UNIVERSITÄT PASSAU
Fakultät für Mathematik und Informatik

Partitionierung von parallelen Schleifenprogrammen

Diplomarbeit

Verfasser: Martina Wieninger

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

Betreuer:
Dr. Martin Griebl

Passau, den 30.09.1997

Vorab möchte ich mich bei allen bedanken, die zum Gelingen dieser Diplomarbeit beigetragen haben. Herrn Prof. Christian Lengauer Ph. D. danke ich für seine Anregungen und Verbesserungsvorschläge zu dieser Arbeit, sowie für seine spontane Hilfsbereitschaft, wenn sich zeitliche Probleme ergaben. Desweiteren bin ich natürlich Dr. Martin Griebl für seine intensive Betreuung während der Erstellung dieser Arbeit zu großem Dank verpflichtet. Mein besonderer Dank gilt meinem Mann und ehemaligen LooPo-Mitglied, Christian Wieninger, der stets ein offenes Ohr für alle Probleme hatte und zu ihrer Lösung beitrug. Außerdem danke ich allen Mitgliedern des LooPo-Teams für viele Diskussionen und Anregungen in einem angenehmen Arbeitsklima: Sven Anders, Andreas Dischinger, Nils Ellmenreich, Peter Faber, Max Geigl, Robert Günz, Harald Keimer, Radko Kubias, Wolfgang Meisl, Sabine Wetzel, die mir eine große Hilfe bei der Einarbeitung in ihren Code war, und Alexander Wüst. Schließlich sei allen Angehörigen des Lehrstuhls für Programmierung für ihre Hilfsbereitschaft gedankt. Zuletzt möchte ich noch meinen Eltern danken, die mir diese Ausbildung überhaupt ermöglicht haben.

Zusammenfassung

Compiler zur automatischen Parallelisierung von sequentiellen Programmen bauen häufig auf Allokationsverfahren zur Verteilung der Berechnungen auf ein Mehrprozessorsystem auf, die von einer unbegrenzten Anzahl an virtuellen Prozessoren ausgehen. Im Zuge dessen ist eine Partitionierung von parallelen Schleifensätzen nötig, um die Ausführung des parallelisierten Programms auf einer realen Rechnerarchitektur basierend auf einem Prozessorarray mit fester Ausdehnung zu ermöglichen. Dabei ist ein weiteres Ziel, die Berechnungspunkte bei der Partitionierung so zu gruppieren, daß möglichst wenig Kommunikationsaufwand entsteht. Grundsätzlich existieren zwei Verfahren zur Partitionierung, zum einen die Gruppierung der Berechnungspunkte in Partitionen, die lokal parallel ausgeführt werden (LPGS), zum anderen die Gruppierung in lokal sequentiell auszuführende Partitionen (LSGP).

In dieser Arbeit werden Methoden für beide Prinzipien vorgestellt. Im Falle der LSGP-Partitonierung wird ein zur Partitionierung sequentieller Programme bereits existierendes Verfahren auf die Partitionierung paralleler Schleifensätze erweitert. Im Falle der LPGS-Partitionierung leiten wir Kriterien zur Partitionierung mit minimalem Kommunikationsaufwand her und entwickeln daraus einen Algorithmus, der diese Kriterien neben dem Aspekt der Ausführbarkeit des parallelen Programms auf einem realen Rechner erfüllt.

Inhaltsverzeichnis

1 Einführung

Die Partitionierung wird in der Literatur über Programmtransformationen als eine Methode zur Gruppierung von Iterationen eines sequentiellen Schleifenprogramms diskutiert, mit dem Ziel

- diese Gruppen auf einer parallelen Rechnerarchitektur mit minimaler Kommunikation zwischen den Prozessoren auszuführen, und
- durch die Größe der Gruppen minimalen Datentransfer zwischen den Schichten einer Speicherhierarchie zu erreichen.

Im Rahmen des Projekts *LooPo* (Loop Parallelization in the Polytope Model) an der Universität Passau wurde ein Programm zur automatischen Parallelisierung von sequentiellen Programmen entwickelt. Bei den implementierten Allokationsverfahren zur Verteilung der Berechnungen auf ein Mehrprozessorsystem wird dabei stets von einer unbegrenzten Anzahl an virtuellen Prozessoren ausgegangen. Um eine Ausführung des parallelisierten Programms auf einer realen Rechnerarchitektur, der ein Prozessorarray zugrunde liegt, zu ermöglichen, ist eine Partitionierung von parallelen Schleifensätzen nötig, die

- zum einen parallele Iterationen so gruppiert, daß das Programm mit einer endlichen Anzahl an realen Prozessoren ausführbar ist,
- zum anderen das Ziel verfolgt, die Iterationen so zu gruppieren, daß zwischen den realen Prozessoren möglichst wenig Kommunikationsaufwand entsteht.

