.10 (Allokation) Wird für ein Schleifenprogramm L durch die Abbildung

$$\Pi : (\Omega_L \times D_{S\in\Omega_L}) \to \mathbb{Q}$$

jeder Instanz einer Zuweisung ein Prozessor zugewiesen, auf dem diese Instanz ausgeführt wird, so heißt die Abbildung Π *Allokation* für L. Die *Allokation einer Zuweisung* S ist

$$\Pi_S(\vec{\imath}) := \Pi(S, \vec{\imath})$$

In vielen Verfahren wird die Allokation in Abhängigkeit vom Schedule berechnet. Man fordert dabei, daß zwei Instanzen, die durch den Schedule auf denselben Ausführungszeitpunkt abgebildet werden, auf unterschiedlichen Prozessoren berechnet werden. Formal:

$$(\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath} \in D_S, \vec{\jmath} \in D_T \,:\, \Theta(S, \vec{\imath}) = \Theta(T, \vec{\jmath}) \Rightarrow \Pi(S, \vec{\imath}) \neq \Pi(T, \vec{\jmath}))$$

Diese Bedingung vereinfacht die im späteren Verlauf der Parallelisierung nötige Programmtransformation. Geht man davon aus, daß ein Prozessor zu einer Zeit nur eine Zuweisung abarbeitet und interpretiert man die Werte des Schedules als Systemzeit, so hat diese Bedingung auch einen technischen Ursprung. Repräsentieren diese Werte jedoch eine virtuelle Zeit, so ist diese Anforderung an die Allokation nicht notwendig, da die verschiedenen Instanzen dann in beliebiger Reihenfolge auf demselben Prozessor ausgeführt werden können.

Wir haben Schedule und Allokation als Abbildungen mit Bild in $\mathbb{Q}$ festgelegt. Es ist aber auch möglich, diese Abbildungen mit einem höherdimensionalen Bild zu berechnen. Dies kann vor allem bei der Allokation im Hinblick auf die Architektur des Parallelrechners hilfreich sein. Hat dieser beispielsweise ein zweidimensionales Prozessorfeld, so benötigt man auch eine zweidimensionale Allokation.

Abhängigkeiten definieren ein zeitliches Verhältnis zwischen Zuweisungen bzw. Instanzen von Zuweisungen. Diese Relation veranschaulicht der allgemeine Abhängigkeitsgraph.

Definition 3.1.11 (Allgemeiner Abhängigkeitsgraph) Ein *allgemeiner Abhängigkeitsgraph* G ist ein Tupel (V, E, D, R). Dabei gilt:

- V ist die Knotenmenge von G. Jede Zuweisung $S \in \Omega_L$ ist ein Knoten in G. Ihr wird mittels der Abbildung $D : V \to D_{\Omega_L}$ ein Indexraum zugeordnet.

- E ist die Kantenmenge von G. Zwischen zwei Knoten $S, T \in V$ existiert eine Kante $e = (S, T)$, falls es abhängige Instanzen dieser Zuweisungen gibt. Formal:

$$E = \{(S, T) : (\exists\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \,:\, (S, \vec{\imath})\delta(T, \vec{\jmath}))\}$$

Mit $\sigma(e)$ bezeichnet man die Zuweisung, von der eine Abhängigkeit ausgeht, mit $\delta(e)$ das Ziel der Abhängigkeit. Einer Kante e wird durch die Abbildung $R : E \to \mathbb{Z}^{p_S+p_T}$ ein Polyeder R_e zugeordnet, wobei

$$R_e = \{(\vec{\imath}, \vec{\jmath}) : \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \wedge (S, \vec{\imath})\delta(T, \vec{\jmath})\} \tag{1}$$

R_e enthält die Paare von Indexvektoren von Instanzen, zwischen denen eine Abhängigkeit existiert.

Beispiel 3.1.12 Gegeben sei folgendes Programm:

$$\begin{array}{ll}
\textbf{for } i := 0 \textbf{ to } n \textbf{ do} & \\
\quad \textbf{for } j := 0 \textbf{ to } n \textbf{ do} & \\
\quad\quad a[i,j] := a[i-2, j+1] & \{S_1\} \\
\quad \textbf{end} & \\
\quad s[i] := a[i,n] & \{S_2\} \\
\textbf{end} &
\end{array}$$

Da es sich hier um ein nicht perfekt verschachteltes Programm handelt, sind die Indexräume der Zuweisungen unterschiedlich. Für die Beschreibung des Indexraums D_{S_1} ergeben sich die Ungleichungen

$$0 \leq i, j \leq n,$$

für D_{S_2} ergibt sich

$$0 \leq i \leq n$$

Es existieren zwei Abhängigkeiten. Zum einen entsteht durch das Schreiben von $a[i,j]$ in Zuweisung S_1 und das spätere Lesen von $a[i-2, j+1]$ in Zuweisung S_1 eine Abhängigkeit mit Kante $e_1 = (S_1, S_1)$ im allgemeinen Abhängigkeitsgraphen. Für die Instanzen gilt:

$$(\forall\, i_1, j_1, i_2, j_2 \;:\; (i_1, j_1) \in D_{S_1} \wedge (i_2, j_2) \in D_{S_1} \wedge i_2 = i_1 - 2 \wedge j_2 = j_1 + 1 \;: \\ (S_1, (i_1, j_1))\delta(S_1, (i_2, j_2)))$$

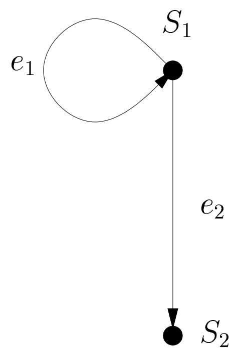

Abbildung 1: Der allgemeine Abhängigkeitsgraph aus Beispiel 3.1.12.

Damit sind auch die Indexpunkte des entsprechenden Polyeders $R_{e_1} \in \mathbb{Z}^4$ festgelegt. Mit (1) beschreiben folgende Ungleichungen R_{e_1}:

$$\begin{array}{rcl} 2 \leq & i_1 & \leq n \\ 0 \leq & j_1 & \leq n-1 \\ i_1 - 2 \leq & i_2 & \leq i_1 - 2 \\ j_1 + 1 \leq & j_2 & \leq j_1 + 1 \end{array}$$

Eine weitere Abhängigkeit mit Kante $e_2 = (S_1, S_2)$ im allgemeinen Abhängigkeitsgraphen gibt es zwischen den Zuweisungen S_1 und S_2. Die letzte Speicherzelle, die in der inneren Schleife beschrieben wird, ist $a[i, n]$. Diese wird im Anschluß von S_2 gelesen. Die Indexpunkte abhängiger Instanzen sind wie folgt festgelegt:

$$(\forall\, i_1, j_1, i_2 \,:\, (i_1, j_1) \in D_{S_1} \wedge i_2 \in D_{S_2} \wedge i_2 = i_1 \wedge j_1 = n \,:\, (S_1, (i_1, j_1))\delta(S_2, i_2))$$

Der allgemeine Abhängigkeitsgraph ist in Abbildung 1 dargestellt. Wie zuvor ergeben sich mit (1) für die Beschreibung des Polyeders $R_{e_2} \in \mathbb{Z}^3$ folgende Ungleichungen:

$$\begin{array}{rcl} 0 \leq & i_1 & \leq n \\ n \leq & j_1 & \leq n \\ i_1 \leq & i_2 & \leq i_1 \end{array}$$

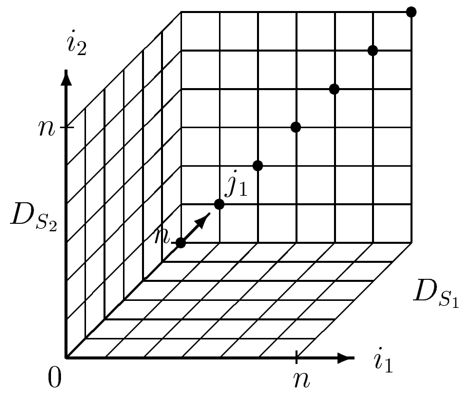

Abbildung 2: Das Polyeder R_{e_2} aus Beispiel 3.1.12.

R_{e_2} hat also die in Abbildung 2 gezeigte graphische Darstellung (Punkte in der Graphik repräsentieren Paare von Indexpunkten abhängiger Instanzen).

Beim allgemeinen Abhängigkeitsgraphen wird also in den assoziierten Polyedern R_e festgehalten, welche Instanzen voneinander abhängen. Diese Darstellung hat aber zum einen Nachteile in der Implementierung, zum anderen sind aus den Ungleichungen, die das Polyeder beschreiben, die Indexvektoren der zugehörigen Instanzen nur schwer zu erkennen, solange keine graphische Darstellung existiert. Abhilfe schafft eine weitere Möglichkeit zur Repräsentation von Abhängigkeiten:

Mit den anfangs genannten Einschränkungen an die Eingabe-Programme ist es mittels Datenflußanalyse möglich, für jede Abhängigkeit zwischen den Zuweisungen $S, T \in \Omega_L$ mit Kante $e = (S, T)$ im allgemeinen Abhängigkeitsgraphen eine affine Funktion $h_e : D_{\delta(e)} \to D_{\sigma(e)}$ zu finden [9], die für jede Instanz im Ziel einer Abhängigkeit die Instanz einer Zuweisung angibt, die Quelle der Abhängigkeit ist.

Definition 3.1.13 (h-Transformation) Seien $S, T \in \Omega_L$ zwei Zuweisungen eines Schleifenprogramms L und e die Kante der Abhängigkeit $S\delta T$ im

```
for i := 0 to n do
    s := i;                    {S1}
    for j := 0 to n do
        a[i, j] := s;          {S2}
    end
end
```

Abbildung 3: Kein eindeutiges Ziel der Abhängigkeit $S_1 \delta S_2$

allgemeinen Abhängigkeitsgraph. Eine Abbildung $h_e \in D_{\delta(e)} \rightarrow D_{\sigma(e)}$ heißt *h-Transformation* der Abhängigkeit, falls sie jeden Indexpunkt einer abhängigen Instanz von T auf den Indexpunkt der zugehörigen Instanz von S abbildet. Formal:

$$(\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath} \in D_{\sigma(e)} \wedge \vec{\jmath} \in D_{\delta(e)} \wedge (S, \vec{\imath})\delta(T, \vec{\jmath}) \,:\, h_e(\vec{\jmath}) = \vec{\imath})$$

Anmerkung 3.1.14 Es stellt sich die Frage, ob die h-Transformation nicht auch als Funktion im Raum $D_{\sigma(e)} \rightarrow D_{\delta(e)}$ definiert werden kann. Das ist aber nicht möglich, da die Quelle einer Abhängigkeit zwar immer eindeutig ist, nicht aber das Ziel einer Abhängigkeit, wie das Fragment in Abbildung 3 für $S_1 \delta S_2$ zeigt. Bei dieser Wahl wäre eine Darstellung der Abhängigkeit durch eine Funktion nicht möglich.

Anmerkung 3.1.15 Bei einer uniformen Abhängigkeit ist die h-Transformation eine Translation um den Distanzvektor.

Definiert man nun das Polyeder P_e, das die Indexpunkte von Instanzen im Ziel der Abhängigkeit beschreibt, mit

$$P_e = \{\vec{\jmath} : \vec{\jmath} \in D_{\delta(e)} \wedge (\exists\, \vec{\imath} \,:\, \vec{\imath} \in D_{\sigma(e)} \,:\, \vec{\imath} = h_e(\vec{\jmath}))\}$$

so gilt:

$$\begin{array}{ll} & (\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath} \in D_{\sigma(e)} \wedge \vec{\jmath} \in D_{\delta(e)} \,:\, (\vec{\imath}, \vec{\jmath}) \in R_e) \\ \Leftrightarrow & (\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath} \in D_{\sigma(e)} \wedge \vec{\jmath} \in D_{\delta(e)} \,:\, \vec{\imath} = h_e(\vec{\jmath}) \wedge \vec{\jmath} \in P_e) \end{array}$$

Durch die h-Transformation h_e und das Polyeder P_e ist es folglich möglich, die Punkte in R_e und damit die Abhängigkeit vollständig zu beschreiben. Wir werden diese Darstellung im folgenden benutzen.

Beispiel 3.1.16 Die h-Transformationen aus Beispiel 3.1.12 ergeben sich wie folgt:

$$h_{e_1}(i,j) = (i-2, j+1)$$
$$h_{e_2}(i) = (i, n)$$

Die zugehörigen Polyeder sind:

$$P_{e_1} = \{i, j : 2 \leq i \leq n \wedge 0 \leq j \leq n-1\}$$
$$P_{e_2} = \{i : 0 \leq i \leq n\}$$

3.2 Grundidee der schleifenbasierten Parallelisierung

Bei der Parallelisierung eines Eingabe-Programms unterscheidet man zwischen zwei Ansätzen. Einerseits existieren Verfahren, welche die Instanzen von Zuweisungen eines Programms unter Berücksichtigung der Abhängigkeiten neu ordnen. Wir bezeichnen diesen Ansatz als *statement-basiert*. Ein anderes und oberflächlicheres Prinzip ist es, eine Schleife mit ihrem Schleifenkörper so umzuformen, daß diese Schleife parallel ausgeführt werden kann. In diesem Abschnitt wollen wir die notwendigen Grundlagen für den letzten, sogenannten *schleifenbasierten* Ansatz vorstellen. Wir suchen hier nach einem Kriterium für die Ersetzung von sequentiellen Schleifen durch parallele Schleifen. Die Schleifenindizes und ihre Grenzen bleiben dabei also erhalten. Wir werden im nächsten Abschnitt ein schleifenbasiertes Verfahren kennenlernen, das auf diesen Grundlagen aufbaut und zusätzlich eine Transformation der Schleifenindizes vornimmt.

Wir beschränken uns bei den folgenden Untersuchungen auf ein perfekt verschachteltes Schleifenprogramm mit ausschließlich uniformen Abhängigkeiten. Zur Notation paralleler Schleifensprogramme verwenden wir zur Kennzeichnung von sequentiellen Schleifen das Konstrukt **seqfor**, zur Kennzeichnung paralleler Schleifen das Konstrukt **parfor**. Für die formale Darstellung der Ersetzung definieren wir das Prädikat *parfor* mit folgender Festlegung: Wird eine Schleife L_i eines Schleifenprogramms $L = (L_1, \ldots, L_m)$ durch eine parallele **parfor**-Schleife ersetzt, so gilt $\mathit{parfor}(L_i)$.

Allgemein gilt, daß sich durch die Parallelisierung die *Ausführungsordnung* eines Eingabe-Programms ändert. Im sequentiellen Eingabe-Programm ist diese Ordnung eine Totalordnung, dadurch festgelegt, daß eine Iteration $B(\vec{\imath})$ vor einer Iteration $B(\vec{\jmath})$ ausgeführt wird, wenn $\vec{\imath} \prec \vec{\jmath}$ gilt. Bei der Erzeugung des parallelen Programms wird ausgenutzt, daß lediglich abhängige Iterationen nacheinander ausgeführt werden müssen, um die Semantik zu erhalten. Die Ausführungsordnung ist dann eine partielle Ordnung. Wir suchen also nach einer neuen Ausführungsordnung für das gegebene Schleifenprogramm und bezeichnen diese als *gültig*, wenn für zwei abhängige Iterationen $B(\vec{\imath})$ und $B(\vec{\jmath})$ des transformierten Schleifenprogramms gilt, daß $B(\vec{\imath})$ vor $B(\vec{\jmath})$ ausgeführt wird.

Um die Semantik eines parallelen Schleifenprogramms bestehend aus **seqfor** - und **parfor**-Schleifen definieren zu können, benötigen wir folgende Festlegung:

Definition 3.2.1 (parallele Ausführungsordnung) Für ein paralleles Schleifenprogramm L' wird die Ausführungsordnung wie folgt definiert:

> Eine Iteration $B(\vec{\imath})$ von L' wird vor einer anderen Iteration $B(\vec{\jmath})$ ausgeführt, wenn es eine **seqfor**-Schleife L'_r gibt und $\vec{\imath} \prec_r \vec{\jmath}$ gilt.

Definition 3.2.2 (Äquivalenz von Schleifensätzen) Sei L ein sequentielles Schleifenprogramm und L' ein aus L erzeugtes Schleifenprogramm mit neuer Ausführungsordnung, so heißen L und L' äquivalent, i. Z. $L \equiv L'$, wenn die Ausführungsordnung von L' gültig ist.

Folgender Satz gibt uns das gesuchte Kriterium [1]:

Lemma 3.2.3 (Parallelisierbarkeit von Schleifen) Gegeben sei ein sequentielles Schleifenprogramm $L = (L_1, \ldots, L_m)$ mit nur uniformen Abhängigkeiten. Sei $L' = (L'_1, \ldots, L'_m)$ das Schleifenprogramm, das aus L durch Ersetzung einiger Schleifen durch **parfor**-Schleifen entsteht. Dann ist L äquivalent zu L', genau dann wenn keine der ersetzten Schleifen eine Abhängigkeit in L trägt. Formal:
Ist $\mathcal{L}' = \{L'_1, \ldots, L'_m\}$ die Menge der Schleifen von L', $\mathcal{L}'_{repl} \subseteq \mathcal{L}'$ die Menge der ersetzten Schleifen und $\mathcal{D}$ die Menge der Distanzvektoren aller Abhängigkeiten, so gilt:

$$L \equiv L' \Leftrightarrow$$
$$(\forall\, i \,:\, 1 \leq i \leq m \wedge L'_i \in \mathcal{L}'_{repl} \,:\, (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, \neg carries_{L_i}(\vec{d}))$$

Beweis:

"$\Leftarrow$": Voraussetzung: Keine der ersetzten Schleifen trägt eine Abhängigkeit in L. Um die Äquivalenz zu beweisen, muß für zwei abhängige Iterationen in L gezeigt werden, daß diese auch in L' nacheinander ausgeführt werden. Sei dazu $B(\vec{\jmath})$ eine Iteration in L, die von einer Iteration $B(\vec{\imath})$ in L mit Distanzvektor $\vec{d}$ abhängt. Dann gilt:

$$
\begin{array}{ll}
 & B(\vec{\jmath})\delta B(\vec{\imath}) \\
\Rightarrow & \quad \{ \text{ sequentielle Ausführungsordnung in } L \ \} \\
 & \vec{\imath} \prec \vec{\jmath} \\
\Leftrightarrow & \quad \{ \text{ lexikographische Ordnung } \} \\
 & (\exists\, r \,:\, 1 \le r \le m \,:\, \vec{\imath} \prec_r \vec{\jmath}) \\
\Leftrightarrow & \quad \{ \ \vec{d} \text{ Distanzvektor der Abhängigkeit } \} \\
 & (\exists\, r \,:\, 1 \le r \le m \,:\, \vec{d} \succ_r \vec{0}) \\
\Leftrightarrow & \quad \{ \text{ Def. 3.1.5 } \} \\
 & (\exists\, r \,:\, 1 \le r \le m \,:\, \mathit{level}(\vec{d}) = r) \\
\Leftrightarrow & \quad \{ \text{ Def. 3.1.6 } \} \\
 & (\exists\, r \,:\, 1 \le r \le m \,:\, \mathit{carries}_{\vec{d}}(L_r)) \\
\Rightarrow & \quad \{ \text{ Voraussetzung } \} \\
 & L_r \notin \mathcal{L}'_{repl} \\
\Rightarrow & \quad \{ \text{ Def. 3.2.1 } \} \\
 & B(\vec{\imath}) \text{ wird in } L' \text{ vor } B(\vec{\jmath}) \text{ ausgeführt} \\
\Leftrightarrow & \quad \{ \text{ Definition gültiger Ausführungsordnung } \} \\
 & \text{Ausführungsordnung gültig} \\
\Leftrightarrow & \quad \{ \text{ Def. 3.2.2 } \} \\
 & L \equiv L'
\end{array}
$$

"$\Rightarrow$": Zu zeigen ist:

$$
\begin{gathered}
(L \equiv L') \Rightarrow (\forall\, r \,:\, 1 \le r \le m \,:\, (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\\
\mathit{parfor}(L_r) \Rightarrow \neg \mathit{carries}_{\vec{d}}(L_r)))
\end{gathered}
$$

Wir zeigen die inverse Implikation:

$$
\begin{gathered}
(\exists\, r \,:\, 1 \le r \le m \,:\, (\exists\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, \mathit{parfor}(L_r)\\
\wedge\, \mathit{carries}_{\vec{d}}(L_r))) \Rightarrow (L \not\equiv L')
\end{gathered}
$$

Voraussetzung: L_r wird in L' durch eine **parfor**-Schleife L'_r ersetzt und trägt eine Abhängigkeit mit Distanzvektor $\vec{d}$:

$$\begin{array}{ll}
& (\exists\, r \,:\, 1 \leq r \leq m \,:\, (\exists\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, carries_{\vec{d}}(L_r)) \wedge parfor(L_r)) \\
\Rightarrow & \quad \{\ \text{Def. 3.1.6, Def. 3.1.5}\ \} \\
& (\exists\, r \,:\, 1 \leq r \leq m \,:\, (\exists\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, \vec{d} \succ_r \vec{0}) \wedge parfor(L_r)) \\
\Rightarrow & \quad \{\ \text{uniforme Abhängigkeit}\ \} \\
& (\exists\, r \,:\, 1 \leq r \leq m \,:\, (\exists\, \vec{\imath}, \vec{\jmath} \,:\,:\, \vec{\imath} \succ_r \vec{\jmath}) \wedge parfor(L_r)) \\
\Rightarrow & \quad \{\ \text{Def. 3.2.1}\ \} \\
& B(\vec{\imath}) \text{ wird in } L' \text{ nicht notwendig vor } B(\vec{\jmath}) \text{ ausgeführt} \\
\Leftrightarrow & \quad \{\ \text{Def. gültige Ausführungsordnung}\ \} \\
& \text{Ausführungsordnung ungültig} \\
\Leftrightarrow & \quad \{\ \text{Def. 3.2.2}\ \} \\
& L \not\equiv L'
\end{array}$$

Folgendes Beispiel soll die Anwendung von Lemma 3.2.3 veranschaulichen:

Beispiel 3.2.4 Gegeben sei folgendes Programm:

```
for i := 0 to n do
    for j := 0 to n do
        for k := 0 to n do
            a[i, j, k] := a[i − 1, j + 2, k − 3];
            b[i, j, k] := b[i, j − 5, k + 2]
        end
    end
end
```

In diesem Schleifenprogramm existieren zwei uniforme Abhängigkeiten mit den Distanzvektoren $\vec{d_1} = (1, -2, 3)$ und $\vec{d_2} = (0, 5, -2)$. Der Level von $\vec{d_1}$ ist 1, der von $\vec{d_2}$ ist 2. Somit tragen die Schleifen L_1 und L_2 eine Abhängigkeit, nicht jedoch L_3. Mit Lemma 3.2.3 ergibt sich also folgendes äquivalente Schleifenprogramm:

```
seqfor i := 0 to n do
    seqfor j := 0 to n do
        parfor k := 0 to n do
            a[i, j, k] := a[i − 1, j + 2, k − 3];
            b[i, j, k] := b[i, j − 5, k + 2]
        end
    end
end
```

Wir sind nun in der Lage, anhand einer Untersuchung der Abhängigkeiten über die Ersetzung von sequentiellen Schleifen durch parallele zu entscheiden. Wir werden im nächsten Abschnitt ein Verfahren zur Berechnung von Schedules kennenlernen, das diese Tatsache nach einer Transformation des Programmcodes ausnutzt.