Diese Arbeit stellt Methoden vor, die diesen beiden Forderungen entsprechen. Nach der Einführung einiger grundlegender Definitionen gehen wir in Kapitel 3 auf die Eingliederung der Partitionierung in das LooPo-Projekt ein. Für die Partitionierung unterscheiden wir grundsätzlich zwei Ansätze, wobei in der Literatur lediglich für einen eine Vielzahl an Verfahren existiert. Deshalb leiten wir in Kapitel 4 als Schwerpunkt dieser Arbeit Verfahren für den zweiten Ansatz her und schließen in Kapitel 5 mit der Vorstellung und Erweiterung einer exemplarischen Methode des ersten Ansatzes.

2 Grundlagen

2.1 Notation

Da im Zusammenhang mit dem Polyedermodell die lexikografische Ordnung eine wesentliche Bedeutung hat, wollen wir im folgenden ihre Notation festlegen.
Dazu definieren wir zunächst den Operator $\prec_l$:

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

Mit dieser Schreibweise ergibt sich die Notation der lexikografischen Ordnung durch den Operator $\prec$:

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

Analog benutzen wir die Operatoren $\succ_l$ und $\succ$.

Mit $\vec{a} \times \vec{b}$ bezeichnen wir das Vektorprodukt der Vektoren $\vec{a}$ und $\vec{b}$.

2.2 Mathematische Definitionen

Für eine formale Herleitung und Darstellung der Ergebnisse benötigen wir einige mathematische Definitionen aus der Theorie des Polyedermodells.

Definition 2.2.1 (Ebene eines Vektors) Die *Ebene eines Vektors* $\vec{\imath} \in \mathbb{R}^n$ hat den Wert $l \in \mathbb{N}$, i.Z. $lev(\vec{\imath}) = l$, falls gilt:

$$\vec{\imath} \succ_l \vec{0}$$

Da wir uns in dieser Arbeit mit Mengen beschäftigen, deren graphische Darstellung durch spezielle geometrische Eigenschaften gegeben ist, benötigen wir den Begriff der Konvexität. Anschaulich heißt eine Menge konvex, wenn eine Strecke zwischen beliebigen Punkten der Menge wieder in der Menge liegt. Formal:

Definition 2.2.2 (konvex) Sei $M \in \mathbb{R}^n$, $\alpha \in [0,1] \subset \mathbb{R}$. M heißt *konvex*, gdw. gilt:

$$(\forall\, \vec{x}, \vec{y} \,:\, \vec{x}, \vec{y} \in M \;:\; (\alpha x + (1-\alpha) y) \subseteq M)$$

Definition 2.2.3 (Halbraum) Sei $H \subseteq \mathbb{R}^n$, $a_1, \ldots, a_n, b \in \mathbb{R}$. Ein *Halbraum* H ist die Menge

$$H = \{(x_1, \ldots, x_n) : (x_1, \ldots, x_n) \in \mathbb{R}^n \wedge a_1x_1 + \ldots + a_nx_n \leq b\}$$

Definition 2.2.4 (Polyeder) Seien $H_1, \ldots, H_m$ $(m \in \mathbb{N})$ Halbräume. Eine Menge $P \subset \mathbb{R}^n (n \in \mathbb{N})$ heißt *Polyeder*, gdw. gilt:

$$P = \bigcap_{i=1}^{n} H_i$$

Ein Polyeder ist also der Durchschnitt einer endlichen Menge von Halbräumen. Erfüllt ein Polyeder zusätzlich die Eigenschaft der Beschränktheit, so bezeichnen wir ihn als Polytop:

Definition 2.2.5 (Polytop) Sei P ein Polyeder. Dann heißt P *Polytop* oder *beschränktes Polyeder*, wenn gilt:

$$\begin{gathered}(\exists w : w \in \mathbb{R}^+ : P = \{\, (x_1, \ldots, x_n) : (x_1, \ldots, x_n) \in \mathbb{R}^n \wedge \\ (\forall\, j \;:\; 1 \leq j \leq n \;:\; -w \leq x_j \leq +w)\,\})\end{gathered}$$

Anmerkung 2.2.6 Halbräume und damit Polyeder und Polytope sind konvex. Zur Darstellung dieser Menge in einer kompakten Form benutzen wir folgende Matrix-Schreibweise:

$$A\vec{x} \leq \vec{b}$$

wobei A die Koeffizientenmatrix des Ungleichungssystem ist, das den Polyeder definiert. Der Vektor $\vec{b}$ entspricht dem konstanten Anteil.

Das Polytop ist ein elementarer Gegenstand im klassischen Polyedermodell. Im Loopo-Projekt benötigen wir daneben noch zur Code-Generierung eine spezielle Form von Polytopen:

Definition 2.2.7 (quasi-konvexes Polytop) Eine Menge P heißt *quasi-konvexes Polytop*, wenn sie die Vereinigung einer endlichen Menge von Polytopen P_i, $1 \leq i \leq n$, ist. Formal:

$$P = \bigcup_{i=1}^{n} P_i$$

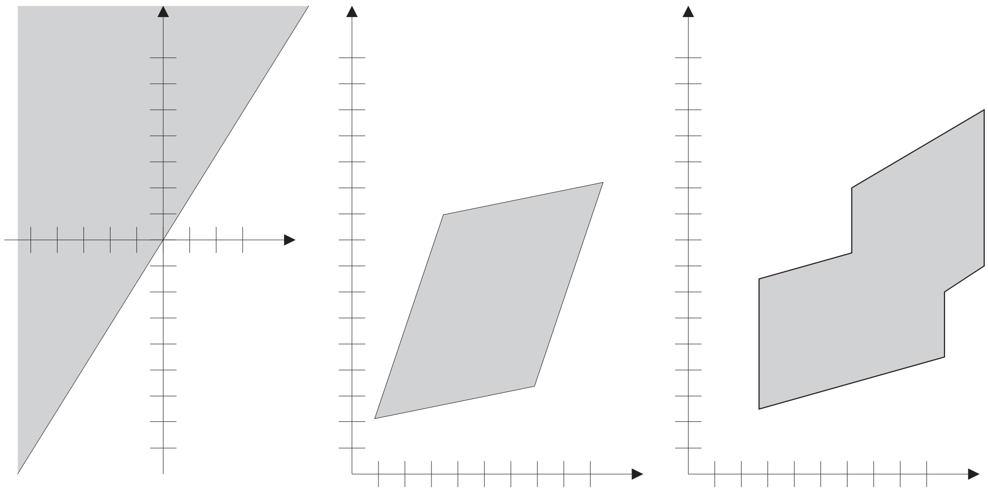

Abbildung 1: a) Halbraum, b) Polytop, c) quasi-konvexes Polytop

Definition 2.2.8 (Rahmen eines Ungleichungssystems) Sei

$$A\vec{x} < \vec{b}$$

ein Ungleichungssystem. Dann wird der *Rahmen* des Ungleichungssystems durch das Gleichungssystem

$$A\vec{x} = \vec{b}$$

definiert.

Zur Verallgemeinerung von Parallelogrammen auf $\mathbb{R}^n$ definieren wir das Parallelotop [5].

Definition 2.2.9 (Parallelotop) Ein n-dimensionales Polytop in $\mathbb{R}^n$ heißt *Parallelotop*, wenn sein beschreibendes Ungleichungssystem in der folgenden Form dargestellt werden kann:

$$\begin{pmatrix} A \\ -A \end{pmatrix} \vec{x} \leq \begin{pmatrix} \vec{b} \\ \vec{c} \end{pmatrix}, \; A \in \mathbb{R}^{n\times n}, \; \vec{x}, \vec{b}, \vec{c} \in \mathbb{R}^n$$

D.h. zu jeder Seite eines Parallelotops existiert eine zweite, die parallel zur ersten verläuft.

2.3 Indexraum und Abhängigkeit

Aufgabe dieses Abschnitts ist die Definition von Begriffen, die Objekte und Eigenschaften von Programmen beschreiben. Zur Vereinfachung bezeichne im folgenden P ein Programm, S ein Statement und Ω_P die Menge aller Statements S_i, $i \in \mathbb{N}$, im Programm P.