4 Berechnung von Schedules

In diesem Abschnitt werden Methoden zur Bestimmung von Schedules vorgestellt.

Wir werden zunächst ein einfaches, schleifenbasiertes Verfahren kennenlernen, das sich nur auf die auftretenden Abhängigkeiten stützt und eine sogenannte *Transformationsmatrix* berechnet.

Als zweites Verfahren wird eine Methode eingeführt, die anhand einer Untersuchung der Indexräume stückweise affine Schedules berechnet. Diese Schedules sind unter noch zu beschreibenden Einschränkungen zeit-optimal.

4.1 Die Hyperebenen-Methode

Bei diesem Verfahren handelt es sich um eine einfache schleifenbasierte Parallelisierungsmethode. Als Ergebnis liefert es eine Transformationsmatrix, in der ein Schedule und eine Allokation enthalten sind. Die Hyperebenen-Methode betrachtet zwar neben den Abhängigkeiten auch den Indexraum, der vorzustellende Algorithmus von Lamport [12] wird diesen aber aus Gründen der Komplexität vernachlässigen. Aus diesem Grund ist der vom Algorithmus implizit berechnete Schedule i. a. nicht zeit-optimal. Ein weiterer Nachteil sind die starken Einschränkungen an die Eingabe-Programme:

- Das Eingabe-Programm ist perfekt verschachtelt.
- Alle Abhängigkeiten sind uniform.

Die Idee des Verfahrens basiert auf einer Partitionierung des Indexraums in Teilmengen von Instanzen, die parallel ausgeführt werden können. Geometrisch betrachtet beschränkt sich das Verfahren dabei auf Teilmengen, die die Form von *Hyperebenen* haben.

Definition 4.1.1 (Hyperebene) Für einen Zeilenvektor $\vec{u} \in \mathbb{R}^m \setminus \{\vec{0}\}$ und eine Konstante $c \in \mathbb{R}$ heißt eine Menge $H \subset \mathbb{R}^m$ mit

$$H = \{\vec{\imath} : \vec{\imath} \in \mathbb{R}^m \wedge \vec{u}\vec{\imath} = c\}$$

eine Hyperebene mit Richtung $\vec{u}$. Durch zwei Konstanten $p, q \in \mathbb{Z}$ wird ein System von Hyperebenen erzeugt mit

$$\begin{aligned}
H_0 &= \{\vec{\imath} : \vec{\imath} \in \mathbb{R}^m \wedge \vec{u}\vec{\imath} = p\} \\
H_1 &= \{\vec{\imath} : \vec{\imath} \in \mathbb{R}^m \wedge \vec{u}\vec{\imath} = p+1\} \\
&\vdots \\
H_{q-p} &= \{\vec{\imath} : \vec{\imath} \in \mathbb{R}^m \wedge \vec{u}\vec{\imath} = p+q\}
\end{aligned}$$

Betrachtet man das perfekt verschachtelte Eingabe-Programm, so gilt hier für die Werte von Index i_1 der äußeren Schleife mit unterer Schleifengrenze l_1 und oberer Schleifengrenze u_1:

$$i_1 = l_1, i_1 = l_1 + 1, \ldots, i_1 = u_1$$

oder

$$\begin{pmatrix} 1 & 0 & \ldots & 0 \end{pmatrix} \begin{pmatrix} i_1 \\ \vdots \\ i_m \end{pmatrix} = l_1, \ \ldots \ , \begin{pmatrix} 1 & 0 & \ldots & 0 \end{pmatrix} \begin{pmatrix} i_1 \\ \vdots \\ i_m \end{pmatrix} = u_1$$

Also definiert ein Schleifenprogramm ein System von Hyperebenen mit Richtung $(1, 0, \ldots, 0)$.

Beispiel 4.1.2 Gegeben sei das Schleifenprogramm

```
for i := 1 to 4 do
    for j := 0 to 5 − i do
        for k := 0 to 2 do
            {Schleifenkörper}
        end
    end
end
```

mit beliebigem Schleifenkörper. Der Indexraum aller Zuweisungen des Schleifenkörpers ist in der Form von Hyperebenen in Abbildung 4 dargestellt.

Unser Ziel ist es, einen neuen Richtungsvektor zu finden, sodaß sich in den damit verbundenen neuen Hyperebenen ausschließlich unabhängige Instanzen von Zuweisungen befinden. Wie zuvor werden diese Hyperebenen

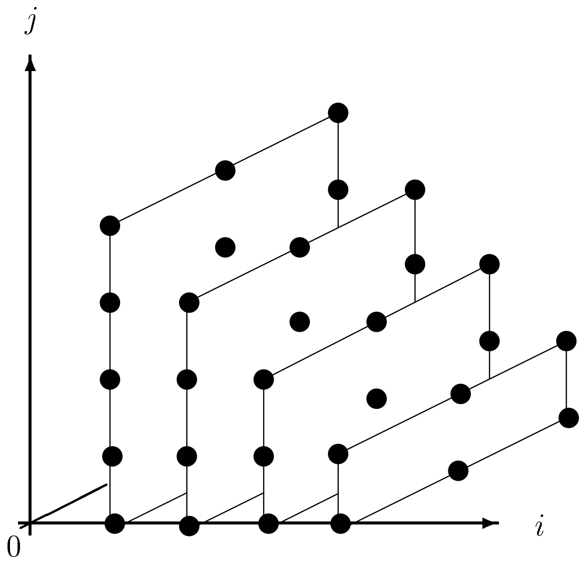

Abbildung 4: Die Hyperebenendarstellung des Indexraums in Beispiel 4.1.2.

durch einen einzigen transformierten Schleifenindex t_1 aufgezählt, alle anderen transformierten Schleifenindizes $t_2, \ldots, t_m$ spannen jeweils eine Dimension einer Hyperebene auf. Da wir die Hyperebenen so konstruieren wollen, daß sie keine abhängigen Instanzen beinhalten, trägt nur der Schleifenindex t_1 Abhängigkeiten. Nach der Grundidee des schleifenbasierten Ansatzes können somit alle Schleifen mit den Schleifenindizes $t_2, \ldots, t_m$ parallel ausgeführt werden.

Mathematisch wird diese Transformation durch eine invertierbare Matrix dargestellt, welche die Schleifenindizes des Eingabe-Programms so umformt, daß die genannte Eigenschaft bzgl. der Hyperebenen erfüllt ist. Wir werden sehen, daß es immer möglich ist, einen solche sogenannte *Transformationsmatrix* zu finden.

Bevor wir den Hauptsatz dieses Abschnitts formulieren können, ist noch theoretische Vorarbeit nötig. Zunächst soll geklärt werden, welche Eigenschaft eine Transformationsmatrix auszeichnet. Um die Semantik des Eingabe-Programms nicht zu zerstören, muß die durch den Schedule bzw. die Abhängigkeiten festgelegte zeitliche Ordnung auch im transformierten Programm gelten. Eine Transformationsmatrix, die diese Eigenschaft erfüllt, heißt*gültig* bezüglich eines Schleifenprogramms.

Definition 4.1.3 (gültige Transformation, $\boldsymbol{valid_L(T)}$) Gegeben sei ein perfekt verschachteltes Schleifenprogramm L mit Zuweisungsmenge Ω_L und

Indexraum D. Eine Transformationsmatrix T heißt *gültig* bezüglich des Schleifenprogramms L, genau dann wenn die durch die Abhängigkeiten festgelegte zeitliche Ordnung auch nach der Anwendung der Transformation auf die Indexvektoren gilt. Formal:

$$(\forall\, S_1, S_2 \,:\, S_1, S_2 \in \Omega_L \,:\, (\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath}, \vec{\jmath} \in D \wedge (S_1, \vec{\imath})\delta(S_2, \vec{\jmath}) \,:\, valid_L(T) \\ \Leftrightarrow T\vec{\imath} \prec T\vec{\jmath}))$$

Lemma 4.1.4 (Charakterisierung gültiger Transformationen) Eine Transformation T eines perfekt verschachtelten Schleifenprogramms L mit ausschließlich uniformen Abhängigkeiten ist genau dann gültig, wenn für jeden Distanzvektor $\vec{d}$ einer uniformen Abhängigkeit $T\vec{d} \succ \vec{0}$ gilt. Formal:

Sei $\mathcal{D}$ die Menge der Distanzvektoren aller uniformen Abhängigkeiten. Dann gilt:

$$(\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, valid_L(T) \Leftrightarrow T\vec{d} \succ \vec{0})$$

Beweis:

$$\begin{array}{ll}
 & valid_L(T) \\
\Leftrightarrow & \{ \text{ Def. 4.1.3 } \} \\
 & (\forall\, S_1, S_2 \,:\, S_1, S_2 \in \Omega_L \,:\, (\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath}, \vec{\jmath} \in D \wedge (S_1, \vec{\imath})\delta(S_2, \vec{\jmath}) \,:\, T\vec{\imath} \prec T\vec{\jmath})) \\
\Leftrightarrow & \{ \text{ Arithmetische Umformung } \} \\
 & (\forall\, S_1, S_2 \,:\, S_1, S_2 \in \Omega_L \,:\, (\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath}, \vec{\jmath} \in D \wedge (S_1, \vec{\imath})\delta(S_2, \vec{\jmath}) \,:\, T(\vec{\jmath} - \vec{\imath}) \succ \vec{0})) \\
\Leftrightarrow & \{ \text{ Sei } \vec{d} = \vec{\jmath} - \vec{\imath} \text{ der Distanzvektor der Abhängigkeit } \} \\
 & (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, T\vec{d} \succ \vec{0})
\end{array}$$

Lemma 4.1.5 (transformierte Distanzvektoren) Ist T eine gültige Transformation eines Schleifenprogramms L in ein Schleifenprogramm L' und $\mathcal{D}$ die Menge der Distanzvektoren der Abhängigkeiten in L, so gilt für die Menge $\mathcal{D}'$ der Distanzvektoren der Abhängigkeiten in L':

$$\mathcal{D}' = \{T\vec{d} : \vec{d} \in \mathcal{D}\}$$

Beweis: Im folgenden bezeichne B' die Iterationen des Schleifenprogramms L'.

$$
\begin{array}{ll}
 & \vec{d} \in \mathcal{D} \\
\Leftrightarrow & \{ \text{ Def. 3.1.4 } \} \\
 & (\exists\, \vec{\imath}, \vec{\jmath} : : \vec{d} = \vec{\jmath} - \vec{\imath} \wedge B(\vec{\imath})\delta B(\vec{\jmath})) \\
\Leftrightarrow & \{ \text{ Definiere } \vec{k} = T\vec{\imath} \text{ und } \vec{l} = T\vec{\jmath} \text{ und mit } valid_L(T) = true \ \} \\
 & B'(\vec{k})\delta B'(\vec{l}) \\
\Rightarrow & \{ \text{ Def. 3.1.4 } \} \\
 & (\exists\, \vec{d'} : \vec{d'} \in \mathcal{D}' : \vec{d'} = \vec{l} - \vec{k}) \\
\Leftrightarrow & \{ \ \vec{k} = T\vec{\imath} \text{ und } \vec{l} = T\vec{\jmath} \ \} \\
 & (\exists\, \vec{d'} : \vec{d'} \in \mathcal{D}' : \vec{d'} = T\vec{\jmath} - T\vec{\imath}) \\
\Leftrightarrow & \{ \ \vec{d} = \vec{\jmath} - \vec{\imath} \ \} \\
 & (\exists\, \vec{d'} : \vec{d'} \in \mathcal{D}' : \vec{d'} = T\vec{d})
\end{array}
$$

Da wegen der Gültigkeit der Transformation eine Abhängigkeit zwischen zwei Iterationen in L' genau dann besteht, wenn die gleiche Abhängigkeit in L existiert, sind damit alle Abhängigkeiten in L' erfaßt.

Eine Transformationsmatrix, deren Determinante ± 1 ist, erleichtert den Prozeß der Programmkonstruktion, wie wir im letzten Abschnitt sehen werden. Eine Matrix mit dieser Eigenschaft heißt *unimodular*.

Wir sind nun in der Lage, den Hauptsatz dieses Abschnitts zu formulieren, der im Beweis bereits den Algorithmus zur Berechnung der Transformationsmatrix enthält [1].

Satz 4.1.6 (Existenz einer Transformation) Für ein gegebenes perfekt verschachteltes Schleifenprogramm $L = (L_1, \ldots, L_m)$ mit ausschließlich uniformen Abhängigkeiten existiert eine gültige unimodulare Transformation T in ein Schleifenprogramm $L' = (L'_1, \ldots, L'_m)$, sodaß die Schleifen $L'_2, \ldots, L'_m$ parallel ausgeführt werden können.

Beweis: Wir unterscheiden zunächst zwei Fälle:

$\mathcal{D} = \emptyset$:

$$\begin{array}{ll} & \mathcal{D} = \emptyset \\ \Rightarrow & \{ \text{ Lemma 3.2.3 } \} \\ & L_1, \ldots, L_m \text{ können parallel ausgeführt werden} \end{array}$$

Also erfüllt $T := \mathcal{I}_m$ die geforderten Eigenschaften.

$\mathcal{D} \neq \emptyset$: Im Beweis benötigen wir folgende Definition einer Partitionierung von $\mathcal{D}$:

$$(\forall\, r \,:\, 1 \le r \le m \,:\, \mathcal{D}_r = \{\vec{d} \,:\, \vec{d} \in \mathcal{D} \wedge level(\vec{d}) = r\})$$

Da $\mathcal{D}$ die Vereinigung der Mengen $\mathcal{D}_r$ darstellt und nicht die leere Menge ist, folgt, daß zumindest eine der Mengen $\mathcal{D}_r$ ungleich der leeren Menge ist. Wir nehmen nun an, daß eine Matrix T existiert, für die gilt

$$(valid_L(T) \wedge L'_2, \ldots, L'_m \text{ sind } \textbf{parfor}\text{-Schleifen})$$

und zeigen durch Äquivalenzumformungen, daß T konstruierbar ist:

$$\begin{array}{ll} & (valid_L(T) \wedge L'_2, \ldots, L'_m \text{ sind } \textbf{parfor}\text{-Schleifen}) \\ \Leftrightarrow & \{ \text{ Def. 4.1.3 } \} \\ & (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, T\vec{d} \succ \vec{0}) \wedge L'_2, \ldots, L'_m \text{ sind } \textbf{parfor}\text{-Schleifen} \\ \Leftrightarrow & \{ \text{ Lemma 3.2.3 } \} \\ & (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, T\vec{d} \succ \vec{0} \wedge (\forall\, i \,:\, 2 \le i \le m \,:\, \neg carries_d(L_i))) \\ \Leftrightarrow & \{ \text{ Lemma 4.1.5 und Def. 3.1.6 } \} \\ & (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, T\vec{d} \succ \vec{0} \wedge level(T\vec{d}) = 1) \\ \Leftrightarrow & \{ \text{ Def. 3.1.5 } \} \\ & (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, T\vec{d} \succ_1 \vec{0}) \\ \Leftrightarrow & \{ \text{ Definiere } \vec{t} \text{ als erste Zeile in } T \} \\ & (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, \vec{t}\vec{d} \ge 1) \\ \Leftrightarrow & \{ \text{ Matrixprodukt } \} \\ & (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D} \,:\, (\sum\, i \,:\, 1 \le i \le m \,:\, t_i d_i) \ge 1) \\ \Leftrightarrow & \{ \text{ Definition von } \mathcal{D}_r \text{ und } level \} \\ & (\forall\, r \,:\, 1 \le r \le m \,:\, (\forall\, \vec{d} \,:\, \vec{d} \in \mathcal{D}_r \,:\, (\sum\, i \,:\, r \le i \le m \,:\, t_i d_i) \ge 1)) \end{array}$$

$\Leftrightarrow$ { Auflösen nach t_r, wobei $d_r > 0$ }

$$(\forall\, r\ :\ 1 \leq r \leq m\ :\ (\forall\, \vec{d}\ :\ \vec{d} \in \mathcal{D}_r\ :\\ t_r \geq (1 - (\sum\ i\ :\ r+1 \leq i \leq m\ :\ t_i d_i))/d_r)) \quad (1)$$

Ist $\mathcal{D}_r$ nicht leer, so existiert damit eine untere Schranke für t_r. Im anderen Fall ist t_r unbeschränkt und kann einen beliebigen Wert annehmen. Die Aussage in (1) kann als Konstruktionsvorschrift verwendet werden, indem man sukzessive für $r = m, \ldots, 1$ die Werte der Einträge t_r jeweils gleich der entsprechenden unteren Schranke setzt.

Somit besitzt also jede Matrix, deren erste Zeile das Ungleichungssystem (1) erfüllt, aufgrund der Äquivalenz im Beweis die in der Annahme geforderten Eigenschaften. Sei also $\vec{t} = (t_1, \ldots, t_m)$ ein solcher Vektor. Es bleibt noch zu zeigen, daß $\vec{t}$ zu einer unimodularen Matrix ergänzt werden kann. Mit $g = \gcd(t_1, \ldots, t_m)$ erfüllt auch der Vektor $\vec{t'} = (t_1/g, \ldots, t_m/g)$ das Ungleichungssystem. Mittels Methoden der Zahlentheorie [1] ist es nun möglich, den Vektor $\vec{t'}$ mittels Echelon-Reduktion zu einer unimodularen Matrix T zu ergänzen, die damit sämtliche geforderten Eigenschaften besitzt.

Anmerkung 4.1.7 Für die berechnete Matrix T gilt laut Beweis:

$$(\forall\, \vec{d}\ ::\ T\vec{d} \succ_1 \vec{0})$$

Mit dem ersten Zeilenvektor $\vec{t}$ gilt damit auch $\vec{t}\vec{d} > \vec{0}$. Mit der Anmerkung 3.1.9 ist also die Abbildung

$$\Theta(\vec{\imath}) = \vec{t}\vec{\imath}$$

ein Schedule für das gegebene Schleifenprogramm.

Der Beweis des Satzes 4.1.6 zeigt, daß es sogar beliebig viele solcher Transformationen gibt, denn für die Einträge der erste Zeile der Matrix existieren lediglich untere Schranken. Unser Ziel ist es, ein laufzeitoptimales Programm zu erhalten. Die Laufzeit eines Programms, dessen innere Schleifen alle parallel ausgeführt werden, wird (abgesehen vom Kommunikationsaufwand) ausschließlich durch die Grenzen der äußeren sequentiellen Schleife bestimmt.

Allgemein ist die Anzahl von Iterationen bei gegebenem Schedule Θ für das parallele Programm gleich

$$(\max\ \vec{\imath}\ :\ \vec{\imath} \in D_{\Omega_L}\ :\ \Theta(\vec{\imath})) - (\min\ \vec{\imath}\ :\ \vec{\imath} \in D_{\Omega_L}\ :\ \Theta(\vec{\imath})) + 1$$

Mit Anmerkung 4.1.7 kann damit die Anzahl von Iterationen eines transformierten Programms berechnet werden. Sei dazu t der erste Zeilenvektor einer Transformationsmatrix wie in Satz 4.1.6. Dann ist diese Anzahl gleich folgendem Ausdruck:

$$(\max\ \vec{\imath}\ :\ \vec{\imath} \in D_{\Omega_L}\ :\ \vec{t}\vec{\imath}) - (\min\ \vec{\imath}\ :\ \vec{\imath} \in D_{\Omega_L}\ :\ \vec{t}\vec{\imath}) + 1$$

Zusammenfassend ist also folgendes Optimierungsproblem zu lösen:

Optimierungsproblem der Hyperebenen-Methode

Gegeben: Ein perfekt verschachteltes Schleifenprogramm $L = (L_1, \ldots, L_m)$ mit einer Menge $\mathcal{D}$ von Distanzvektoren uniformer Abhängigkeiten.