Definition 2.3.1 (Indexraum) Seien $i_1, \ldots, i_m$ die m umgebenden Schleifenindizes eines Statements S, l_j bzw. u_j die untere bzw. die obere Grenze eines Index i_j, $1 \leq j \leq m$. Dann ist der *Indexraum* D_S eines Statements S die Menge aller Indexvektoren $\vec{\imath}$, für die gilt:

1. $(i_1, \ldots, i_m) \in \mathbb{Z}^m$ und
2. $(\forall j : 1 \leq j \leq m : l_j \leq i_j \leq u_j)$

Anmerkung 2.3.2 Aus der Definition von Indexräumen wird ersichtlich, daß die Menge D_S ein Polytop ist.

Definition 2.3.3 (Perfekte Verschachtelung) Ein Programm P ist *perfekt verschachtelt*, wenn es 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**
    $\vdots$
    for $i_n := l_n(i_1, \ldots, i_{n-1})$ **to** $u_n(i_1, \ldots, i_{n-1})$ **do**
      {Schleifenkörper}

Der Schleifenkörper besteht aus einer beliebigen Folge von Statements S_i.

Anmerkung 2.3.4 Bei perfekter Verschachtelung eines Programms P besitzen alle Statements S_i von P denselben Indexraum D.

Definition 2.3.5 (Instanz) Sei $\vec{\imath} \in D_S$ ein Indexvektor eines Statements S. Dann heißt das Tupel $(S, \vec{\imath})$ *Instanz* des Statements S mit Indexvektor $\vec{\imath}$.

Die Semantik eines Programms wird unter anderem durch die Abhängigkeiten seiner Statements bestimmt. In der Literatur finden sich unterschiedliche Definitionen des Abhängigkeitsbegriffs. Wir verwenden folgende Festlegung [2]:

Definition 2.3.6 (Abhängigkeit) Seien S und T zwei nicht notwendigerweise verschiedene Statements.
T *hängt* von S *ab*, i.Z. $S\delta T$, gdw. gilt:

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

2. $(S,\vec{\imath})$ wird im Programm vor $(T,\vec{\jmath})$ ausgeführt und

3. vom Ende von $(S,\vec{\imath})$ bis zum Anfang von $(T,\vec{\jmath})$ wird nicht schreibend auf M zugegriffen.

Definition 2.3.7 (uniforme Abhängigkeit) Eine Abhängigkeit $S\delta T$ bezeichnet man als *uniform*, falls gilt:

1. $D_S = D_T$, $D_S, D_T \in \mathbb{Z}^n$

2. Es existiert ein lexikografisch positiver konstanter Vektor $\vec{d} \in \mathbb{Z}^n$, der die Differenz zwischen den Indexvektoren der abhängigen Instanzen von S und T ist. Formal:

$$(\forall \vec{\imath}, \vec{\jmath} : \vec{\imath} \in D_S, \vec{\jmath} \in D_T, \vec{d} \in \mathbb{Z}^n, \vec{d} \succeq \vec{0} : (S,\vec{\imath})\delta(T,\vec{\jmath}) \Rightarrow \vec{d} = \vec{\jmath} - \vec{\imath})$$

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

Definition 2.3.8 (Abhängigkeitsebene, Tragende Schleife) Sei $\vec{d} \in \mathbb{Z}^n$ der Distanzvektor einer uniformen Abhängigkeit in einem Programm mit n Schleifen. Dann bezeichnet $l = lev(\vec{d})$ die *Abhängigkeitsebene* des Distanzvektors.
Hängt ein Statement T von Statement S mit Distanzvektor $\vec{d}$ ab und ist $l = lev(\vec{d})$, $l \in (1,\ldots,n)$, dann wird diese Abhängigkeit von Schleife L_l *getragen.*

2.4 Transformation des Programms

Ausgangspunkt für diese Arbeit ist ein paralleles Programm, das durch die Transformation eines sequentiellen entsteht. Dieser Abschnitt soll kurz das Prinzip dieser Transformation erläutern, deren Kern im Modul Target Loops[1] implementiert ist. Eingabe für dieses Modul sind folgende zwei Abbildungen:

Definition 2.4.1 (Schedule) Die Abbildung

$$\Theta_S : (\Omega_P \times D_S) \to \mathbb{Q}^k$$

weist jeder Instanz eines Statements S eine k-dimensionale Ausführungszeit zu. Diese Abbildung heißt *Schedule*, falls sie folgende Eigenschaft erfüllt:
Für jedes abhängige Paar von Statements S, $T \in \Omega_P$ muß gelten:

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

Anmerkung 2.4.2 Das Modul Scheduler in LooPo bietet auch die Möglichkeit von stückweise definierten Schedules, z.B. :

$$\text{if} \quad i > j \quad \text{then} \quad i \quad \text{else} \quad j$$

Definition 2.4.3 (Allokation) Die Abbildung

$$\Pi_S : (\Omega_P \times D_S) \to \mathbb{Q}^k$$

weist jeder Instanz eines Statements S einen Prozessor eines k-dimensionalen Prozessorarrays zu. Diese Abbildung heißt *Allokation*.

Anmerkung 2.4.4 Bei der Allokation wird von einer beliebigen Anzahl an virtuellen Prozessoren je Dimension ausgegangen. Bei LooPo kann die gewünschte Dimensionalität des Prozessorarrays bei der Berechnung der Allokation berücksichtigt werden.

Schedule und Allokation bilden die Basis für die Transformation des Quellprogramms, die durch folgende Abbildung dargestellt wird:

[1] *Target Loops* bezeichnet die Implementierung der in [22] vorgestellten Verfahren im Rahmen des LooPo-Projekts.

Definition 2.4.5 (Transformationsmatrix) Sei Θ ein k-dimensionaler Schedule eines Statements S, Π eine l-dimensionale Allokation desselben Statements. Dann ist die *Transformationsmatrix* T die Matrix, deren Zeilen die Koeffizienten von Schedule und Allokation sind.

Anmerkung 2.4.6 Ist der Schedule wie in Anmerkung 2.4.2 stückweise definiert, so wird auch die Transformation stückweise festgelegt, z.B. :

$$\text{if } i > j \text{ then } \begin{pmatrix} i \\ k \end{pmatrix} \text{ else } \begin{pmatrix} j \\ k \end{pmatrix}, \text{wobei } \Pi(i,j,k) = k$$

Eine solche Transformation heißt *stückweise affin*. Handelt es sich um eine nicht stückweise Transformation, wird diese als *elementar* bezeichnet.

In Verbindung mit dem Begriff der Transformationsmatrix benötigen wir noch die Definition der Unimodularität von Matrizen:

Definition 2.4.7 (unimodular) Eine Matrix $M \in \mathbb{N}^{n \times n}$, $n \in \mathbb{N}$ heißt *unimodular*, wenn

$$det(M) = \pm 1$$

Bei den in [22] diskutierten Methoden werden im wesentlichen zwei Verfahren unterschieden:

1. Code-Erzeugung auf der Basis von iterations-basierten Transformationen, wie z.B. bei Lamport [14]. Dieses Verfahren beschränkt sich auf perfekt verschachtelte Eingabeprogramme mit ausschließlich uniformen Abhängigkeiten.

2. Code-Erzeugung auf der Basis von statement-basierten Transformationen, wie z.B. bei Verfahren von Feautrier [8]. Diese Methoden arbeiten sowohl auf nicht perfekt verschachtelten Programmen, als auch mit affinen Abhängigkeiten.[2]

[2]Eine Abhängigkeit heißt *affin*, falls die Relation zwischen abhängigen Instanzen durch eine affine Abbildung dargestellt werden kann.

Die Code-Erzeugung für den ersten Fall gestaltet sich relativ einfach, da der Lamport-Scheduler und -Allokator sehr restriktive Verfahren sind. Man erhält immer eine elementare, unimodulare Transformationsmatrix, deren Inverse ganzzahlig ist.
Für ein k-dimensionales Schleifenprogramm mit Schleifenindizes $i_1, \ldots, i_k$ gilt folgende Gleichung:

$$\vec{\imath} = T^{-1}\vec{\jmath} \qquad \vec{\imath}, \vec{\jmath} \in \mathbb{N}^k, T \in \mathbb{N}^{k \times k},$$

wobei $\vec{\jmath}$ die Indizes der Zielschleifen darstellen.

Diese Gleichung stellt jeden Quell-Index i als Linearkombination der Ziel-Indizes dar. Um das Zielprogramm zu erhalten, muß jeder Quell-Index durch die zugehörige Linearkombination ersetzt werden. Man erhält die Schleifengrenzen des Zielprogramms durch Fourier-Elimination. Anschließend müssen alle Quell-Indizes, die in Statements des Programms auftauchen, durch die entsprechenden Linearkombinationen der Ziel-Indizes ersetzt werden. Eine genauere Beschreibung dieser Methode findet sich in [22].