Gesucht: Ein Vektor $\vec{t} = (t_1, \ldots, t_m)$ mit folgenden Eigenschaften:

1. $\gcd(t_1, \ldots, t_m) = 1$
2. $(\forall\ \vec{d}\ :\ \vec{d} \in \mathcal{D} \wedge \vec{d} = (d_1, \ldots, d_m)\ :\ (\sum\ k\ :\ 1 \le k \le m\ :\ t_k d_k) \ge 1)$
3. Der Wert des Ausdrucks

$$(\max\ \vec{\imath}\ :\ \vec{\imath} \in D_{\Omega_L}\ :\ \vec{t}\vec{\imath}) - (\min\ \vec{\imath}\ :\ \vec{\imath} \in D_{\Omega_L}\ :\ \vec{t}\vec{\imath}) + 1$$

ist minimal.

Die Lösung dieses Minimierungsproblems gestaltet sich allerdings sehr aufwendig. Dies gilt vor allem dann, wenn die Grenzen innerer Schleifenindizes von äußeren Schleifenindizes abhängen.

Deshalb stellen wir eine Methode vor, welche die dritte Forderung außer acht läßt [12]. Sie beruht auf der empirischen Aussage, daß bei der Wahl der jeweils unteren Schranken des Ungleichungssystems (1) im Beweis des Satzes 4.1.6 meistens eine Transformation für ein laufzeitoptimales Programm entsteht. Das Verfahren besteht aus den beiden Schritten, die auch im Beweis von Lemma 4.1.6 durchgeführt werden. Zunächst wird ein Vektor $\vec{t}$ mit den oben beschriebenen Eigenschaften konstruiert, der dann zu einer unimodularen Matrix ergänzt wird. Zur Konstruktion der Matrix wird jedoch ein einfacheres Verfahren verwendet, als die im Beweis von Satz 4.1.6 verwendete Echelon-Reduktion.

Algorithmus 4.1.8 (Hyperebenen-Methode von Lamport)
Eingabe: Eine Menge $\mathcal{D} \neq \emptyset$ von Distanzvektoren uniformer Abhängigkeiten

Ausgabe: Eine gültige Transformation T

Methode:

1. Konstruktion des ersten Zeilenvektors $\vec{t}$ von T:

   { Konstruiere die Mengen $\mathcal{D}_r$ }
   for $r := 1$ **to** m **do**
   $\quad \mathcal{D}_r := \{\vec{d} : \vec{d} \in \mathcal{D} \wedge level(d) = r\}$
   end;
   { Berechne t_r aus den Werten $t_{r+1}, \ldots, t_m$ }
   for $r := m$ **to** 1 **step** -1 **do**
   $\quad$ **if** $\mathcal{D}_r = \emptyset$ **then**
   $\quad\quad t_r := 0$
   $\quad$ **else**
   $\quad\quad t_r := \left\lceil (\max\ \vec{d}\ :\ \vec{d} \in \mathcal{D}_r\ :\ (1 - t_{r+1}d_{r+1} - \ldots - t_m d_m)/d_r) \right\rceil$
   $\quad$ **end**
   end

2. Ergänzung zu einer unimodularen Matrix $T = (t_{ij})$, wobei $1 \leq i, j \leq m$:

   $r := m$;
   { Suche das erste $t_r \neq 0$ in der Sequenz $(t_m, \ldots, t_1)$ }
   while $t_r = 0$ **do**
   $\quad r := r - 1$
   end;
   k:=r;
   { Setze T' gleich der Einheitsmatrix $\mathcal{I}_{m-1}$ }
   $T' := \mathcal{I}_{m-1}$;
   füge einen Nullvektor als k-te Spalte in T' ein;
   { Ergänze T' um $\vec{t}$ als erste Zeile }
   $T := \begin{pmatrix} \vec{t} \\ T' \end{pmatrix}$

Terminierung und Korrektheit:

Terminierung: Die Terminierung der beiden Schleifen im ersten Schritt des Algorithmus ist offensichtlich, da im Schleifenkörper weder die Laufvariable noch die Schleifengrenzen verändert werden. Um die Terminierung der **while**-Schleife im zweiten Schritt zu sehen, muß die Existenz eines Eintrags $t_r \neq 0$ gezeigt werden. Da wir davon ausgehen, daß die Menge $\mathcal{D}$ nicht leer ist, existiert eine Menge $\mathcal{D}_k \neq \emptyset$, die diese Eigenschaft als erste in der Sequenz $\mathcal{D}_m, \dots, \mathcal{D}_1$ besitzt. Also folgt nach der Berechnungsvorschrift im ersten Schritt, daß $t_k = 1$ gilt. Somit terminiert auch die **while**-Schleife.

Korrektheit: Es gilt, die beiden ersten Eigenschaften des Optimierungsproblems zu zeigen. Da die Berechnung der Einträge des Vektors $\vec{t}$ der Konstruktion im Beweis von Satz 4.1.6 folgt, erfüllt $\vec{t}$ die zweite geforderte Eigenschaft. Für den ersten Eintrag $t_r \neq 0$ in der Sequenz $(t_m, \dots, t_1)$ gilt nach der Berechnungsvorschrift $t_r = 1$. Damit ist

$$\gcd(t_1, \dots, t_m) = 1,$$

wodurch die erste geforderte Eigenschaft erfüllt wird.

Wird in der Matrix T die k-te Spalte und k-te Zeile gestrichen, so ergibt sich die Einheitsmatrix $\mathcal{I}_{m-1}$. Die k-te Spalte von T hat die Form $(1, 0, \dots, 0)$. Damit ergibt sich nach dem Entwicklungssatz von Laplace für die Determinante von T:

$$|T| = \pm 1$$

Somit ist T unimodular.

Anmerkung 4.1.9 Häufig wird in der Definition eines Schedules anders als in Definition 3.1.8 die Nicht-Negativität der entsprechenden Abbildung gefordert. Da durch die Hyperebenen-Methode lineare Schedules berechnet werden, ist diese zusätzliche Eigenschaft des Schedules i. a. nicht gegeben. Um positive Schedules zu erhalten, ist ein Übergang von linearen Schedules zu affinen Schedules durch die Ergänzung eines konstanten Anteils notwendig, der sich aus dem linearen Schedule und den Schleifengrenzen des Eingabe-Programms ableitet und eventuell von Beziehungen zwischen Strukturparametern abhängig ist. In der Implementierung der Hyperebenen-Methode ist es möglich, Schedules mit dieser Option zu berechnen.

Anmerkung 4.1.10 In Definition 3.1.4 wurden Distanzvektoren als nicht negative Vektoren definiert. Die Ursache dafür ist in der Festlegung von Schleifenprogrammen zu finden, für die wir eine konstante Schrittweite von $+1$ gefordert haben. Im Beweis von Satz 4.1.6 wird zur Herleitung von (1) die Positivität von Distanzvektoren ausgenutzt, um für jeden Eintrag des ersten Spaltenvektors eine untere Schranke zu berechnen. Erlaubt man in der Formulierung von Schleifenprogrammen auch die Schrittweite -1, so ergeben sich negative Distanzvektoren. Die Hyperebenen-Methode läßt sich auch auf derartige Schleifenprogramme anwenden, da es im Beweis von Satz 4.1.6 im Schritt *Auflösen nach* t_r lediglich notwendig ist, daß d_r ungleich Null ist. Insbesondere ist also auch d_r kleiner Null möglich. In diesem Fall wird für t_r eine obere Schranke berechnet, während für positive Distanzvektoren stets eine untere Schranke ermittelt wird. Diese Erweiterung des Verfahrens wurde auch in die Implementierung der Hyperebenen-Methode aufgenommen.

Abschließend wollen wir die Hyperebenen-Methode für ein Beispiel betrachten, aus dem auch hervorgeht, daß der Algorithmus, wie bereits erwähnt, nicht immer optimale Ergebnisse liefert:

Beispiel 4.1.11 Gegeben sei folgendes Schleifenprogramm:

```
for i := 0 to 10 do
    for j := 0 to 5 do
        a[i,j] := a[i-1,j-1];      {S1}
        b[i,j] := a[i-1,j-2]       {S2}
    end
end
```

Das Schleifenprogramm enthält zwei uniforme Abhängigkeiten mit den Distanzvektoren $\vec{d_1} = \begin{pmatrix} 1 & 1 \end{pmatrix}$ und $\vec{d_2} = \begin{pmatrix} 1 & 2 \end{pmatrix}$. Der Indexraum des Schleifenprogramms und die auftretenden Abhängigkeiten sind in Abbildung 5 dargestellt. Für den Level der Abhängigkeiten gilt:

$$\begin{aligned} level(\vec{d_1}) &= 1 \\ level(\vec{d_2}) &= 1 \end{aligned}$$

Daraus ergeben sich die Mengen $\mathcal{D}_r$ wie folgt:

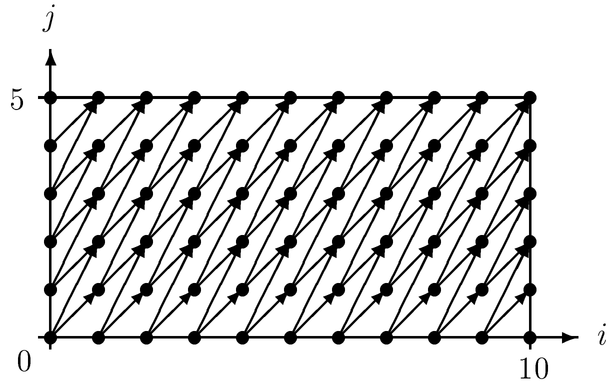

Abbildung 5: Der Indexraum des Schleifenprogramms aus Beispiel 4.1.11.

$$\mathcal{D}_1 = \{\vec{d_1}, \vec{d_2}\}$$
$$\mathcal{D}_2 = \emptyset$$

Mit diesen Mengen gilt dann für den ersten Zeilenvektor der Transformationsmatrix T: $\vec{t} = \begin{pmatrix} 1 & 0 \end{pmatrix}$. Die Ergänzung zu einer unimodularen Matrix liefert schließlich als Ergebnis:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Somit wird also keine Transformation vorgenommen. Lediglich die innere Schleife wird parallel ausgeführt. Die Laufzeit des parallelen Programms beträgt 11 Zeiteinheiten. Berücksichtigt man jedoch auch die Minimierungsbedingung, gilt es den Ausdruck

$$10t_1 + 5t_2$$

mit den Nebenbedingungen, die durch die erste und zweite Eigenschaft festgelegt sind, zu minimieren. Man erhält dann für den Vektor $\vec{t}$:

$$\vec{t} = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

Dies führt zu der Transformationmatrix:

$$T = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Eine derartige Transformation tauscht die innere mit der äußeren Schleife, weshalb die Laufzeit des parallelen Programms nun lediglich 6 Zeiteinheiten beträgt.

4.2 Die Methode von Feautrier

Die zweite Methode zur Parallelisierung verfolgt einen statementbasierten Ansatz. Außer den anfangs genannten Einschränkungen fordert sie keine zusätzlichen Eigenschaften der Eingabe-Programme. Im Vergleich zur Hyperebenen-Methode entfällt also die Beschränkung auf die Uniformität der Abhängigkeiten und die perfekte Verschachtelung.

Anders als bei der Hyperebenen-Methode wird hier für jede Zuweisung ein eigenständiger Schedule berechnet. Wie bereits erwähnt, berücksichtigt dieses Verfahren auch den Indexraum einer Zuweisung. Da dieser von Strukturparametern abhängig sein kann, gilt dies auch für den berechneten Schedule.

Wir leiten zunächst ein Optimierungsproblem her, dessen Lösung den Schedule einer Zuweisung darstellt. Darauf soll eine Möglichkeit zur Lösung des Optimierungsproblems vorgestellt werden.

4.2.1 Herleitung des Optimierungsproblems

Die Grundidee des Verfahrens besteht darin, zunächst eine Menge von Schedules zu berechnen und darauf nach dem Kriterium der kürzesten Laufzeit das Minimum zu suchen. Da die Vorstellung dieser Methode die Implementierung eines Algorithmus ermöglichen soll, werden wir auch auf die Probleme dieser Minimierung eingehen.

Die Grundlage für die Konstruktion der Menge von Schedules leitet sich aus der Definition des Schedules selbst ab. Wie bereits angegeben, besitzt ein Schedule Θ für zwei abhängige Zuweisungen $S, T \in \Omega_L$ eines Schleifenprogramms L folgende Eigenschaft:

$$(\forall\, \vec{\imath}, \vec{\jmath} : \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \wedge (S, \vec{\imath})\delta(T, \vec{\jmath}) \,:\, \Theta(S, \vec{\imath}) < \Theta(T, \vec{\jmath}))$$

Der Einfachheit halber nehmen wir für die Abarbeitungszeit einer jeden Instanz eine Zeiteinheit an. Damit gilt für den Schedule:

$$(\forall\, \vec{\imath}, \vec{\jmath} : \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \wedge (S, \vec{\imath})\delta(T, \vec{\jmath}) \,:\, \Theta(S, \vec{\imath}) \leq \Theta(T, \vec{\jmath}) + 1) \qquad (2)$$

Die Grundform des Verfahrens von Feautrier beschränkt sich auf eindimensionale affine Schedules mit der allgemeinen Form:

$$\Theta(S, \vec{\imath}) = \vec{\alpha}_S \vec{\imath} + \vec{\beta}_S \vec{n} + \gamma_S$$

Dabei bezeichnet $\vec{n}$ den Vektor der Strukturparameter, $\vec{\alpha}_S$ und $\vec{\beta}_S$ die entsprechenden zu bestimmenden Koeffizientenvektoren und γ_S eine ebenfalls noch unbekannte Konstante.

Zur Darstellung des Indexraums D_S einer Zuweisung $S \in \Omega_L$ verwenden wir folgende Notation für die zugehörigen Ungleichungen:

$$D_S = \{\, \vec{\imath} : (\forall\, k \,:\, 1 \leq k \leq m_S \,:\, \vec{a}_S^k \begin{pmatrix} \vec{\imath} \\ \vec{n} \end{pmatrix} + b_S^k \geq 0) \,\} \tag{3}$$

Die Anzahl der Ungleichungen zur Beschreibung von D_S wird von m_S angegeben. Der Vektor $\vec{a}_S^k$ ist der Koeffizientenvektor der Schleifeninidizes und Sturkturparameter, b_S^k entspricht dem konstanten Anteil. Analog notieren wir die Polyeder P_e, welche die Indexpunkte im Ziel einer Abhängigkeit beschreiben, durch

$$P_e = \{\, \vec{\jmath} : (\forall\, k \,:\, 1 \leq k \leq m_e \,:\, \vec{c}_e^k \begin{pmatrix} \vec{\jmath} \\ \vec{n} \end{pmatrix} + d_e^k \geq 0) \,\}$$

Hier wird die Anzahl der Ungleichungen durch m_e wiedergegeben.

Mit folgendem Lemma der linearen Programmierung läßt sich ein erster Zusammenhang zwischen Schedule und Indexraum herstellen:

Lemma 4.2.1 (Affine Form von Farkas' Lemma) Sei $\mathcal{D}$ ein nicht leeres Polyeder, das durch die n Ungleichungen

$$\{\, \vec{a}_i \vec{\imath} + b_i \geq 0 \,:\, 1 \leq i \leq n \,\}$$

beschrieben ist. Dann ist eine affine Abbildung $\psi : \mathcal{D} \to \mathbb{Q}$ genau dann in ihrem gesamten Bild nicht negativ, wenn ψ eine positive Linearkombination in diesen Ungleichungen ist, also ψ folgende Form hat:

$$\psi(x) = \lambda_0 + (\sum\, i \,:\, 1 \leq i \leq n \wedge \lambda_i \geq 0 \,:\, \lambda_i(\vec{a}_i \vec{\imath} + b_i))$$

Die Koeffizienten λ_i werden als Farkas-Multiplikatoren bezeichnet.

Wenden wir diese Aussage auf den zu bestimmenden affinen Schedule Θ an, so erhalten wir eine Darstellung des Schedules in Abhängigkeit der Ungleichungen der Indexräume von Zuweisungen.

Definition 4.2.2 (Prototyp eines Schedules) Sei $S \in \Omega_L$ eine Zuweisung eines Schleifenprogramms L und sei D_S durch eine Menge von Ungleichungen wie in Gleichung (3) gegeben. Die Anwendung von Lemma 4.2.1 auf einen Schedule Θ ergibt, daß dieser für einen Indexvektor $\vec{\imath} \in D_S$ genau dann positive Werte besitzt, wenn Farkas-Multiplikatoren μ_S^k mit folgender Eigenschaft existieren:

$$\Theta(S,\vec{\imath}) = \mu_S^0 + (\sum\ k\ :\ 1 \leq k \leq m_S\ :\ \mu_S^k(\vec{a}_S^k \begin{pmatrix} \vec{\imath} \\ \vec{n} \end{pmatrix} + b_S^k)) \tag{4}$$

Eine derartige Darstellung des Schedules bezeichnen wir als den *Prototypen eines Schedules.*

Anmerkung 4.2.3 Der Indexraum einer Zuweisung ist neben den Schleifenindizes auch von der Größe der Strukturparameter abhängig. Aus diesem Grund werden Strukturparameter wie Schleifenindizes ohne Grenzen behandelt. Wir fordern allerdings für Strukturparameter, daß sie stets positiv sind. Dies wird durch die zusätzlichen Ungleichungen

$$\vec{n} \geq \vec{0}$$

zur Beschreibung eines Indexraums ausgedrückt. Die Ungleichungen zur Beschreibung der Polyeder P_e sind ebenfalls um diese Ungleichungen zu ergänzen. Eine Einschränkung stellt dies nicht dar, weil ein negatives Vorzeichen stets in den Ausdruck, in dem der Strukturparameter steht, verlagert werden kann. Wir benötigen diese Forderung vor allem für Zuweisungen ohne umgebende Schleifen, also Zuweisungen mit 0-dimensionalem Indexraum. Mit den zusätzlichen Ungleichungen für eine solche Zuweisung ergibt sich damit für ein Schleifenprogramm mit r Strukturparametern $n_1, \ldots, n_r$ folgender Prototyp:

$$\Theta(S,0) = \mu_S^0 + \mu_S^1 n_1 + \ldots + \mu_S^r n_r$$

Der einzige Indexpunkt soll mit 0 bezeichnet werden. Bei Zuweisungen mit ein- oder mehrdimensionalem Indexraum sind diese Ungleichungen meist nicht notwendig, da Strukturparameter in Schleifengrenzen häufig wie in

$$\textbf{for } i := 0 \textbf{ to } n$$

auftreten und damit implizit größer oder gleich Null gesetzt werden. Allgemein sollten zur Beschreibung des Indexraums sämtliche redundanten Ungleichungen eliminiert werden, um damit die Anzahl der Farkas-Multiplikatoren zu reduzieren.

Jeder Abhängigkeit mit Kante e im allgemeinen Abhängigkeitsgraphen des Eingabe-Programms läßt sich mittels der Ungleichung (2) eine Verzögerung Δ_e zuordnen:

$$(\forall\, \vec{\imath}, \vec{\jmath} : (\vec{\imath}, \vec{\jmath}) \in R_e : \Delta_e(\vec{\imath}, \vec{\jmath}) = \Theta(\delta(e), \vec{\jmath}) - \Theta(\sigma(e), \vec{\imath}) - 1)$$

Nutzt man zur Darstellung der Abhängigkeiten die anfangs eingeführte h-Transformation und das Polyeder P_e, ergibt sich für die Verzögerung:

$$(\forall\, \vec{\jmath} : \vec{\jmath} \in P_e : \Delta_e(\vec{\jmath}) = \Theta(\delta(e), \vec{\jmath}) - \Theta(\sigma(e), h(\vec{\jmath})) - 1)$$

Aufgrund der Definition eines Schedules ist Δ_e stets positiv und wegen der Affinität des Schedules und der h-Transformation selbst affin. Somit erfüllt Δ_e die Anforderungen zur Anwendung des obigen Lemmas, wodurch der Zusammenhang von Schedule und Abhängigkeiten hergestellt wird. Wieder existieren Farkas-Multiplikatoren λ_e^k, sodaß gilt:

$$\begin{aligned}(\forall\, \vec{\jmath} : \vec{\jmath} \in P_e : \Theta(\delta(e), \vec{\jmath}) - \Theta(\sigma(e), h(\vec{\jmath})) - 1 = \\ \lambda_e^0 + (\textstyle\sum\, k : 1 \leq k \leq m_e : \lambda_e^k(\vec{c}_e^{\,k} \begin{pmatrix} \vec{\jmath} \\ \vec{n} \end{pmatrix} + d_e^k)))\end{aligned} \tag{5}$$

Benutzt man zur Darstellung der Schedules in Gleichung (5) die oben definierten Prototypen und führt nach Einsetzen der Prototypen einen Koeffizientenvergleich für alle Schleifenindizes, Strukturparameter und den konstanten Anteil durch, erhält man für jede Abhängigkeit e ein Gleichungssystem in den Unbekannten μ_S^k und λ_e^k. Dieses Gleichungssystem ist erfüllt, falls die eingesetzten Prototypen Schedules sind. Somit schränkt jedes solche Gleichungssystem die Werte der Farkas-Multiplikatoren in den Prototypen derart ein, daß die Prototypen bezüglich e die Scheduleeigenschaft besitzen.

Anmerkung 4.2.4 Bei einer uniformen Abhängigkeit ist die Anwendung von Farkas' Lemma auf die Verzögerung nicht notwendig, wenn diese Abhängigkeit zwischen zwei Instanzen derselben Zuweisung auftritt. Das liegt daran, daß die zugehörige Verzögerung stets konstant ist und nicht von den Indexvektoren abhängig ist. Hier genügt es also, nur für den konstanten Anteil einen Koeffizientenvergleich durchzuführen. Auf der rechten Seite der erzeugten Gleichung steht lediglich der Farkas-Multiplikator λ_e^0 als Repräsentant für die konstante Verzögerung. Dieser Spezialfall wird in der Implementierung ebenfalls berücksichtigt.

Beispiel 4.2.5 Wir betrachten folgendes Schleifenprogramm:

```
for i := 0 to 5 do
    for j := i to i + 9 do
        a[i, j] := a[i − 1, j]+a[i − 2, j + 2]        {S1}
    end
    b[i] := a[i, i];                                {S2}
end
```

Für die Indexräume D_{S_1} und D_{S_2} gelten die Ungleichungen:

$$D_{S_1} = \{(i, j) \,:\, 0 \le i \le 5 \wedge i \le j \le i + 9\}$$
$$D_{S_2} = \{i \,:\, 0 \le i \le 5\}$$

Es existieren drei Abhängigkeiten: $e_1 = S_1 \delta S_1$, $e_2 = S_1 \delta S_1$, die jeweils uniform sind, und $e_3 = S_1 \delta S_2$ mit den folgenden h-Transformationen und Polyedern P_{e_i}:

$$\begin{array}{lll} e_1: & h_{e_1}(i, j) = (i - 1, j) & P_{e_1} = \{(i, j) \,:\, 1 \le i \le 5 \wedge i \le j \le i + 8\} \\ e_2: & h_{e_2}(i, j) = (i - 2, j + 2) & P_{e_2} = \{(i, j) \,:\, 2 \le i \le 5 \wedge i \le j \le i + 5\} \\ e_3: & h_{e_3}(i) = (i, i) & P_{e_3} = \{i \,:\, 0 \le i \le 5\} \end{array}$$

In Abbildung 6 ist der Indexraum beider Zuweisungen jeweils mit den verschiedenen Abhängigkeiten dargestellt.

Wir stellen zunächst die Prototypen der Zuweisungen gemäß Gleichung (4) auf:

$$\Theta_{S_1}(i, j) = \mu_1^0 + \mu_1^1 i + \mu_1^2(5 - i) + \mu_1^3(j - i) + \mu_1^4(i - j + 9)$$
$$\Theta_{S_2}(i) = \mu_2^0 + \mu_2^1 i + \mu_2^2(5 - i)$$

Mit Anmerkung 4.2.4 und der Uniformität der Abhängigkeiten e_1 und e_2 vereinfacht sich die Konstruktion der Gleichungssysteme. Für diese beiden Abhängigkeiten erhält man direkt die Gleichungen

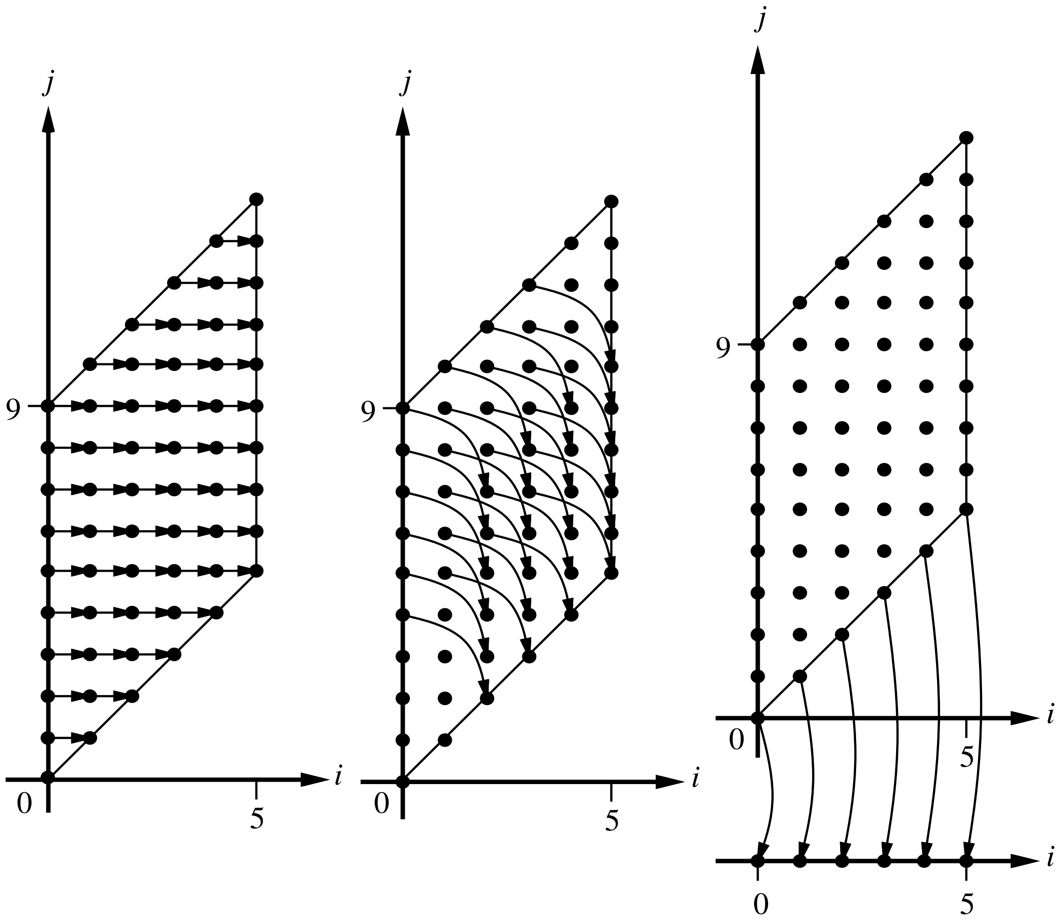

Abbildung 6: Indexraum und Abhängigkeiten aus Beispiel 4.2.5.

$$\begin{array}{ll} e_1: & \mu_1^1 - \mu_1^2 - \mu_1^3 + \mu_1^4 - 1 = \lambda_1^0 \\ e_2: & 2\mu_1^1 - 2\mu_1^2 - 4\mu_1^3 + 4\mu_1^4 - 1 = \lambda_2^0 \end{array}$$

Für die dritte Abhängigkeit gilt aufgrund von Gleichung (5):

$$\begin{array}{ll} & \mu_2^0 + \mu_2^1 i + \mu_2^2(5-i) \\ - & (\mu_1^0 + \mu_1^1 i + \mu_1^2(5-i) + \mu_1^3(i-i) + \mu_1^4(i-i+9)) - 1 \\ = & \lambda_3^0 + \lambda_3^1 i + \lambda_3^2(5-i) \end{array}$$

Der Koeffizientenvergleich für i,j und den konstanten Anteil liefert folgendes

Gleichungssystem für e_3:

$$\begin{aligned}
\mu_2^1 - \mu_2^2 - \mu_1^1 + \mu_1^2 &= \lambda_3^1 - \lambda_3^2 \\
0 &= 0 \\
\mu_2^0 + 5\mu_2^2 - \mu_1^0 - 5\mu_1^2 - 9\mu_1^4 - 1 &= \lambda_3^0 + 5\lambda_3^2
\end{aligned}$$

Die Lösungen der Vereinigung dieser Gleichungssysteme, die je Abhängigkeit erzeugt wurden, ergeben damit eine Menge von Werten für die Farkas-Multiplikatoren. Setzt man eine solche Lösung in die Prototypen ein, erhält man einen affinen, nicht negativen Schedule für das Eingabe-Programm. Die Menge von Schedules, die durch das Einsetzen aller Lösungen entsteht, bezeichnen wir mit $\mathcal{G}$.

Wie bei der Hyperebenen-Methode sind wir auch hier an laufzeitoptimalen Schedules interessiert. Wir suchen also einen „minimalen" Schedule in $\mathcal{G}$, wobei der Begriff der Minimalität noch festzulegen ist. Um diesen Prozeß zu vereinfachen, formen wir das erhaltene Gleichungssystem weiter um. Nach einer Heuristik [9] existieren in dem hergeleiteten Gleichungssystem etwa zweimal soviele Unbekannte wie Gleichungen. Somit bildet der Lösungsraum, falls er existiert, eine sehr große Menge, aus welcher der minimale Schedule zu wählen ist. Es ist jedoch möglich die Anzahl von Unbekannten zu reduzieren, ohne den Lösungsraum zu verändern. Die Grundlage dafür bildet die Nicht-Negativitäts-Bedingung für Farkas-Multiplikatoren. Wir führen dazu eine Abwandlung der Elimination nach Gauss-Jordan [14] durch. Wie im bekannten Verfahren löst man nach einer Unbekannten x, hier den Farkas-Multiplikatoren, auf und ersetzt diese durch den gewonnenen Ausdruck e in den übrigen Gleichungen. Da die Farkas-Multiplikatoren nicht negativ sind, kann man nun wegen der Gleichheit $x = e$ eine Ungleichung der Form $e \geq 0$ erzeugen. Im einzelnen:

1. Wähle eine Gleichung des Systems und daraus eine Unbekannte x mit Koeffizient ungleich null. Löse die Gleichung nach x auf. Es ergibt sich eine Gleichung der Form $x = e$.

2. Da x ein Farkas-Multiplikator ist, gilt $x \geq 0$. Erzeuge also eine Ungleichung $e \geq 0$ und streiche die ursprüngliche Gleichung aus dem System.

3. Ersetze x durch e in allen noch existierenden Gleichungen und in allen bereits erzeugten Ungleichungen.

4. Falls noch Gleichungen vorhanden sind, setze mit 1. fort.

Durch dieses Verfahren wird jede Gleichung durch eine Ungleichung ersetzt, wobei sich bei jeder Ersetzung die Zahl von Unbekannten um eins reduziert.

Anmerkung 4.2.6 Bei der Auswahl von Farkas-Multiplikatoren im ersten Schritt sollten bevorzugt die Farkas-Multiplikatoren λ_e^i der Verzögerung eliminiert werden. Zur späteren Konstruktion der Schedules aus den Prototypen sind lediglich die Werte der Farkas-Multiplikatoren μ_S^i von Interesse. Werden jedoch letztere eliminiert, so müssen diese später wieder aus den Werten der restlichen μ_S^i's und λ_e^i's berechnet werden. In einer Implementierung des Verfahrens erfordert dies also zusätzlichen Aufwand.

Es kann jedoch vorkommen, daß durch den Koeffizientenvergleich Gleichungen entstehen, die keine λ_e^i's enthalten. In diesem Fall ergäbe sich für die Gleichung die Form $e = 0$, wobei e ein Ausdruck in den μ_S^i's ist. Will man auch hier den Mehraufwand einsparen, so formt man aus dieser Gleichung die beiden Ungleichungen

$$e \geq 0 \qquad e \leq 0$$

und fügt diese ebenso in das Ungleichungssystem ein. Nachteilig ist dabei jedoch, daß damit die Anzahl von Ungleichungen steigt.

Anmerkung 4.2.7 Im Hinblick auf das später verwendete Programm zur Lösung von Ungleichungssystemen und dessen Schwierigkeiten mit großen Zahlen ist es hilfreich, bei der Wahl eines zu eliminierenden Multiplikators zunächst nach Multiplikatoren zu suchen, die nach dem Auflösen möglichst kleine Koeffizienten ergeben. Eine weitere Möglichkeit zur Verringerung der Komplexität des erzeugten Ungleichungssystems besteht darin, redundante Ungleichungen und Variablen zu eliminieren. In die Implementierung des Verfahrens wurden Methoden aufgenommen, die beide Optimierungen durchführen.

Das erzeugte Ungleichungssystem besitzt denselben Lösungsraum wie das vorherige Gleichungssystem, abgesehen von den eliminierten Unbekannten. Im folgenden sei es durch

$$G \begin{pmatrix} \vec{\mu} \\ \vec{\lambda} \end{pmatrix} \geq \vec{h} \qquad (6)$$

notiert. G bezeichne dabei die Koeffizientenmatrix, $\vec{\mu}$ den Vektor der Farkas-Multiplikatoren aller Prototypen, $\vec{\lambda}$ den Vektor der Farkas-Multiplikatoren der Verzögerungen Δ_e aller Abhängigkeiten und schließlich $\vec{h}$ den Vektor der Konstanten in jeder Ungleichung.

Beispiel 4.2.8 Mit dem beschriebenen Verfahren zum Erzeugen der Ungleichungen erhält man für die Vereinigung der Gleichungssysteme in Beispiel 4.2.5 etwa (in Abhängigkeit der Wahl der zu eliminierenden Farkas-Multiplikatoren) folgendes Ungleichungssystem:

$$\begin{aligned}
\mu_1^1 - \mu_1^2 - \mu_1^3 + \mu_1^4 - 1 &\geq 0 \\
2\mu_1^1 - 2\mu_1^2 - 4\mu_1^3 + 4\mu_1^4 - 1 &\geq 0 \\
\mu_1^1 - \mu_1^2 - \mu_2^1 + \mu_2^2 + \lambda_3^1 &\geq 0 \\
-\mu_1^0 - 5\mu_1^1 - 9\mu_1^4 + \mu_2^0 + 5\mu_2^1 - 5\lambda_3^1 - 1 &\geq 0
\end{aligned}$$

Um die Matrix G und den Konstantenvektor $\vec{h}$ zu erhalten, stellen wir das hergeleitete Unlgeichungssystem in der folgenden Form dar:

$$\underbrace{\begin{pmatrix} 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & -2 & -4 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & -1 & 1 & 1 \\ -1 & -5 & 0 & 0 & -9 & 1 & 5 & 0 & -5 \end{pmatrix}}_{G} \begin{pmatrix} \mu_1^0 \\ \mu_1^1 \\ \mu_1^2 \\ \mu_1^3 \\ \mu_1^4 \\ \mu_2^0 \\ \mu_2^1 \\ \mu_2^2 \\ \lambda_3^1 \end{pmatrix} \geq \underbrace{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}}_{\vec{h}}$$

Anmerkung 4.2.9 Ist das Ungleichungssystem (6) bei Einhaltung der Nicht-Negativitäts-Bedingung für die Farkas-Multiplikatoren nicht lösbar, so existiert kein affiner eindimensionaler Schedule für das gegebene Eingabe-Programm. Das Programm kann also nicht in linearer Zeit parallel ausgeführt werden. In diesem Fall weicht man auf mehrdimensionale Schedules aus. Eine Methode zu deren Berechnung ist in [7] beschrieben.

Im folgenden soll der Begriff der Minimalität festgelegt werden. Wir führen dazu eine Ordnung auf Schedules ein. Die naheliegende punktweise Ordnung wird wie folgt festgelegt:

Definition 4.2.10 (Ordnung auf $\mathcal{G}$)

$$(\forall\, \Theta_1, \Theta_2 \,:\, \Theta_1, \Theta_2 \in \mathcal{G} \,:\, \Theta_1 \prec_{\mathcal{G}} \Theta_2 \Leftrightarrow (\forall\, S \,:\, S \in \Omega_L \,:\\ (\forall\, \vec{\imath} \,:\, \vec{\imath} \in D_S \,:\, \Theta_1(S,\vec{\imath}) \le \Theta_2(S,\vec{\imath}))))$$

Es stellt sich jedoch als Problem heraus, daß diese Ordnung partiell ist, was zur Folge hat, daß in ihr i. a. kein Minimum existiert. Einen Ausweg aus dieser Situation erreichen wir mit folgendem Satz, der eine Aussage über das punktweise Minimum zweier Schedules macht:

Satz 4.2.11 (Abgeschlossenheit der Scheduleeigenschaft) Sind Θ_1 und Θ_2 Schedules eines Schleifenprogramms L, so gilt dies auch für die Abbildung Θ_3, definiert durch:

$$(\forall\, S \,:\, S \in \Omega_L \,:\, (\forall\, \vec{\imath} \,:\, \vec{\imath} \in D_S \,:\, \Theta_3(S,\vec{\imath}) = \min(\Theta_1(S,\vec{\imath}), \Theta_2(S,\vec{\imath}))))$$

Beweis:

$$\begin{array}{ll}
& (\forall\, S,T \,:\, S,T \in \Omega_L \,:\, (\forall\, \vec{\imath},\vec{\jmath} \,:\, \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \wedge (S,\vec{\imath})\delta(T,\vec{\jmath}) \,:\\
& \quad \Theta_1(S,\vec{\imath}) + 1 \le \Theta_1(T,\vec{\jmath}) \wedge \Theta_2(S,\vec{\imath}) + 1 \le \Theta_2(T,\vec{\jmath})))\\
\Rightarrow & \quad \{ \text{ Anwendung des min-Operators und Monotonie des min-Operators } \}\\
& (\forall\, S,T \,:\, S,T \in \Omega_L \,:\, (\forall\, \vec{\imath},\vec{\jmath} \,:\, \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \wedge (S,\vec{\imath})\delta(T,\vec{\jmath}) \,:\\
& \quad \min(\Theta_1(S,\vec{\imath}) + 1, \Theta_2(S,\vec{\imath}) + 1) \le \min(\Theta_1(T,\vec{\jmath}), \Theta_2(T,\vec{\jmath}))))\\
\Rightarrow & \quad \{ \min(a+1, b+1) = \min(a,b) + 1 \}\\
& (\forall\, S,T \,:\, S,T \in \Omega_L \,:\, (\forall\, \vec{\imath},\vec{\jmath} \,:\, \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \wedge (S,\vec{\imath})\delta(T,\vec{\jmath}) \,:\\
& \quad \min(\Theta_1(S,\vec{\imath}), \Theta_2(S,\vec{\imath})) + 1 \le \min(\Theta_1(T,\vec{\jmath}), \Theta_2(T,\vec{\jmath}))))
\end{array}$$

Dieser Satz stellt die mathematische Grundlage des Verfahrens dar. Es ist also möglich, den minimalen Schedule aus affinen Schedules in $\mathcal{G}$ zu konstruieren, wobei der dadurch entstehende Schedule nicht notwendig aus $\mathcal{G}$ ist, sondern eine stückweise Kombination derartiger Schedules ist. Die Auswahl der Kombination wird durch den Indexraum bestimmt, da durch den min-Operator jeweils der in bestimmten Bereichen des Indexraums zeitoptimale Schedule gewählt wird. Als Folgerung dieses Satzes ergibt sich der laufzeitoptimale Schedule Θ durch

$$(\forall\, S\ :\ S \in \Omega_L\ :\ (\forall\, \vec{\imath}\ :\ \vec{\imath} \in D_S\ :\ \Theta(S,\vec{\imath}) = (\min\ t\ :\ t \in \mathcal{G}\ :\ t(S,\vec{\imath}))))$$

Damit sind wir in der Lage das Optimierungsproblem für diese Methode zu formulieren:

Optimierungsproblem der Methode von Feautrier

$$\begin{aligned}\Theta(S,\vec{\imath}) &= (\min \begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} : G\begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} \ge \vec{h} \wedge \vec{\mu} \ge \vec{0} \wedge \vec{\lambda} \ge \vec{0} :\\ &\qquad \mu_S^0 + (\textstyle\sum\ k\ :\ 1 \le k \le m_S\ :\ \mu_S^k(\vec{a}_S^k \begin{pmatrix}\vec{\imath}\\ \vec{n}\end{pmatrix} + b_S^k)))\end{aligned} \tag{7}$$

(Zur Erinnerung: m_S bezeichnet die Anzahl von Ungleichungen in der Beschreibung des Indexraums der Zuweisung S.) Die Lösung dieses Problems für eine feste Zuweisung liefert somit den entsprechenden zeitoptimalen Schedule der Zuweisung.

Beispiel 4.2.12 Für das Beispiel 4.2.5 ergeben sich folgende Optimierungsprobleme. Der Schedule der ersten Zuweisung ist:

$$\begin{aligned}\Theta(S_1,(i,j)) &= (\min \begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} : G\begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} \ge \vec{h} \wedge \vec{\mu} \ge \vec{0} \wedge \vec{\lambda} \ge \vec{0} :\\ &\qquad \mu_1^0 + \mu_1^1 i + \mu_1^2(5-i) + \mu_1^3(j-i) + \mu_1^4(i-j+9))\end{aligned}$$

Für die zweite Zuweisung ergibt sich:

$$\begin{aligned}\Theta(S_2,i) &= (\min \begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} : G\begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} \ge \vec{h} \wedge \vec{\mu} \ge \vec{0} \wedge \vec{\lambda} \ge \vec{0} :\\ &\qquad \mu_2^0 + \mu_2^1 i + \mu_2^2(5-i))\end{aligned}$$

wobei $G\begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} \geq \vec{h}$ folgendes Ungleichungssystem ist:

$$\begin{pmatrix} 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & -2 & -4 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & -1 & 1 & 1 \\ -1 & -5 & 0 & 0 & -9 & 1 & 5 & 0 & -5 \end{pmatrix} \begin{pmatrix} \mu_1^0 \\ \mu_1^1 \\ \mu_1^2 \\ \mu_1^3 \\ \mu_1^4 \\ \mu_2^0 \\ \mu_2^1 \\ \mu_2^2 \\ \lambda_3^1 \end{pmatrix} \geq \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Eine implementierbare Lösung dieser Optimierungsprobleme mit Hilfe eines Programms zur Berechnung des lexikographischen Minimums wird in Abschnitt 4.2.3 vorgestellt.

Anmerkung 4.2.13 Der berechnete Schedule Θ ist i. a. eine rationale Funktion. Um den Schedule in eine Funktion Θ' mit Bild in $\mathbb{N}$ umzuwandeln, ist es ausreichend, rationale Bildpunkte auf die nächste ganze Zahl aufzurunden.

Satz 4.2.14 (Runden von Schedules) Ist $\Theta : (\Omega_L, D_{S\in\Omega_L}) \to \mathbb{Q}$ ein Schedule für ein Schleifenprogramm L, so gilt dies auch für die Abbildung $\Theta' : (\Omega_L, D_{S\in\Omega_L}) \to \mathbb{N}$, die durch Aufrunden rationaler Bildpunkte von Θ konstruiert wird:

$$\Theta'(\vec{\imath}) = \lceil \Theta(\vec{\imath}) \rceil$$

Beweis: Wir zeigen dies durch einen Widerspruchsbeweis. Wir nehmen also an, Θ' besitze nicht die Scheduleeigenschaft. Für zwei abhängige Instanzen $(S, \vec{\imath})$ und $(T, \vec{\jmath})$ seien dazu

$$\begin{aligned} \alpha &= \Theta'(S,\vec{\imath}) - \Theta(S,\vec{\imath}) \\ \beta &= \Theta'(T,\vec{\jmath}) - \Theta(T,\vec{\jmath}) \end{aligned}$$

die rationalen Differenzen der beiden Abbildungen für diese Instanzen.

$$
\begin{array}{ll}
 & (\exists\, S,T \;:\; S,T \in \Omega_L \;:\; (\exists\, \vec{\imath},\vec{\jmath} \;:\; \vec{\imath} \in D_S \wedge \vec{\jmath} \in D_T \;:\; \Theta'(T,\vec{\jmath}) - \Theta'(S,\vec{\imath}) < 1)) \\
\Rightarrow & \{\ \Theta' \text{ ist eine Abbildung mit Bild in } \mathbb{N}\ \} \\
 & (\exists\, S,T \;::\; (\exists\, \vec{\imath},\vec{\jmath} \;::\; \Theta'(T,\vec{\jmath}) - \Theta'(S,\vec{\imath}) \leq 0)) \\
\Leftrightarrow & \{\ \text{Festlegung von } \alpha \text{ und } \beta\ \} \\
 & (\exists\, S,T \;::\; (\exists\, \vec{\imath},\vec{\jmath} \;::\; \Theta(T,\vec{\jmath}) + \beta - \Theta(S,\vec{\imath}) - \alpha \leq 0)) \\
\Leftrightarrow & \{\ \text{Arithmetische Umformung}\ \} \\
 & (\exists\, S,T \;::\; (\exists\, \vec{\imath},\vec{\jmath} \;::\; \Theta(T,\vec{\jmath}) - \Theta(S,\vec{\imath}) \leq \alpha - \beta)) \\
\Rightarrow & \{\ 0 \leq \alpha < 1\ \} \\
 & (\exists\, S,T \;::\; (\exists\, \vec{\imath},\vec{\jmath} \;::\; \Theta(T,\vec{\jmath}) - \Theta(S,\vec{\imath}) < 1 - \beta)) \\
\Rightarrow & \{\ 0 \leq \beta < 1\ \} \\
 & (\exists\, S,T \;::\; (\exists\, \vec{\imath},\vec{\jmath} \;::\; \Theta(T,\vec{\jmath}) - \Theta(S,\vec{\imath}) < 1))
\end{array}
$$

Dies führt also zu einem Widerspruch zur Scheduleeigenschaft von Θ.

4.2.2 Bewertung der Methode

Die berechneten Schedules sind jeweils zeitoptimal in der Menge der stückweise affinen Abbildungen, die aus Schedules in $\mathcal{G}$ zusammengesetzt sind. Zur Bewertung dieser Schedules ist ein Vergleich mit dem sogenannten freien Schedule notwendig. Der *freie Schedule* ist der Schedule, der jeder Instanz unter Berücksichtigung der Abhängigkeiten den frühest möglichen Zeitpunkt der Ausführung zuordnet.

Wir stellen zunächst fest, daß der von diesem Verfahren berechnete Schedule nicht immer dem freien Schedule entspricht. Wir benötigen dazu einige Begriffe der linearen Algebra [14]:

Definition 4.2.15 (konvexe Menge) Sei V ein $\mathbb{R}$-Vektorraum. Für die Vektoren $\vec{a},\vec{b} \in V$ heißt die Menge

$$S_{\vec{a},\vec{b}} := \{\, \lambda\vec{a} + (1-\lambda)\vec{b} \;:\; \lambda \in [0,1] \,\}$$

Strecke von $\vec{a}$ bis $\vec{b}$. Eine Menge $A \subseteq V$ heißt *konvex*, wenn gilt

$$(\forall\, \vec{a},\vec{b} \;:\; \vec{a},\vec{b} \in A \;:\; S_{\vec{a},\vec{b}} \subseteq A)$$

Da ein Indexraum eine konvexe Menge ist, bildet ein Schedule also Elemente einer konvexen Menge nach $\mathbb{Q}$ ab. Folgende Definition ermöglicht uns die Charakterisierung von Schedules, die mit der Methode von Feautrier berechnet wurden.

Definition 4.2.16 (konvexe und konkave Abbildung) Sei A eine konvexe Teilmenge eines $\mathbb{R}$-Vektorraums. Eine Abbildung $f : A \to \mathbb{R}$ heißt *konvex*, wenn gilt:

$$(\forall\, \vec{a}, \vec{b} \,:\, \vec{a}, \vec{b} \in A \,:\, (\forall\, \lambda \,:\, \lambda \in [0,1] \,:\, f(\lambda\vec{a} + (1-\lambda)\vec{b}) \leq \lambda f(\vec{a}) + (1-\lambda) f(\vec{b})))$$

f ist *konkav*, wenn $-f$ konvex ist.

Geometrisch betrachtet besitzt eine konkave Abbildung die Eigenschaft, in keiner Komponente eine Linkskrümmung aufzuweisen, also nie in der Steigung zuzunehmen.

Schedules, die mit der Methode von Feautrier berechnet wurden, besitzen die Eigenschaft, konkave Funktionen zu sein. Das liegt daran, daß die Menge $\mathcal{G}$ affine Abbildungen enthält, die trivialerweise konkav und konvex sind. Das Bilden des punktweisen Minimums affiner Abbildungen ergibt eine konkave Funktion, wie der nächste Satz zeigt.

Satz 4.2.17 Sind f und g affine Abbildungen von einem $\mathbb{R}$-Vektorraum V nach $\mathbb{R}$ und ist $\vec{x} \in V$, so ist die Abbildung

$$h(\vec{x}) = \min(f(\vec{x}), g(\vec{x}))$$

konkav.

Beweis: Für den Beweis benötigen wir folgende Eigenschaft des Minimums, die sich durch eine Fallunterscheidung leicht herleiten läßt :

$$\min(a+b, c+d) \geq \min(a,c) + \min(b,d) \tag{8}$$

Seien also f und g affine Abbildungen:

$$\begin{array}{ll}
 & (\forall\, \vec{a}, \vec{b} \,:\, \vec{a}, \vec{b} \in V \,:\, (\forall\, \lambda \,:\, \lambda \in [0,1] \,:\, h(\lambda\vec{a} + (1-\lambda)\vec{b}) = \\
 & \quad \min(f(\lambda\vec{a} + (1-\lambda)\vec{b}), g(\lambda\vec{a} + (1-\lambda)\vec{b})))) \\
\Leftrightarrow & \quad \{\ f \text{ und } g \text{ sind affine Abbildungen }\} \\
 & (\forall\, \vec{a}, \vec{b} \,:\, \vec{a}, \vec{b} \in V \,:\, (\forall\, \lambda \,:\, \lambda \in [0,1] \,:\, h(\lambda\vec{a} + (1-\lambda)\vec{b}) = \\
 & \quad \min(\lambda f(\vec{a}) + (1-\lambda) f(\vec{b}), \lambda g(\vec{a}) + (1-\lambda) g(\vec{b})))) \\
\Rightarrow & \quad \{\ \text{Ungleichung (8)}\ \} \\
 & (\forall\, \vec{a}, \vec{b} \,:\, \vec{a}, \vec{b} \in V \,:\, (\forall\, \lambda \,:\, \lambda \in [0,1] \,:\, h(\lambda\vec{a} + (1-\lambda)\vec{b}) \geq \\
 & \quad \min(\lambda f(\vec{a}), \lambda g(\vec{a})) + \min((1-\lambda) f(\vec{b}), (1-\lambda) g(\vec{b}))))
\end{array}$$

$$
\begin{array}{ll}
\Leftrightarrow & \{ \text{ Homogenität des min-Operators } \} \\
& (\forall\, \vec{a},\vec{b} \,:\, \vec{a},\vec{b} \in V \,:\, (\forall\, \lambda \,:\, \lambda \in [0,1] \,:\, h(\lambda\vec{a} + (1-\lambda)\vec{b}) \geq \\
& \quad \lambda \min(f(\vec{a}), g(\vec{a})) + (1-\lambda)\min(f(\vec{b}), g(\vec{b})))) \\
\Leftrightarrow & \{ \text{ Definition von } h \text{ } \} \\
& (\forall\, \vec{a},\vec{b} \,:\, \vec{a},\vec{b} \in V \,:\, (\forall\, \lambda \,:\, \lambda \in [0,1] \,:\, h(\lambda\vec{a} + (1-\lambda)\vec{b}) \geq \\
& \quad \lambda h(\vec{a}) + (1-\lambda)h(\vec{b}))) \\
\Leftrightarrow & \{ \text{ Multiplikation mit } -1 \text{ } \} \\
& (\forall\, \vec{a},\vec{b} \,:\, \vec{a},\vec{b} \in V \,:\, (\forall\, \lambda \,:\, \lambda \in [0,1] \,:\, (-h)(\lambda\vec{a} + (1-\lambda)\vec{b}) \leq \\
& \quad \lambda(-h)(\vec{a}) + (1-\lambda)(-h)(\vec{b}))) \\
\Leftrightarrow & \{ \text{ Definition 4.2.16 } \} \\
& h \text{ ist konkav}
\end{array}
$$

Somit wird von diesem Verfahren der sogenannte *beste konkave Schedule* berechnet. Es gibt jedoch Programme, deren freier Schedule nicht konkav ist:

Beispiel 4.2.18 Gegeben sei folgendes Schleifenprogramm:

```
for i := 0 to 2n do
    a[i] := a[2n - i];
end
```

Der freie Schedule dafür ist:

$$t(i) = \textbf{if } i > n \textbf{ then } 1 \textbf{ else } 0$$

Diese Abbildung ist nicht konkav. Somit kann ein Schedule nach der Methode von Feautrier für dieses Programm nicht zeitoptimal sein. Der von dieser Methode gelieferte Schedule ist

$$\Theta(i) = \lceil \frac{i}{2} \rceil$$

Für den Fall, daß im Eingabe-Programm nur uniforme Abhängigkeiten enthalten sind, läßt sich jedoch zeigen, daß die Differenz zwischen der Laufzeit des freien Schedules und der des besten konkaven Schedules eine von den

Strukturparametern meistens unabhängige Konstante ist. Nach einer Heuristik gilt diese Aussage[3] für den besten affinen Schedule und den freien Schedule [3]. Da die Laufzeit des besten konkaven Schedule aufgrund seiner Konstruktion kleiner oder gleich der des besten affinen Schedules ist, folgt dies also auch für die Schedules dieses Verfahrens. Auf nicht uniforme Abhängigkeiten läßt sich diese Aussage allerdings nicht erweitern, wie Beispiel 4.2.18 zeigt, da der beste konkave Schedule hier die Laufzeit $O(n)$ besitzt.

Eine weiteres Problem stellen Programme dar, für die kein affiner konkaver Schedule existiert. Für derartige Programme ist die Berechnung eines multidimensionalen Schedules erforderlich [7].

Abgesehen von den genannten Einschränkungen gibt die Methode von Feautrier jedoch eine schnelle und effiziente Möglichkeit zur Berechnung von Schedules.

Aus theoretischer Sicht ist damit die Berechnung eines Schedules für ein Eingabe-Programm durch die Lösung des genannten Optimierungsproblems möglich.

In der Praxis stellt sich jedoch das Problem, daß derzeit kein Verfahren in der Lage ist, dieses Optimierungsproblem zu lösen. Die Schwierigkeit liegt darin, daß in der Zielfunktion Parameter auftreten, nämlich die Vektoren $\vec{\imath}$ und $\vec{n}$. Somit scheidet beispielsweise der klassische Simplex-Algorithmus [2] aus.

PIP (Parametric Integer Programming) ist ein Programm, das unter bestimmten Voraussetzungen auch mit Parametern arbeiten kann [10]. Eine Einschränkung betrifft den Begriff der Affinität. Da ein Strukturparameter k im mathematischen Sinn eine Konstante darstellt, ist ein Ausdruck der Form $k\,i$ mit i als Variable nach üblicher Definition affin. Dies gilt allerdings nicht bei PIP. Hier dürfen Strukturparameter lediglich additiv mit Variablen verknüpft werden. Im obigen Optimierungsproblem (7) sind die Farkas-Multiplikatoren die Variablen, $\vec{\imath}$ und $\vec{n}$ die Parameter der Zielfunktion. Diese sind jedoch multiplikativ verknüpft. Eine zweite Schwierigkeit besteht darin, daß PIP nicht mit Zielfunktionen arbeitet, sondern lediglich in der Lage ist,

[3] In [11] wird von einem Test mit 25 Algorithmen mit uniformen Abhängigkeiten berichtet. Dabei ergab sich für 23 dieser Algorithmen eine identische Laufzeit für freien Schedule und affinen Schedule. Bei zwei Algorithmen betrug die Laufzeit für einen affinen Schedule eine Zeiteinheit mehr als für den freien Schedule. Diese Größen blieben auch bei einer Änderung der Strukturparameter gleich.

für ein gegebenes parametrisiertes Ungleichungssystem das lexikografische Minimum aller Lösungen zu berechnen. Somit besitzt Optimierungsproblem (7) keine der beiden erforderlichen Eigenschaften.

Eine Notlösung für dieses Problem ist, die angegebene Zielfunktion zu verwerfen, und nach dem lexikografischen Minimum der Lösungen von

$$G\begin{pmatrix}\vec{\mu}\\ \vec{\lambda}\end{pmatrix} \geq \vec{h}$$

zu suchen:

Beispiel 4.2.19 Für das lexikografische Minimum des Gleichungssystems in Beispiel 4.2.12 ergibt sich für den Vektor $\vec{\mu}$:

$$(\mu_1^0, \mu_1^1, \mu_1^2, \mu_1^3, \mu_1^4, \mu_2^0, \mu_2^1, \mu_2^2, \lambda_3^1) = (0, 0, 0, 0, 1, 0, 2, 2, 0)$$

Dies entspricht nach dem Einsetzen der Farkas-Multiplikatoren in die Prototypen folgendem Schedule:

$$\Theta_{S_1}(i, j) = i - j + 9$$
$$\Theta_{S_2}(i) = 10$$

Unnötigerweise werden jedoch bei diesem Schedule alle Instanzen der Zuweisung S_2 erst ausgeführt, nachdem alle Instanzen von Zuweisung S_1 ausgeführt sind. Auch in Zuweisung S_1 könnten einige Instanzen früher ausgeführt werden. Wir werden sehen, daß die Lösung des Optimierungsproblems zu einem anderen Resultat führt.

Problematisch ist bei dieser Notlösung, daß das Ergebnis von der bei der Konstruktion des Gleichungssystems gewählten Reihenfolge der Abhängigkeiten und Zuweisungen bestimmt wird. Außerdem erhält man im Idealfall lediglich den besten affinen und nicht den besten stückweise affinen Schedule.

Eine weitere Möglichkeit besteht darin, für die Parameter $\vec{\imath}$ und $\vec{n}$ des Problems konkrete Werte einzusetzen, um somit ein nicht-parametrisiertes Problem zu erhalten, das mit einem Verfahren wie Simplex gelöst werden kann. Bei Zuweisungen mit großem Indexraum ist diese Lösung aber nicht tragbar. Außerdem liefert sie lediglich die Werte des Schedules für einen isolierten Indexpunkt, jedoch keinen geschlossenen Ausdruck in Form einer mathematischen Abbildung.

Wir werden aber im nächsten Abschnitt sehen, daß durch eine Umwandlung des Optimierungsproblems diese Schwierigkeiten beseitigt werden können.

4.2.3 Anpassung an PIP

Die Grundlage für eine Umwandlung des Optimierungsproblems dieser Methode liefert folgendes aus der linearen Programmierung bekannte Theorem [14]:

Satz 4.2.20 (Dualitätstheorem) Für ein Minimierungsproblem mit nichtleerer Lösungsmenge gilt:

$$(\min\ \vec{\mu}\ :\ \vec{\mu}\geq\vec{0}\ \wedge\ G\vec{\mu}\geq\vec{h}\ :\ \vec{k}\vec{\mu}) = (\max\ \vec{\nu}\ :\ \vec{\nu}\geq\vec{0}\ \wedge\ \vec{\nu}G\leq\vec{k}\ :\ \vec{\nu}\vec{h})$$

Anmerkung 4.2.21 Die Voraussetzung zur Anwendung des Dualitätstheorem ist die Lösbarkeit des Ungleichungssystems. In einer Implementierung dieses Verfahrens muß also vor der Umformung des Problems das Ungleichungssystem $G\vec{\mu} \geq \vec{h}$ auf seine Lösbarkeit getestet werden. Da in diesem Ungleichungssystem keine Parameter auftreten, muß dazu nicht unbedingt PIP benutzt werden. Diese Aufgabe kann auch von einem Algorithmus (wie Simplex mit beliebiger Zielfunktion) gelöst werden. Wir gehen im folgenden davon aus, daß das Ungleichungssystem $G\vec{\mu}\geq\vec{h}$ lösbar ist, und daß folglich ein Minimum dieser Lösungen existiert. Als Konsequenz des Satzes gilt dann auch die Existenz des Maximums.

Die Anwendung des Dualitätstheorems ergibt damit folgendes umgeformte Optimierungsproblem:

$$\begin{aligned}\Theta(S,\vec{\imath}) = (&\max\ \vec{\nu}\ :\ \vec{\nu}\geq\vec{0}\wedge \\ &\vec{\nu}G\leq(\overbrace{0,\ldots,0}^{N},1,\vec{a}_S^1\begin{pmatrix}\vec{\imath}\\ \vec{n}\end{pmatrix}+b_S^1,\ldots,\vec{a}_S^{ms}\begin{pmatrix}\vec{\imath}\\ \vec{n}\end{pmatrix}+b_S^{ms},\overbrace{0,\ldots,0}^{N'})\ :\ \vec{\nu}\vec{h})\end{aligned}$$

wobei für N und N' gilt:

$$\begin{aligned}N &= (\textstyle\sum\ R\ :\ R\in\Omega_L\wedge R<_L S\ :\ m_R+1),\\ N' &= (\textstyle\sum\ T\ :\ T\in\Omega_L\wedge S<_L T\ :\ m_T+1)+|\vec{\lambda}|\end{aligned}$$

In der Zielfunktion treten nun keine Parameter mehr auf, da die Einträge des Vektors $\vec{h}$ stets Konstanten sind. Lediglich in den Bedingungen für $\vec{\nu}$ existieren die Parameter $\vec{\imath}$ und $\vec{n}$, doch hier nur in additiver Form. Damit ist die Problemstellung im Sinne von PIP affin.

Anmerkung 4.2.22 Bei der Überführung des aus Gleichung (5) erzeugten Gleichungssystems in das Unlgeichungssystem (6) werden Farkas-Multiplikatoren eliminiert. Die obige Formulierung des Optimierungsproblems gibt aber für alle Farkas-Multiplikatoren Koeffizienten an, weil die Wahl der zu eliminierenden Multiplikatoren auf die Anzahl der insgesamt eliminierten Multiplikatoren Einfluß nimmt. Diese Koeffizienten sind in den entsprechenden Vektoren und in G mit Null besetzt.

Wie zuvor erwähnt, berechnet PIP für ein parametrisiertes Ungleichungssystem das lexikografische Minimum aller Lösungen. Somit stehen wir vor der Aufgabe, das durch die Anwendung des Dualitätstheorems erhaltene Maximierungsproblem mit Hilfe des lexikografischen Minimums zu lösen.

Zunächst soll gezeigt werden, wie die Zielfunktion Θ in PIP dargestellt werden kann. Hier benutzen wir eine Eigenschaft des lexikografischen Maximums. Wir führen dazu eine neue Variable m in das Ungleichungssystem ein, die durch die beiden Ungleichungen

$$\begin{aligned} m &\geq \vec{\nu}\vec{h} \\ m &\leq \vec{\nu}\vec{h} \end{aligned}$$

mit der Zielfunktion identisch ist. Ordnet man diese Variable, die in allen anderen Ungleichungen den Koeffizienten 0 hat, als erste Variable in das System ein, so wird bei der Berechnung des lexikografischen Maximums gerade das Maximum von m und damit von Θ berechnet.

Es bleibt zu klären, wie mit PIP das lexikografische Maximum berechnet wird. Dabei hilft uns folgende Eigenschaft der lexikografischen Ordnung. Ist die Existenz eines lexikografischen Maximums gegeben, so läßt es sich durch das lexikografische Minimum für ein Polyeder $P = \{\vec{\imath} : A\vec{\imath} \geq \vec{b}\}$ aufgrund folgender Beziehungen darstellen:

$$\begin{aligned} (\text{lexmax}\ \vec{\imath} : A\vec{\imath} \geq \vec{b} : \vec{\imath}) &= -(\text{lexmin}\ \vec{\imath} : A\vec{\imath} \geq \vec{b} : -\vec{\imath}) \\ &= -(\text{lexmin}\ \vec{\imath} : A(-\vec{\imath}) \geq \vec{b} : \vec{\imath}) \end{aligned}$$

Geometrisch betrachtet führt man dadurch eine Spiegelung aller Punkte des Polyeders an den Koordinatenachsen durch. Nach Konstruktion ist der Schedule und damit die neu eingeführte Unbekannte m stets positiv. Gleiches gilt nach Anwendung des Dualitätstheorems für den Vektor $\vec{\nu}$ der restlichen

Variablen. Durch die Spiegelung erhält man somit einen Lösungsvektor, dessen Komponenten ausschließlich kleiner oder gleich null sind. PIP ist aber auf Lösungsvektoren beschränkt, die in allen Komponenten positiv sind. Für ein Polyeder wie z. B. $\{(i,j)\ :\ 0 \leq i \leq n \wedge 0 \leq j \leq n\}$ mit lexikographischen Maximum (n,n) würde PIP bei Anwendung der genannten Umformung also $(0,0)$ als Ergebnis liefern. Abhilfe schafft hier die Möglichkeit, daß in PIP einer der Parameter mit der Eigenschaft ausgestattet werden kann, beliebige positive Werte annehmen zu können. Wir führen dazu einen neuen Parameter p ein und vollziehen zusätzlich zur Spiegelung eine Translation um p:

$$\begin{aligned} & -(\text{lexmin}\ \vec{\imath}\ :\ A(-\vec{\imath}) \geq \vec{b}\ :\ \vec{\imath}) \\ = \quad & (\lim\ p\ :\ p \to \infty\ :\ (\, p\vec{1} - (\text{lexmin}\ \vec{\imath}\ :\ A(p\vec{1} - \vec{\imath}) \geq \vec{b}\ :\ \vec{\imath}))) \end{aligned} \tag{9}$$

Ist p groß genug, so ist der Lösungsvektor des lexikographischen Minimums, der nun von p abhängig ist, positiv. Da man sich durch die Translation von der Einschränkung auf positive Lösungsvektoren löst und man aber wieder nur an positiven Lösungen interessiert ist, benötigen wir noch folgende Ungleichungen:

$$\begin{gathered} p - m \geq 0 \\ p\vec{1} - \vec{\imath} \geq \vec{0} \end{gathered}$$

Wie man durch eine Rücktransformation und Spiegelung sieht, garantieren diese Ungleichungen die Positivität aller Variablen.

Somit läßt sich mit PIP also auch das lexikographische Maximum für ein Ungleichungssystem berechnen, wobei auf das Resultat noch eine Rücktransformation, wie in (9) angegeben, angewendet werden muß.

Als Ergebnis der Anpassung an PIP ergibt sich insgesamt folgendes Ungleichungssystem als Eingabe:

$$\begin{gathered} (p\vec{1} - \vec{\nu})G \leq (\overbrace{0,\ldots,0}^{N}, 1, \vec{a}_S^{1} \begin{pmatrix} \vec{\imath} \\ \vec{n} \end{pmatrix} + b_S^1, \ldots, \vec{a}_S^{m_S} \begin{pmatrix} \vec{\imath} \\ \vec{n} \end{pmatrix} + b_S^{m_S}, \overbrace{0,\ldots,0}^{N'}) \\ p - m \geq (p\vec{1} - \vec{\nu})\vec{h} \\ p - m \leq (p\vec{1} - \vec{\nu})\vec{h} \\ p - m \geq 0 \\ p\vec{1} - \vec{\imath} \geq \vec{0} \end{gathered} \tag{10}$$

mit der Variablenreihenfolge $m, \nu_1, \ldots, \nu_k$, wobei k die Anzahl der Zeilen von G ist. Als Lösung ergibt sich somit $p - m$.

Beispiel 4.2.23 Die Transformation des Optimierungsproblems aus Beispiel 4.2.12 in eine für PIP zugängliche Form liefert mit (10) folgendes System von Ungleichungen als Eingabe für PIP:

Für Zuweisung S_1:

$$\begin{array}{rcl@{\qquad}rcl}
-\nu_1^4 + 1 + p & \geq & 0 & \nu_1^3 - 5\nu_1^4 + 4p & \geq & 0 \\
\nu_1^1 + 2\nu_1^2 + \nu_1^3 - 5\nu_1^4 + i + p & \geq & 0 & -m_1 + \nu_1^1 + \nu_1^2 + \nu_1^4 - 2p & \geq & 0 \\
-\nu_1^1 - 2\nu_1^2 - \nu_1^3 - i + 5 + 4p & \geq & 0 & m_1 - \nu_1^1 - \nu_1^2 - \nu_1^4 + 2p & \geq & 0 \\
-\nu_1^1 - 4\nu_1^2 - i + j + 5p & \geq & 0 & p - m_1 & \geq & 0 \\
\nu_1^1 + 4\nu_1^2 - 9\nu_1^4 + i - j + 9 + 4p & \geq & 0 & p - \nu_1^1 & \geq & 0 \\
\nu_1^4 - p & \geq & 0 & p - \nu_1^2 & \geq & 0 \\
-\nu_1^3 + 5\nu_1^4 - 4p & \geq & 0 & p - \nu_1^3 & \geq & 0 \\
\nu_1^3 - p & \geq & 0 & p - \nu_1^4 & \geq & 0
\end{array}$$

Für Zuweisung S_2:

$$\begin{array}{rcl@{\qquad}rcl}
-\nu_2^4 + 1 + p & \geq & 0 & \nu_2^3 - 5\nu_2^4 + 4p & \geq & 0 \\
\nu_2^1 + 2\nu_2^2 + \nu_2^3 - 5\nu_2^4 + p & \geq & 0 & -m_2 + \nu_2^1 + \nu_2^2 + \nu_2^4 - 2p & \geq & 0 \\
-\nu_2^1 - 2\nu_2^2 - \nu_2^3 + 4p & \geq & 0 & m_2 - \nu_2^1 - \nu_2^2 - \nu_2^4 + 2p & \geq & 0 \\
-\nu_2^1 - 4\nu_2^2 + 5p & \geq & 0 & p - m_2 & \geq & 0 \\
\nu_2^1 + 4\nu_2^2 - 9\nu_2^4 + 4p & \geq & 0 & p - \nu_2^1 & \geq & 0 \\
\nu_2^4 + 1 - p & \geq & 0 & p - \nu_2^2 & \geq & 0 \\
-\nu_2^3 + 5\nu_2^4 + i - 4p & \geq & 0 & p - \nu_2^3 & \geq & 0 \\
\nu_2^3 + 5 - i - p & \geq & 0 & p - \nu_2^4 & \geq & 0
\end{array}$$

Als Lösung für $p - m_1$ und $p - m_2$ und damit für Θ_{S_1} und Θ_{S_2} ergibt sich schließlich:

$$\begin{array}{rcl}
\Theta_{S_1}(i,j) & = & \textbf{if } j \leq 9 \textbf{ then } i \textbf{ else } i - j + 9 \\
\Theta_{S_2}(i) & = & i + 1
\end{array}$$

In Abbildung 7 ist der Verlauf beider Abbildungen im Indexraum dargestellt. Der berechnete Schedule ist hier sogar der freie Schedule.

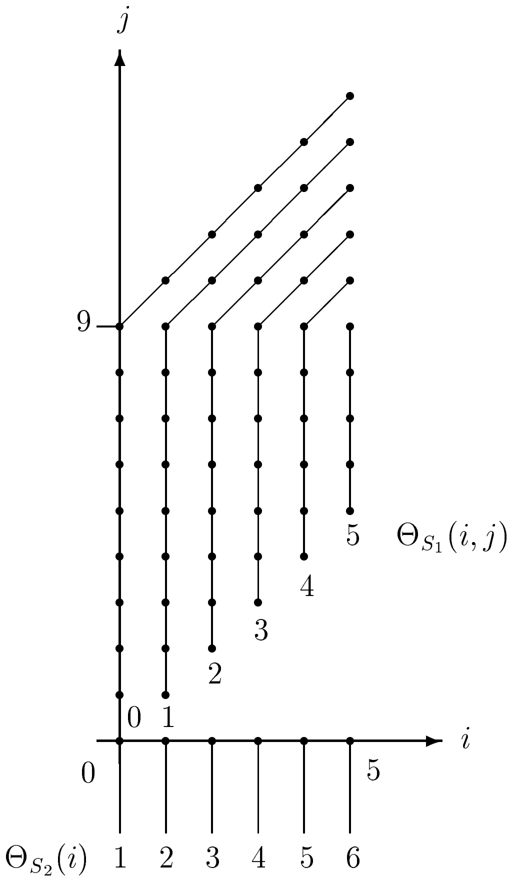

Abbildung 7: Der Schedule aus Beispiel 4.2.23.

Anmerkung 4.2.24 PIP unterstützt die Angabe von Nebenbedingungen für die Beschreibung eines Ungleichungssystems. Mit Nebenbedingungen werden durch Ungleichungen Beziehungen zwischen den Parametern des Systems angegeben. Für unsere Berechnungen stellen u. a. die Schleifenindizes des Eingabe-Programms Parameter dar. Also werden durch die Ungleichungen des Indexraums einer Zuweisung Größenverhältnisse zwischen den Parametern des Optimierungsproblems für den Schedule der Zuweisung ausgedrückt. Um die von PIP berechnete Lösung möglichst einfach zu halten, also redundante Bedingungen in der Lösung zu vermeiden, ist es deshalb ratsam, die Ungleichungen des Indexraums jeweils als Nebenbedingungen mitanzugeben.

5 Berechnung von Allokationen

In diesem Abschnitt soll eine Methode zur Berechnung von Allokationen für Eingabe-Programme vorgestellt werden [8]. Im ersten Teil werden wir ein Verfahren kennenlernen, das in der Lage ist, für jedes solche Programm eine eindimensionale Allokation zu konstruieren, im zweiten Teil wird dieses Verfahren auf mehrdimensionale Allokationen erweitert.

5.1 Eindimensionale Allokation

Wie im ersten Abschnitt erwähnt, stellt eine Allokation eine Abbildung dar, die jeder Instanz einer Zuweisung einen virtuellen Prozessor zur Abarbeitung zuweist. Die Berechnung einer Allokation ist damit einerseits abhängig vom verwendeten Prozessorlayout, andererseits vom Speichermodell des Parallelrechners. Wir machen deshalb für das vorzustellende Verfahren folgende Annahmen:

- Jedem Prozessor steht ein eigener Speicher zur Verfügung; es gibt keinen gemeinsamen Speicher. Es gilt die *owner computes rule*, die besagt, daß die Ergebnisse einer Berechnung stets im Speicher des Prozessors abgelegt werden, der die Berechnung durchgeführt hat. Werden zur Berechnung Ergebnisse benötigt, die nicht im Speicher des ausführenden Prozessors liegen, so können diese über ein Kommunikationsnetz angefordert werden.

- Das Kommunikationsnetz stellt direkte Verbindungen zwischen allen Prozessoren her. Damit soll sichergestellt werden, daß die Kosten für die Kommunikation unabhängig von der Entfernung der beteiligten Prozessoren sind. Fordert ein Prozessor einen Variablenwert an, so entstehen keine Kosten, falls dieser Wert in seinem eigenen Speicher liegt. Ist dies nicht der Fall, fallen für die entsprechende Kommunikation sehr hohe, aber konstante Kosten an.

- Für die Eingabe-Programme dieser Methode gelten die im ersten Abschnitt genannten Einschränkungen.

Während bei der Berechnung eines Schedules meistens ein laufzeit-optimales Programm angestrebt wird, steht bei der Berechnung einer Allokation die Reduktion der Anzahl benötigter Prozessoren oder der Kommunikation zwischen Prozessoren im Vordergrund. Das im folgenden beschriebene Verfahren hat letzteres Ziel.

Nach der owner computes rule wird eine Kommunikation immer dann erforderlich, wenn die auszuführende Instanz von einer anderen Instanz abhängig ist, die nicht auf demselben Prozessor ausgeführt wurde, und deren Resultat damit nicht in dessen Speicher vorhanden ist. Die Grundlage für die vorzustellende Methode besteht darin, die Allokation so zu konstruieren, daß abhängige Zuweisungen auf demselben Prozessor ausgeführt werden. Formal bedeutet das für eine Allokation Π und jede Abhängigkeit e:

$$(\forall\ \vec{\jmath} : \vec{\jmath} \in P_e : \Pi(\delta(e), \vec{\jmath}) = \Pi(\sigma(e), h_e(\vec{\jmath}))) \tag{11}$$

Die einfachste Lösung dieses Gleichungssystems ist die triviale, die alle Instanzen von Zuweisungen auf denselben Prozessor abbildet, aber damit keinerlei Parallelität ausnutzt. Es stellt sich jedoch heraus, daß es Programme gibt, die nur diese Lösung zulassen, wie z. B. folgendes Schleifenprogramm:

for $i := 1$ **to** 10 **do**
  for $j := i - 9$ **to** i
    $a[i,j] = a[i-1,j] + a(i-2,j+1)$
  end
end

Hier müsste die Allokation folgende Gleichung erfüllen:

$$\Pi(i,j) = \Pi(i-1,j) = \Pi(i-2,j+1)$$

Eine Lösung ist offensichtlich nur für ein konstantes Π möglich. Deshalb versucht man eine Allokation zu finden, welche die Anzahl von Instanzen maximiert, für die Gleichung (11) erfüllt ist. Diese Aufgabe stellt sich jedoch als sehr schwierig heraus, da es allgemein aufwendig ist, die Größe eines Polyeders, also die Anzahl der enthaltenen Punkte, zu berechnen. Aus diesem Grund vereinfacht man die Problemstellung dadurch, daß man nach Allokationen sucht, welche die Anzahl von Abhängigkeiten maximieren, deren zugehörige Indexpunkte Gleichung (11) erfüllen und damit keine Kommunikation verursachen. Wir bezeichnen Abhängigkeiten mit dieser Eigenschaft bzgl. einer gegebenen Allokation im weiteren als *Trennkanten*.

Unser Ziel ist es nun, die Anzahl von Trennkanten zu maximieren. Dabei gilt die anfangs genannte Forderung der Nicht-Trivialität der Allokation bezüglich jeder Zuweisung. Jeder Abhängigkeit wird dabei ein Gewicht entsprechend des verursachten Kommunikationsaufwands zugeteilt. Abhängigkeiten mit großem Gewicht sollen bevorzugt eliminiert werden. Die geeignete Wahl einer Gewichtsfunktion wird am Ende dieses Abschnitts vorgestellt.

5.1.1 Vorstellung des Verfahrens

Wie bei der Methode von Feautrier zur Berechnung von Schedules bildet man auch hier zunächst Prototypen für die Allokation jeder Zuweisung, die jeweils affine Abbildungen mit unbekannten Koeffizienten darstellen:

$$\Pi(S, \vec{\imath}) = \vec{a}_S \vec{\imath} + \vec{b}_S \vec{n} + c_S \tag{12}$$

Mit $\vec{n}$ bezeichnen wir wieder den Vektor der Strukturparameter.

Ausgangspunkt des Verfahrens ist die Tatsache, daß sich wegen Gleichung (11) jeder Abhängigkeit eine Distanzfunktion $d_e : P_e \to \mathbb{Z}$ zuordnen läßt:

$$d_e(\vec{\jmath}) = \Pi(\delta(e), \vec{\jmath}) - \Pi(\sigma(e), h_e(\vec{\jmath})) \tag{13}$$

Anmerkung 5.1.1 In [8] werden die Strukturparameter nicht explizit in der Formulierung der Prototypen angegeben. Die Prototypen haben dort folgendes Format:

$$\Pi(S, \vec{\imath}) = \vec{a}_S \vec{\imath} + b_S$$

Da aber eine Berücksichtigung von Strukturparametern erforderlich ist, etwa bei der h-Transformation $h_e(i) = n - i$, werden diese wohl in den konstanten Anteil b_S verlagert. Für eine Implementierung des Verfahrens bedeutet das unnötigen Aufwand zur Speicherung von Konstanten. Deshalb verwenden wir im folgenden die oben eingeführte Definition von Prototypen.

Eine Abhängigkeit ist offensichtlich eliminiert, wenn für alle Indexpunkte $\vec{\jmath} \in P_e$ gilt: $d_e(\vec{\jmath}) = 0$. Ist P_e volldimensional, so kann dies nur der Fall sein, wenn sämtliche Koeffizienten der Distanzfunktion d_e, die als Differenz affiner Abbildungen wieder affin ist, null sind. Dies gilt nicht, wenn P_e nicht volldimensional ist. Da wir in der weiteren Berechnung die Koeffizienten von

d_e gleich Null setzen wollen, stehen wir vor der Aufgabe, entweder P_e in ein volldimensionales Polyeder umzuformen, oder die Distanzfunktion so zu definieren, daß nur die Schleifenindizes abgebildet werden, die eine Dimension in P_e erzeugen. Dazu benötigen wir einige Begriffe aus der linearen Programmierung [14]:

Definition 5.1.2 (implizite Gleichung) Sei $A\vec{x} \leq \vec{b}$ ein System von Ungleichungen. Dann heißt eine Ungleichung $\vec{a}\vec{x} \leq \beta$ dieses Systems implizite Gleichung, wenn gilt:

$$(\forall\ \vec{x}\ :\ A\vec{x}\leq\vec{b}\ :\ \vec{a}\vec{x} = \beta)$$

Mit $A^{=}\vec{x} \leq \vec{b}^{=}$ wird das System der impliziten Gleichungen in $A\vec{x} \leq \vec{b}$ beschrieben, $A^{+}\vec{x}\leq\vec{b}^{+}$ bezeichne das System der übrigen Ungleichungen.

Satz 5.1.3 (Dimension eines Polyeders) Sei $A\vec{x}\leq\vec{b}$ ein System von Ungleichungen in n Variablen, das ein Polyeder P beschreibt. Dann ist die Dimension von P gleich n minus dem Rang der Matrix, die die impliziten Gleichungen beschreibt:

$$dim(P) = n - rg(A^{=})$$

In [8] ist eine Methode angesprochen, die P_e zu einem volldimensionalen Polyeder. Die Idee ist dabei, dieses Polyeder parametrisiert darzustellen, wobei die Parameter gerade die Schleifenindizes sind, die in P_e keine Dimension aufspannen. Wir werden hier die zweite Möglichkeit besprechen, welche die Distanzfunktion umformt.

Besitzt P_e die Dimension k und ist l die Anzahl umgebender Schleifenindizes $i_1, \ldots, i_l$ der Zuweisung $\delta(e)$, so existieren also $l-k$ explizite Gleichungen der Form $\vec{a}\vec{x} - \beta = 0$ in der Beschreibung von P_e, die jeweils einen Schleifenindex evtl. in Abhängigkeit von umgebenden Schleifenindizes festlegen. Löst man die impliziten Gleichungen nach diesen Schleifenindizes auf und ersetzt sie in der Distanzfunktion durch den erhaltenen Ausdruck, so erhält man für die Distanzfunktion eine Abbildung in ausschließlich den k Schleifenindizes, die eine Dimension erzeugen. Mit der neuen Festlegung der Distanzfunktion gilt dann wieder, daß diese Abbildung genau dann überall in P_e null ist, wenn ihre Koeffizienten null sind.

Für jede Abhängigkeit entsteht eine Abbildung d_e, deren Koeffizienten sich aus additiven Kombinationen von Koeffizienten der Prototypen zusammensetzen. Nach den eben genannten eventuellen Umformungen bzgl. d_e, kann d_e nur null sein, wenn die Koeffizienten von d_e null sind.

Führt man nun in d_e einen Koeffizientenvergleich bezüglich aller Variablen, Strukturparameter und des konstanten Anteils durch, so ergibt sich damit folgendes System von Gleichungen:

$$C_e \vec{\alpha}_S = 0$$

wobei $\vec{\alpha}_S$ den Vektor aller unbekannten Koeffizienten $\vec{a}_S$, $\vec{b}_S$ und c_S aus (12) darstellt. Die Vereinigung dieser Systeme, die jeweils durch eine Abhängigkeit erzeugt werden, notieren wir mit

$$C\vec{\alpha} = 0$$

Wir nehmen im folgenden an, daß C n Zeilen und m Spalten besitzt.

Der Lösungsraum dieses Gleichungssystems für den Koeffizientenvektor $\vec{\alpha}$ besteht i. a. nur aus dem Nullvektor. Wie bereits erwähnt, ist aber unser Ziel die Berechnung von nicht-trivialen Allokationen. Eine Allokation wird als nicht-trivial betrachtet, wenn mindestens einer der Koeffizienten, die vor den Schleifenindizes stehen, nicht null ist. Zur Lösung des Gleichungssystems mit der Zusatzbedingung der nicht-trivialen Allokationen benutzen wir folgende Abwandlung der Gauss-Jordan-Elimination zur Berechnung der Koeffizienten [8]. Die zugrundeliegende Idee ist, nach jedem Ersetzungsschritt eines Koeffizienten durch eine additive Kombination anderer Koeffizienten zu testen, ob dadurch einer der Prototypen trivial wird. Ist dies der Fall, wird diese Ersetzung verworfen.

Algorithmus 5.1.4 (Bestimmung der Prototypen-Koeffizienten)

Eingabe: Ein Gleichungssystem $C\vec{\alpha} = 0$ und eine Menge P von Prototypen.

Ausgabe: Substitution für $\vec{\alpha}$.

Methode: Der Algorithmus benutzt folgende Strukturen und Funktionen:

– $\mathcal{S}$ und $\mathcal{S}'$ bezeichnen Substitutionen von Variablen, hier den Koeffizienten $\alpha_1, \ldots, \alpha_m$, durch lineare Ausdrücke. Eine Substitution wird mit $[x \leftarrow \mathit{expr}]$ notiert, wodurch ausgedrückt wird, daß die Variable x durch den Ausdruck expr substituiert wird.

– Das Array *eliminated* vom Typ *boolean* speichert, welche Variablen eliminiert wurden.

– Die Funktion $\mathit{get\_equation}(i, C\vec{\alpha} = 0)$ liefert die i-te Gleichung des Systems.

– Die Funktion $\mathit{subst\_equation}(S, eq)$ wendet eine Substitution S auf die Gleichung eq an und gibt als Resultat die umgeformte Gleichung zurück.

– *zero_equation* bezeichnet die Gleichung $0 = 0$.

– Die Funktion $\mathit{coeff}(j, eq)$ hat den Koeffizienten der j-ten Variable in der Gleichung eq als Resultat.

– Die Funktion $\mathit{solve}(j, eq)$ löst eine Gleichung eq nach der j-ten Variable auf und gibt die rechte Seite der umgeformten Gleichung zurück.

– Die Funktion $\mathit{prototypes\_trivial}(S', P)$ testet, ob sich die Prototypen in P durch die Anwendung einer Substitution S' auf die Koeffizienten zu trivialen Funktionen auswertet, deren Koeffizienten alle null sind. Gilt dies für mindestens einen der Prototypen, so gibt diese Funktion *true* als Resultat zurück.

Initialisierung: $S =$ leere Substitution
$(\forall\, i \,:\, 1 \le i \le m \,:\, \mathit{eliminated}[i] = \mathit{false})$

for $i := 1$ **to** n **do**
  $eq := \mathit{get\_equation}(i, C\vec{\alpha} = 0)$;
  if $\mathit{subst\_equation}(S, eq) \neq \mathit{zero\_equation}$ **then**
    $j := 1$;
    while $\mathit{coeff}(j, eq) = 0$ **or** $\mathit{eliminated}[j]$ **do**
      $j := j + 1$;
    end
    $\mathit{expr} := \mathit{solve}(j, eq)$;

```
        eliminated(j) := true;
        τ := [α_j ← expr];
        S' := τ ∘ S;
        if not prototypes_trivial(S', P) then
            S := S';
        else
            eliminated(j) := false;
        fi
    fi
end
```

Alle nicht eliminierten Koeffizienten von $\vec{\alpha}$ können nun auf Werte ungleich null gesetzt werden, so daß alle eliminierten Koeffizienten von $\vec{\alpha}$ durch die berechnete Substitution ebenfalls ungleich null werden. Um dies zu sehen, stellen wir die Substition in allgemeiner Form dar. Wir führen dazu folgende Bezeichnungen ein:

- m bezeichne wieder die Anzahl der Koeffizienten.
- k ist die Anzahl nicht eliminierter Koeffizienten.
- l ist die Anzahl der Koeffizienten, die durch die Substitution explizit gleich Null gesetzt werden.
- r ist die Anzahl aller übrigen Koeffizienten, deren Werte sich in Abhängigkeit der nicht eliminierten Koeffizienten ergeben.
- Der Vektor $\vec{a}'$ bezeichne das Resultat einer Substitution des Vektors $\vec{a}$.

Nach entsprechender Sortierung hat dann die errechnete Substition die Form:

$$\left[\begin{pmatrix} a'_1 \\ a'_2 \\ \vdots \\ a'_n \end{pmatrix} \leftarrow \begin{pmatrix} & S & \\ & N & \\ 0 & \cdots & 0 & \\ \vdots & \ddots & \vdots & \mathcal{I}_k \\ 0 & \cdots & 0 & \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}\right] \tag{14}$$

Die $l \times m$-Matrix N hat ausschließlich Einträge gleich Null. Sie repräsentiert diejenigen Substitutionen, durch die Koeffizienten gleich Null gesetzt werden. Der untere Teil der Matrix stellt die nicht eliminierten Koeffizienten dar. Dies entspricht der Substitution eines Koeffizienten durch sich selbst. Die $r \times m$-Matrix S gibt die Substitution eliminierter Koeffizienten wieder, die von nicht eliminierten Koeffizienten abhängig sind. Bezeichnet $\vec{s_i}$ den i-ten Zeilenvektor von S, so kann die obige Aussage bzgl. der Existenz geeigneter Werte für nicht eliminierte Koeffizienten nun formal ausgedrückt werden:

$$(\exists\ \vec{a}\ :\ \vec{a} \in \mathbb{Q}^m\ :\ (\forall\ i\ :\ 1 \leq i \leq r\ :\ \vec{s_i}\vec{a} \neq 0))$$

Wir zeigen dies durch die Herleitung eines Widerspruchs zur Negation:

$$(\forall\ \vec{a}\ :\ \vec{a} \in \mathbb{Q}^m\ :\ (\exists\ i\ :\ 1 \leq i \leq r\ :\ \vec{s_i}\vec{a} = 0))$$

Dies führt sofort zum Widerspruch, weil dazu für eine Zeile der Matrix S eine Nullzeile benötigt wird und S nach Konstruktion keine Nullzeilen besitzt.

Anmerkung 5.1.5 Wie bei der Berechnung von Schedules im zweiten Verfahren kann es auch hier vorkommen, daß die berechnete Allokation rational ist. Aus theoretischer Sicht stellt dies kein Problem dar, da hier anders als für einen Schedule kein Mindestabstand für die Prozessornummern gefordert ist. In der Praxis ist es jedoch üblich, Prozessoren ganzzahlig aufzuzählen. Eine Möglichkeit, dies zu erreichen, besteht darin, die Allokationen aller Zuweisungen mit dem gemeinsamen Nenner zu multiplizieren.

Anmerkung 5.1.6 Im vorgestellten Algorithmus wird durch das Akzeptieren einer Substitution erreicht, daß für eine Abhängigkeit die in Gleichung (13) angegebene Distanzfunktion in einer Komponente null gesetzt wird. Wird dagegen eine Substitution verworfen, hat das zur Folge, daß die Abhängigkeit, aus der diese Substitution mit Gleichung (13) erzeugt wurde, keine Trennkante sein kann. Alle anderen zu dieser Abhängigkeit gehörenden Substitutionen werden aber dennoch getestet und eventuell akzeptiert. Damit erreicht man, daß eine Nicht-Trennkante eine Distanzfunktion besitzt, die wenigstens in einigen Komponenten null ist. Diese Optimierung der Distanzfunktion ist aber unnötig, da wir bei den am Anfang des Abschnitts vorgestellten Annahmen von einer direkten Verbindung aller Prozessoren ausgehen. Wir schlagen deshalb folgende naheliegende Änderung des Algorithmus vor: Falls eine der zu einer Abhängigkeit gehörenden Substitutionen verworfen wird, so verwerfe alle bisherigen und folgenden Substitutionen dieser

Abhängigkeit. Wir ermöglichen dadurch anderen Abhängigkeiten, zu Trennkanten zu werden. Diese Modifikation wurde auch für die Implementierung der Methode gewählt.

Beispiel 5.1.7 Gegeben sei folgendes Schleifenprogramm:

```
for i := 0 to 20 do
    for j := 0 to 20 do
        for k := 0 to 20 do
            a[i, j] := b[i, j − 6] + d[i − 1, j + 3];     {S1}
            b[i + 1, j − 1] := c[i + 2, j + 5];           {S2}
            d[i, j − 1] := a[i, j − 1]                    {S3}
        end
    end
end
```

Nach Gleichung (12) ergibt sich für die Prototypen der Allokation jeder Zuweisung:

$$\begin{aligned}
\Pi(S_1,(i,j,k)) &= a_1^1 i + a_1^2 j + a_1^3 k + c_1\\
\Pi(S_2,(i,j,k)) &= a_2^1 i + a_2^2 j + a_2^3 k + c_2\\
\Pi(S_3,(i,j,k)) &= a_3^1 i + a_3^2 j + a_3^3 k + c_3
\end{aligned}$$

Es existieren folgende drei Abhängigkeiten:

$$\begin{array}{ll}
e_1 = S_1\delta S_3: & h_{e_2}(i,j,k) = (i, j-1, k)\\
& P_{e_2} = \{(i,j,k) \,:\, 0 \le i,k \le 20 \wedge 1 \le j \le 20\}\\
e_2 = S_3\delta S_1: & h_{e_3}(i,j,k) = (i-1, j+4, k)\\
& P_{e_3} = \{(i,j,k) \,:\, 1 \le i \le 20 \wedge 0 \le j \le 16 \wedge 1 \le k \le 20\}\\
e_3 = S_2\delta S_1: & h_{e_1}(i,j,k) = (i-1, j-5, k)\\
& P_{e_1} = \{(i,j,k) \,:\, 1 \le i \le 20 \wedge 5 \le j \le 20 \wedge 1 \le k \le 20\}
\end{array}$$

Mit der Beschreibung der Abhängigkeiten stellen wir nun Gleichung (13) für e_1, e_2 und e_3 auf:

$$\begin{array}{ll}
e_1: & a_3^1 i + a_3^2 j + a_3^3 k + c_3 - (a_1^1 i + a_1^2(j-1) + a_1^3 k + c_1) = 0\\
e_2: & a_1^1 i + a_1^2 j + a_1^3 k + c_1 - (a_3^1(i-1) + a_3^2(j+4) + a_3^3 k + c_3) = 0\\
e_3: & a_1^1 i + a_1^2 j + a_1^3 k + c_1 - (a_2^1(i-1) + a_2^2(j-5) + a_2^3 k + c_2) = 0
\end{array}$$

Der Koeffizientenvergleich bzgl. aller Indizes und des konstanten Anteils liefert folgende Gleichungssysteme:

Für Abhängigkeit e_1:

$$\begin{aligned} a_3^1 - a_1^1 &= 0 \\ a_3^2 - a_1^2 &= 0 \\ a_3^3 - a_1^3 &= 0 \\ c_3 + a_1^2 - c_1 &= 0 \end{aligned}$$

Für Abhängigkeit e_2:

$$\begin{aligned} a_1^1 - a_3^1 &= 0 \\ a_1^2 - a_3^2 &= 0 \\ a_1^3 - a_1^3 &= 0 \\ c_1 + a_3^1 - 4a_3^2 - c_3 &= 0 \end{aligned}$$

Für Abhängigkeit e_3:

$$\begin{aligned} a_1^1 - a_2^1 &= 0 \\ a_1^2 - a_2^2 &= 0 \\ a_1^3 - a_2^3 &= 0 \\ c_1 + a_2^1 + 5a_2^2 - c_2 &= 0 \end{aligned}$$

Faßt man die drei Gleichungssysteme nun zu einem zusammen und übergibt sie dem vorgestellten Algorithmus, so ergibt sich folgende Substitution:

$$\left[\begin{pmatrix} a_1^1 \\ a_1^2 \\ a_1^3 \\ c_1 \\ a_2^1 \\ a_2^2 \\ a_2^3 \\ c_2 \\ a_3^1 \\ a_3^2 \\ a_3^3 \\ c_3 \end{pmatrix} \leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_1^1 \\ a_1^2 \\ a_1^3 \\ c_1 \\ a_2^1 \\ a_2^2 \\ a_2^3 \\ c_2 \\ a_3^1 \\ a_3^2 \\ a_3^3 \\ c_3 \end{pmatrix} \right]$$

wobei a_3^2, a_3^3 und c_3 die nicht eliminierten Variablen sind. Es ist bereits ausreichend nur a_3^2 gleich 1 zu setzen, um nicht-triviale Prototypen zu erhalten. Damit ergibt sich als Ergebnis:

$$\begin{aligned}
\Pi_{S_1}(i,j,k) &= 3i+j+1 \\
\Pi_{S_2}(i,j,k) &= 3i+j+9 \\
\Pi_{S_3}(i,j,k) &= 3i+j
\end{aligned}$$

5.1.2 Einführung einer Gewichtsfunktion

Im Algorithmus werden die Abhängigkeiten, deren Gleichungen als erste im Gleichungssystem auftreten, bevorzugt eliminiert. Dies kann man sich dadurch erklären, daß jede Substitution, die akzeptiert wird, zur Elimination einer Abhängigkeit beiträgt. Das Akzeptieren einer neuen Substitution ist davon abhängig, daß die neue Substitution in Verbindung mit den bereits bestehenden keine trivialen Prototypen erzeugt. Somit ist die Chance auf ein Akzeptieren von Substitutionen am Anfang des Algorithmus höher als im weiteren Verlauf.

Um die Kosten für die Kommunikation zu minimieren, sind wir daran interessiert, Abhängigkeiten zu eliminieren, die einen großen Kommunikationsaufwand erzeugen. Wir benötigen also ein Maß zur Angabe dieses Aufwands, das wir als *Kommunikationsvolumen* bezeichnen. Die Gleichungen der Abhängigkeiten werden dann nach diesem Maß absteigend geordnet dem vorgestellten Algorithmus übergeben.

Zunächst ist es naheliegend, die Anzahl von Instanzen, die Quellen für eine Abhängigkeit sind, als Maß zu wählen. Das entspricht wieder dem Problem, die Punkte eines Polyeders aufzuzählen, was i. a. sehr aufwendig ist. Deshalb beschränken wir uns auf die Dimension des Polyeders E_e, das die Menge dieser Instanzen beschreibt. Eine Voraussetzung für die Korrektheit dieses Ansatzes ist, daß der verwendete Parallelrechner über die Möglichkeit des *broadcasting* verfügt, daß also ein Wert, der von mehreren Prozessoren zur Ausführung benötigt wird, nur einmal an das Kommunikationsnetz gesendet werden muß.

Formal ist das Polyeder E_e isomorph zum Bild des Polyeders P_e bezüglich der h-Transformation h_e:

$$E_e = \{\, h_e(\vec{\jmath}) \;:\; \vec{\jmath} \in P_e \,\}$$

Für die Berechnung der Dimension des Polyeders E_e benutzt man wieder Satz 5.1.3.

Anmerkung 5.1.8 Im Fall uniformer Abhängigkeiten mit gleichem Kommunikationsvolumen und identischen Indexräumen bzgl. ihrer Quellen und Ziele ist eine weitere Verfeinerung des genannten Maßes dadurch möglich, daß man die Länge der Distanzvektoren betrachtet. Aus der Definition einer uniformen Abhängigkeit und den Festlegungen für die Polyeder P_e und E_e folgt unmittelbar, daß es anhand des Distanzvektors möglich ist, aus der Beschreibung des Indexraums sowohl das Polyeder P_e als auch das Polyeder E_e zu berechnen. Der Distanzvektor bestimmt dabei, wieweit sich die Grenzen des Indexraums bzgl. P_e und E_e verschieben. Je größer die Länge eines Distanzvektors ist, umso mehr wird P_e und E_e im Vergleich zum Indexraum eingeschränkt. Uniforme Abhängigkeiten mit kürzeren Distanzvektoren sind also solchen mit längeren Distanzvektoren in der Elimination vorzuziehen, sofern die Distanz keinen Einfluß auf die Kosten der Kommunikation hat.

Anmerkung 5.1.9 Der Nachteil bei der vorgestellten Festlegung der Gewichtsfunktion ist, daß die Auswahl der Ordnung von Abhängigkeiten mit gleichem Kommunikationsvolumen einen Einfluß auf die Elimination von Abhängigkeiten mit niedrigerem Kommunikationsvolumen haben kann. Diese Auswirkung ist aber erst nach der Ausführung des Algorithmus bekannt. Deshalb ist die Wahl von Gewichtsfunktionen noch immer Gegenstand der Forschung.

5.1.3 Bewertung der Methode

Obwohl diese Methode, wie aus der letzten Anmerkung zu sehen ist, noch nicht vollständig ausgereift ist, bietet sie dennoch eine schnelle und effiziente Möglichkeit zur Konstruktion von Allokationen.

Neben der Wahl der Gewichtsfunktion gibt es jedoch noch eine weitere Schwierigkeit. Das Ergebnis des Verfahrens ist stets eine eindimensionale Allokation. Dadurch entsteht der Nachteil, daß die in einem Programm enthaltene Parallelität nicht vollständig ausgenutzt werden kann. Eine Abhilfe für dieses Problem stellen mehrdimensionale Allokationen dar, die im nächsten Abschnitt behandelt werden.

Der vorgestellte Algorithmus ist zwar deterministisch, das Verfahren für sich jedoch nicht. Dies liegt an der Auswahl eines zu eliminierenden Koeffizienten aus einer Gleichung. Der Algorithmus wählt immer den erstmöglichen. Würde man beispielsweise im Algorithmus immer den letztmöglichen Koeffizienten wählen, so ergäbe sich für die Allokation in Beispiel 5.1.7:

$$\begin{aligned}
\Pi_{S_1}(i,j,k) &= i + \frac{1}{3}j \\
\Pi_{S_2}(i,j,k) &= i + \frac{1}{3}j + \frac{8}{3} \\
\Pi_{S_3}(i,j,k) &= i + \frac{1}{3}j - \frac{1}{3}
\end{aligned}$$

In beiden Fällen werden hier alle Abhängigkeiten eliminiert, dies gilt aber nicht allgemein.

5.2 Mehrdimensionale Allokation

Moderne Rechnerarchitekturen, die eine große Anzahl von Prozessoren verwalten, ordnen diese häufig in einem mehrdimensionalen Gitter an. Benutzt man für die Abbildung von Instanzen von Zuweisungen auf dieses Gitter eine eindimensionale Allokation, ist zusätzlich eine sogenannte *Faltungsfunktion* notwendig, die eine Transformation der virtuellen Prozessornummern in physikalische vornimmt. Zur Vermeidung dieses Zwischenschritts ist es deshalb naheliegend, eine Allokation mehrdimensional zu konstruieren. Die Komponenten einer mehrdimensionalen Allokation geben dann jeweils eine Koordinate des Prozessors in der jeweiligen Dimension an.

Neben der Anpassung der Allokation an die Rechnerarchitektur gibt es aber einen noch wesentlicheren Grund zur Berechnung mehrdimensionaler Allokationen. Bei der Verwendung von eindimensionalen Allokationen kann es vorkommen, daß die in einem Eingabe-Programm mögliche Parallelität nicht ausgeschöpft wird, oder daß die durch die Allokation festgelegte Anzahl von virtuellen Prozessoren kleiner ist, als die der physikalisch vorhandenen. Ergibt sich z. B. für eine Zuweisung mit n^k-dimensionalem Indexraum ein eindimensionaler Schedule, so können pro Zeiteinheit $O(n^{k-1})$ Instanzen der Zuweisung gleichzeitig ausgeführt werden. Existiert nun eine eindimensionale

Allokation, die eine Menge von Instanzen stets auf $O(n)$ virtuelle Prozessoren verteilt, so werden alle $O(n^{k-1})$ Instanzen auf diesen Prozessoren ausgeführt, während alle übrigen Prozessoren ohne Auftrag sind. Die vorhandene Rechenleistung wird also nicht ausgenutzt.

Wird für ein Schleifenprogramm L beispielsweise mit der Methode von Feautrier ein d_Θ-dimensionaler Schedule konstruiert, so besitzt das Bild der Allokation jeder Zuweisung $S \in \Omega_L$ im Idealfall die Dimension $p_S - d_\Theta$, wobei p_S wieder die Dimension des Indexraums D_S von S bezeichnet. Diese Größen sind i. a. nicht für alle Zuweisungen gleich und damit auch nicht gleich der Dimension des Gitters zur Prozessoranordnung. Es können also Zuweisungen existieren, die nicht die notwendige Dimension im Indexraum aufweisen, um alle Prozessoren zu belegen.

Deshalb ist entweder ein System erforderlich, das es erlaubt, die Geometrie der Prozessoranordnung bzgl. jeder Zuweisung anzupassen oder man muß für alle Zuweisungen eine Allokation mit gleicher Dimension wählen und in Kauf nehmen, daß während der Abarbeitung des gesamten Programms manche Prozessoren vorübergehend keinen Abarbeitungsauftrag erhalten. Durch den ersten Ansatz geht keine Rechenleistung verloren, andererseits wird eine Anpassung der Geometrie von den meisten Rechnern nicht unterstützt.

Wir werden im folgenden deshalb den zweiten Ansatz verfolgen [8].

5.2.1 Vorstellung des Verfahrens

Unser Ziel ist es, eine Allokation zu berechnen, welche die Instanzen jeder Zuweisung nach $\mathbb{Q}^{d_\Pi}$ abbildet, wobei d_Π die maximal mögliche Dimension darstellt:

$$d_\Pi = (\max\ S\ :\ S \in \Omega_L\ :\ p_S - d_\Theta) \tag{15}$$

Um Allokationen mit dieser Eigenschaft zu berechnen, ist lediglich eine kleine Erweiterung des Algorithmus zur Berechnung eindimensionaler Allokationen notwendig. Diese Erweiterung betrifft den Test[4] auf die Trivialität der Prototypen. In der ursprünglichen Version wird getestet, ob mindestens einer der Koeffizienten eines Prototypen nicht eliminiert ist, und damit ungleich null gesetzt werden kann. Um nun d_Π-dimensionale Allokationen zu erhalten, muß

[4] Wegen der Erweiterung dieses Tests und der damit veränderten Semantik bezeichnen wir diesen Test im folgenden als *Dimensionstest*.

für die Substitution von Koeffizienten eines Prototypen folgendes gelten. Es müssen so viele nicht-eliminierte Koeffizienten in dem Prototyp verbleiben, daß bei d_Π-maliger Besetzung aller nicht-eliminierten Variablen mit geeigneten Werten die dadurch entstehenden d_Π Komponenten der Allokation linear unabhängig sind. Ergibt sich beispielsweise nach der Substitution für einen Prototypen die Form

$$\Pi_S(i,j,k) = a_S^1 i + 0j + a_S^3 k + c_S$$

und ist die für die Allokation geforderte Dimension d_Π gleich 2, so kann beispielsweise durch eine Besetzung von a_S^1,a_S^3 und c_S mit (1 0 0) und (0 1 0) folgende zweidimensionale Allokation erzeugt werden:

$$\Pi_S(i,j,k) = \begin{pmatrix} i \\ k \end{pmatrix}$$

Die Koeffizienten eines Prototypen sind nach einer Substitution i. a. durch eine additive Kombination der nicht eliminierten Koeffizienten gegeben. Um die Frage zu entscheiden, ob genügend nicht eliminierte Koeffizienten vorhanden sind, um eine d_Π-dimensionale Allokation zu konstruieren, betrachten wir folgende alternative Darstellung eines Prototypen:

$$\Pi_S(\vec{\imath}, \vec{\alpha_S}) = M(\vec{\alpha_S}\vec{\imath})$$

$\vec{\alpha_S}$ bezeichne wieder den Vektor aller Koeffizienten von Π_S. M sei die Matrix, die aus den Zeilen der Substitutionsmatrix in (14) besteht, die Substitutionen für Koeffizienten dieses Prototypen darstellen. Mit einer Aussage der linearen Algebra gilt, daß die Dimension des Bildes von Π_S gleich dem Rang der Matrix M ist. Durch die Berechnung dieses Rangs läßt sich also die gestellte Frage beantworten.

Beispiel 5.2.1 In Beispiel 5.1.7 besitzen alle Zuweisungen denselben Indexraum mit Dimension 3. Nehmen wir an, daß für diese Zuweisungen jeweils ein Schedule mit Dimension 1 berechnet wurde, so ist $d_\Pi = 2$. Mit der Änderung des Trivialitätstests in einen Dimensionstest ergibt sich dieselbe Substitution wie in Beispiel 5.1.7 mit a_2^3, a_3^3 und c_3 als nicht-eliminierten Variablen. Besetzt man diese Koeffizienten nun mit den Werten linear unabhängiger Vektoren, etwa (1 0 0) und (0 1 0) erhalten wir folgende zweidimensionale Allokation:

$$\begin{aligned}
\Pi_{S_1}(i,j,k) &= \begin{pmatrix} 3i+j+1 \\ k \end{pmatrix} \\
\Pi_{S_2}(i,j,k) &= \begin{pmatrix} 3i+j+9 \\ k \end{pmatrix} \\
\Pi_{S_3}(i,j,k) &= \begin{pmatrix} 3i+j \\ k \end{pmatrix}
\end{aligned}$$

Aufgrund der oben beschriebenen Eigenschaften ist die Forderung nach einer d_Π-dimensionalen Allokation nicht für alle Prototypen sinnvoll. In [8] wird jedoch kein Hinweis[5] gegeben, wie eine Zuweisung, für deren Prototyp $p_S - d_\Theta$ kleiner als d_Π ist, zu behandeln ist. Wir schlagen deshalb folgende Möglichkeiten vor:

- Im Dimensionstest werden Zuweisung mit der Eigenschaft $p_S - d_\Theta < d_\Pi$ nicht berücksichtigt, abgesehen davon, daß eine Dimension größer Null garantiert wird.
- Eine Substitution wird nur dann akzeptiert, wenn bezüglich jeder Zuweisung die maximal mögliche Dimension $p_S - d_\Theta$ der Allokation erhalten bleibt.

In unserer Implementierung des Verfahrens ist es möglich, zwischen diesen Alternativen zu wählen. Welche der Möglichkeiten zu bevorzugen ist, hängt vom Einzelfall und vom verwendeten Rechnersystem ab. Während die erste Alternative weniger Restriktionen an die zu prüfende Substitution stellt und damit mehr Trennkanten erzeugt, verteilt eine Allokation, die mit der zweiten Alternative berechnet wurde, die Instanzen weiter über das Prozessorfeld und nutzt dadurch die verfügbare Rechenleistung besser aus.

[5] Feautrier schreibt in [8]: „*In that case, one will obtain a placement function, whose components are not linearly independent.*“ Diese Aussage ist jedoch nicht nachzuvollziehen, da durch den Dimensionstest lediglich auf ausreichende Dimension getestet wird, aber nicht verhindert wird, daß ein Prototyp die für ihn maximal mögliche Dimension überschreitet. Eine Reduktion der Dimension, die eine lineare Abhängigkeit bewirkt, findet somit nicht statt. Es ist also durchaus möglich, daß für eine Zuweisung eine Allokation berechnet wird, für die nicht $p_S - d_\Theta$ kleiner als d_Π gilt.

Beispiel 5.2.2 Gegeben sei folgendes Schleifenprogramm:

$$
\begin{array}{ll}
\textbf{for } i := 0 \textbf{ to } 20 \textbf{ do} & \\
\quad \textbf{for } j := 0 \textbf{ to } 20 \textbf{ do} & \\
\quad\quad \textbf{for } k := 0 \textbf{ to } 20 \textbf{ do} & \\
\quad\quad\quad a[i,j] := d[i-1, j+3]; & \{S_1\} \\
\quad\quad\quad b[i,j,k] := 0; & \{S_2\} \\
\quad\quad\quad d[i,j-1] := a[i,j-1] & \{S_3\} \\
\quad\quad \textbf{end} & \\
\quad \textbf{end} & \\
\textbf{end} &
\end{array}
$$

Mit der Methode von Feautrier zur Berechnung von Schedules erhalten wir:

$$
\begin{aligned}
\Theta_{S_1}(i,j,k) = {} & \textbf{if } 3i + j \geq 20 \textbf{ then} \\
& \quad -\tfrac{2}{3}j + \tfrac{40}{3} \\
& \textbf{else} \\
& \quad 2i \\
\Theta_{S_2}(i,j,k) = {} & 0 \\
\Theta_{S_3}(i,j,k) = {} & \textbf{if } 3i + j \geq 21 \textbf{ then} \\
& \quad -\tfrac{2}{3}j + 15 \\
& \textbf{else} \\
& \quad \textbf{if } 3i + j \geq 21 \textbf{ then} \\
& \quad\quad 2i + 1 \\
& \quad \textbf{else} \\
& \quad\quad 5i + j
\end{aligned}
$$

Wegen des nulldimensionalen Schedules für die Zuweisung S_2 und des dreidimensionalen Indexraums ist die maximal mögliche Dimension der Allokation drei. Bei den Zuweisungen S_1 und S_3 ist jedoch nur eine zweidimensionale Allokation sinnvoll. Es stellt sich also das zuvor erwähnte Problem, wie diese Zuweisungen im Dimensionstest zu behandeln sind. Wählen wir die erste Alternative, die derartige Zuweisungen außer acht läßt, ergeben sich folgende Allokationen:

$$\begin{aligned}\Pi_{S_1}(i,j,k) &= (k,0,0)\\ \Pi_{S_2}(i,j,k) &= (i,j,k)\\ \Pi_{S_2}(i,j,k) &= (k,0,0)\end{aligned}$$

Mit diesem Resultat werden beide im Schleifenprogramm auftretenden Abhängigkeiten eliminiert. Dagegen ist die Verteilung auf ein dreidimensionales Prozessorfeld für die Zuweisung S_1 und S_3 auf eine Dimension beschränkt. Führt man den Dimensionstest mit der zweiten Alternative durch ergibt sich:

$$\begin{aligned}\Pi_{S_1}(i,j,k) &= (j+4,k,0)\\ \Pi_{S_2}(i,j,k) &= (i,j,k)\\ \Pi_{S_2}(i,j,k) &= (j,k,0)\end{aligned}$$

In dieser Lösung wird lediglich eine Abhängigkeit eliminiert, die Instanzen von Zuweisung S_1 und S_3 werden nun aber auf einen zweidimensionalen Unterraum der Prozessorfeldes verteilt.

Anmerkung 5.2.3 Die für eine Allokation maximal mögliche Dimension d_Π wird mit Gleichung (15) durch die Dimension des Schedules bestimmt. Steht ein Verfahren zur Verfügung, das es erlaubt, neben ein- auch mehrdimensionale Schedules zu berechnen, so läßt sich durch die Wahl eines mehrdimensionalen Schedules die Dimension d_Π der Allokation verringern.

Ziel der Berechnung einer Allokation ist die Maximierung der Anzahl von Trennkanten und damit eine Verringerung der Kommunikationskosten. Bei empirischen Untersuchungen des vorgestellten Verfahrens zur Berechnung der Allokation fällt auf, daß eine Allokation mit geringerer Dimension mehr Trennkanten besitzt, als eine Allokation mit höherer Dimension. Das liegt daran, daß zur Einhaltung der geforderten Dimension im Algorithmus weniger Substitutionen akzeptiert werden können. Jede akzeptierte Substitution trägt aber zur Maximierung von Trennkanten bei.

Bei einer isolierten Betrachtung der Güte einer Allokation folgt also, daß für die Berechnung eine möglichst geringe Dimension zu wählen ist. Im Gegenzug verschlechtert sich aber die Laufzeit des Ziel-Programms, weil durch einen mehrdimensionalen Schedule die Parallelität im Eingabe-Programm i. a. nicht voll ausgenutzt wird. Hier muß also in Abwägung der Laufzeit und der Kosten für eine Kommunikation für ein konkretes Rechnersystem ein Mittelweg gefunden werden.

5.2.2 Bewertung der Methode

Da sich die Berechnung mehrdimensionaler Allokationen direkt aus der für eindimensionale Allokation ableitet, gelten die gleichen Aussagen bezüglich der Wahl einer Gewichtsfunktion und des Nichtdeterminismus. Darüberhinaus entsteht neuer Nichtdeterminismus bei der Besetzung der nicht-eliminierten Koeffizienten. Bei mehrdimensionalen Allokationen wirkt sich dieser Nichtdeterminismus stark auf die Auslastung des Prozessorfeldes aus. Aus diesem Grund wird derzeit am Laboratoire MASI der Universität Versaille erforscht, wie die verwendete Rechnerarchitektur in das Berechnungsverfahren miteinzubeziehen ist.

Abgesehen von den beschriebenen Problemen stellt diese Methode aber eine sehr schnelle Möglichkeit zur Berechnung von Allokationen dar.

6 Erzeugung des parallelen Programms

In diesem Abschnitt wird eine allgemein anwendbare Methode zur Konstruktion des parallelen Programms gemäß Schedule und Allokation vorgestellt [8]. Eine Diskussion über ausgereiftere Methoden findet sich in [15].

Die Idee des Verfahrens ist, sämtliche Kombinationen von Zeit- und Raumpunkten aufzuzählen und zur Laufzeit für jede aufgezählte Instanz einer Zuweisung die Ausführung zu prüfen. Die Steuerung, die von Schedule und Allokation vorgegeben ist, wird damit explizit in den Zielcode integriert.

Dazu wird zunächst die Menge von Instanzen von Zuweisungen eines Schleifenprogramms L berechnet, die zu einem gegebenen Zeitpunkt ausgeführt werden sollen. Eine derartige Menge wird als *Front* [8] bezeichnet. Für eine Front gilt also:

$$\mathcal{F}(t) = \{(S, \vec{\imath}) \,:\, S \in \Omega_L \wedge \vec{\imath} \in D_S \wedge \Theta(S, \vec{\imath}) = t\}$$

Alle Instanzen einer Front können parallel ausgeführt werden und entsprechend der Allokation Π über das Prozessorfeld verteilt werden. Für die Ausführung von Instanzen einer Front auf einem Prozessor q ergibt sich also folgende Teilmenge einer Front $\mathcal{F}(t)$:

$$\mathcal{F}(t, q) = \{(S, \vec{\imath}) \,:\, S \in \Omega_L \wedge \vec{\imath} \in D_S \wedge \Theta(S, \vec{\imath}) = t \wedge \Pi(S, \vec{\imath}) = q\}$$

Ist l die Laufzeit des parallelen Programms bezüglich Θ und p die Anzahl von Prozessoren bezüglich Π, so kann daraus das parallele Programm wie in Abbildung 8 aufgebaut werden. Es kann durchaus vorkommen, daß auf einem Prozessor zum selben Zeitpunkt mehrere Instanzen von Zuweisungen abgearbeitet werden müssen, daß also die Mengen $\mathcal{F}(t, q)$ nicht einelementig sind. Das gilt insbesondere, wenn Schedule und Allokation, wie in den letzten beiden Abschnitten, unabhängig voneinander berechnet wurden. Deshalb wurde die innerste **seqfor**-Schleife eingefügt. Zu einem Konflikt führt dies allerdings nicht, da der Schedule die Unabhängigkeit garantiert. Die Reihenfolge der Abarbeitung ist dabei freigestellt. Lediglich die Prozessorauslastung des ganzen Systems verschlechtert sich in einem solchen Fall, da andere Prozessoren auf die Freischaltung durch *barrier* warten müssen. Die Zeitschleife kann dann allerdings nicht mehr als Systemzeit interpretiert werden. Sie stellt eine virtuelle, grobkörnige Zeit dar. Eine Implementierung dieses Verfahrens ist durch Prozesse möglich. Ein Prozeß besteht dann jeweils aus der Berechnung von Instanzen einer Menge $\mathcal{F}(t, q)$. Die Synchronisation erfolgt über

```
parfor q := 0 to p do
        seqfor t := 0 to l do
                {read remote data}
                seqfor
                        F(t, q)
                end
                {barrier}
        end
end
```

Abbildung 8: Explizite Steuerung durch $\mathcal{F}(t, q)$

die Konstrukte *read remote data* und *barrier* jeweils vor und nach den zu berechnenden Instanzen.

Anmerkung 6.0.4 Die Berechnung der Werte p und l für eine Allokation Π und einen Schedule Θ kann mit Hilfe von PIP durchgeführt werden, indem man die in Abschnitt 4.2.3 vorgestellten Methoden zur Lösung eines Maximierungsproblems benutzt. Die Abbildungen Π und Θ stellen jeweils die Zielfunktion dar, während die Beschreibung des Indexraums das Ungleichungssystem $G\vec{\mu} \geq \vec{h}$ bildet.

Die Frage, ob die Mengen $\mathcal{F}(t, q)$ einelementig sind, kann durch eine Betrachtung von Schedule und Allokation entschieden werden. Ist die Matrix

$$T = \begin{pmatrix} \Theta(S, \vec{\imath}) \\ \Pi(S, \vec{\imath}) \end{pmatrix}$$

für jede Zuweisung S invertierbar, so folgt, daß die Mengen $\mathcal{F}(t, q)$ einelementig sind. Die Matrix T ist eine Transformationsmatrix, wie sie in Abschnitt 4.1 eingeführt wurde. Für eine solche Matrix ist es möglich die Mengen $\mathcal{F}(t, q)$ durch die Anwendung der inversen Matrix T^{-1} bereits zur Compilezeit zu berechnen [13]. Gilt dies nicht, so wird in der konkreten Implementierung des Zielcodes die Berechnung der Mengen $\mathcal{F}(t, q)$ dadurch realisiert, daß zunächst der ursprüngliche Code anstelle von $\mathcal{F}(t, q)$ eingesetzt wird. Alle Schleifen des

Eingabe-Programms werden jedoch durch **parfor**-Schleifen ersetzt. Durch ein umgebendes **if**-Konstrukt, das Schedule und Allokation beinhaltet, wird dann zur Laufzeit auf die Ausführung getestet.

Folgendes Beispiel zeigt die Programmkonstruktion nach diesem Verfahren:

Beispiel 6.0.5 Gegeben sei folgendes Schleifenprogramm:

```
for i := 1 to 10 do
    for j := i to i + n + 5 do
        b[i] := b[i] + 1
    end
end
```

Mit den statementbasierten Methoden aus Abschnitt 2 und 3 berechnet man folgende Abbildungen als Schedule und Allokation:

$$\Theta(i, j) = j - i$$
$$\Pi(i, j) = i$$

Damit entsteht folgender Zielcode:

```
parfor q := 0 to 10 do
    seqfor t := 0 to n + 14 do
        {read remote data}
        parfor i := 1 to 10 do
            parfor j := i to i + n + 5 do
                if j - i = t and i = q then
                    b[i] := b[i] + 1
                fi
            end
        end
        {barrier}
    end
end
```

7 Schlußbemerkungen

Mit den vorgestellten Methoden zur Berechnung eines Schedules und einer Allokation sind die Hauptbestandteile zur Parallelisierung eines sequentiellen Programms gegeben.

Wir haben für den Bereich Schedule zwei Verfahren kennengelernt. Dabei steht die Hyperebenen-Methode für ein Verfahren mit großen Einschränkungen an die Eingabe-Programme. Sie besitzt jedoch in der Form von Lamport eine sehr kurze Laufzeit. Die Hyperebenen-Methode wurde vielfach weiterentwickelt, etwa zum statement-orientierten Ansatz [4]. Ihre größte Einschränkung stellt jedoch die Beschränkung auf uniforme Abhängigkeiten und perfekte Verschachtelung dar.

Eine Methode ohne derartige Beschränkungen ist mit dem Verfahren von Feautrier gegeben. Es liefert unter den genannten Einschränkungen sogar zeitoptimale Schedules. Während die Methode für sich ebenfalls sehr schnell arbeitet, besteht der Großteil der benötigten Laufzeit aus der Lösung der Optimierungsprobleme mit PIP. Ein Problem beim Einsatz von PIP ist außerdem, daß es bei größeren Zahlen in der Eingabe nicht mehr stabil arbeitet. Dieses Programm ist aber bislang als einziges in der Lage, parametrisierte Probleme mit akzeptablem Zeitaufwand zu lösen.

Für den Bereich der Allokation wurde eine Methode vorgestellt, die ebenfalls geringe Einschränkungen für die Eingabe-Programme fordert. Diese Methode ist allerdings noch nicht vollständig ausgereift, da sie beispielsweise die verwendete Rechnerarchitektur nicht in die Berechnung miteinbezieht. Ebenfalls ist die Wahl einer geeigneten Gewichtsfunktion noch zu untersuchen.

Die vorliegende Arbeit sollte einerseits die notwendige Theorie der einzelnen Verfahren vermitteln, andererseits aber auch die Grundlage für eine Implementierung geben. Die vorgestellten Methoden wurden im Rahmen des Projekts LooPo (Loop Parallelization in the Polytope Model) implementiert. LooPo ist ein Programm zur Parallelisierung von sequentiellen Programmen und soll auch einen kleinen Teil zur Entwicklung des Gebiets der Parallelisierung beitragen.

Literatur

[1] Uptal Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1993.

[2] G. B. Dantzig. *Maximization of a linear function of variables subject to linear inequalities*, in: Activity Analysis of Production and Allocation, Wiley, New York, 1951, pp. 339-347.

[3] Alain Darte, Leonid Khachiyan, Yves Robert. *Linear Scheduling is nearly optimal*. Parallel Processing Letters, Vol. 1, No. 2, pp. 73-81, 1991.

[4] Alain Darte, Yves Robert. *Constructive Methods for Scheduling Uniform Loop Nests*. IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 8, August 1994.

[5] E. W. Dijkstra, C. S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[6] Paul Feautrier. *Some efficient solutions to the affin scheduling problem, Part I, One-dimensional Time*. International Journal of Parallel Programming, Vol. 21, No. 5, pp. 313-347, 1992.

[7] Paul Feautrier. *Some efficient solutions to the affin scheduling problem, Part II, Multidimensional Time*. International Journal of Parallel Programming, Vol. 21, No. 6, pp. 389-420, 1992.

[8] Paul Feautrier. *Toward Automatic Distribution*. Parallel Processing Letters, Vol. 4, No. 3, pp. 233-244, 1994.

[9] Paul Feautrier. *Dataflow Analysis of Array and Scalar References*. International Journal of Parallel Programming, Vol. 20, pp. 23-53, 1991.

[10] Paul Feautrier. *Résolution de systèmes d'inéquations linéaire; mode d'emploi du logiciel PIP*. Technical Report 90.2, IBP-MASI, Paris, 1990.

[11] J. A. B. Fortes, F. Parisi-Presicce, *Optimal linear schedules for the parallel execution of algorithms*. Int. Conf. Parallel Processing, August 1984, pp. 322-328.

[12] Leslie Lamport. *The Parallel Execution of DO Loops*. Communications of the ACM, Vol. 17, No. 2, pp. 83-93, February 1974.

[13] Christian Lengauer. *Loop Parallelization in the Polytope Model*. In E. Best, editor, CONCUR '93, Lecture Notes in CS 715, pp. 398-416, Springer-Verlag, 1993.

[14] Alexander Schrijver. *Theory of Linear and Integer Programming*. Series in Discrete Mathematics. John Wiley & Sons, 1986.

[15] Sabine Wetzel. *Automatic Code Generation in the Polyhedron Model*. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1995, in Vorbereitung.