Die Code-Erzeugung für den zweiten Fall gestaltet sich weitaus schwieriger. Es treten Probleme wie nicht-unimodulare Transformationsmatrizen und stückweise Transformationen auf. Da statement-basierte Methoden in der Regel auch auf nicht perfekt verschachtelten Programmen arbeiten, muß die Code-Erzeugung außerdem auch diesen Programmtyp abdecken.

Die Lösung dieser Probleme besteht darin, das Programm in mehrere perfekt verschachtelte Programmteile zu zerlegen:

Definition 2.4.8 (Programmteil) Ein perfekt verschachteltes Programm mit genau einem Statement wird als *Programmteil* bezeichnet. Jeder Programmteil besitzt genau einen Schedule und eine Allokation.

Das Quell-Programm wird zuerst in Programmteile aufgeteilt, von denen jeder ein Statement des Programms enthält. Alle Programmteile werden gemäß des zugehörigen Schedules und der Allokation in Ziel-Programmteile transformiert, die nun die transformierten Statements enthalten. Aus einem Quell-Programmteil können sich durch stückweise Transformationen mehrere Ziel-Programmteile ergeben.

Das vollständige Zielprogramm, das sowohl *synchron* (sequentielle Schleifen außen, parallele Schleifen innen) als auch *asynchron* (parallele Schleifen außen, sequentielle Schleifen innen) erzeugt werden kann, entsteht durch die

Vereinigung aller Ziel-Programmteile. Da jeder Programmteil geometrisch durch ein Polytop dargestellt werden kann, ergibt sich durch die Vereinigung aller Ziel-Programmteile ein m-dimensionales quasi-konvexes Polytop, wobei m das Maximum der Dimensionen aller Programmteile sei. Durch dieses Polytop wird der Ziel-Indexraum beschrieben.

Wie bereits in der Einleitung zu diesem Abschnitt erwähnt, ist der Ausgangspunkt der Partitionierung ein paralleles Programm. Ziel der Partitionierung ist die Lauffähigkeit des Programms auf einem Parallelrechner mit beschränkter Prozessoranzahl. Somit interessiert nur der parallele Anteil des Programms, der durch den parallelen Indexraum repräsentiert wird.

Definition 2.4.9 (Paralleler Indexraum) Der *parallele Indexraum* D_S^{par} eines Statements S ist die Projektion des Indexraums D_S auf die Allokations-Dimensionen.
Bei perfekter Verschachtelung bezeichne D^{par} den für alle Statements eines Programms gleichen parallelen Indexraum.

Da die einzelnen Ziel-Programmteile nicht notwendigerweise dieselbe Anzahl an Schleifen besitzen müssen, kann sich auch ein nicht perfekt verschachteltes Zielprogramm ergeben. Eine Forderung des Moduls Target Loops ist jedoch, daß die Allokationen für alle Statements eines Programms dieselbe Dimensionalität haben müssen.[3] Dadurch erreicht man im Zielprogramm stets einen perfekt verschachtelten parallelen Indexraum.

Von besonderem Interesse für diese Arbeit sind die Ausdrücke, die das Zielprogramm als Schleifengrenzen enthalten darf.

Sei l_{j_i} die Untergrenze der j-ten Schleife eines Ziel-Programmteils P_i, u_{j_i} analog die Obergrenze dieser Schleife. Dann dürfen l_{j_i} bzw. u_{j_i} folgende Form haben:

$$\begin{aligned} l_{j_i} &= \max(expr_1, \ldots, expr_r),\ r \in \mathbb{N} \\ u_{j_i} &= \min(expr_1{}', \ldots, expr_l{}'),\ l \in \mathbb{N} \end{aligned}$$

[3]In LooPo wird dies folgendermaßen erreicht: Sei m die maximale Dimensionaltität aller Allokationen. Für eine Allokation Π mit Dimensionalität $r < m$ erweitern wir Π zu Π' durch

$$\Pi'(\mathrm{i}_1, \ldots, \mathrm{i}_\mathrm{n}) = (\Pi(\mathrm{i}_1, \ldots, \mathrm{i}_\mathrm{n}), \underbrace{0, \ldots, 0}_{\mathrm{m-r}})$$

$expr_i$ und $expr'_i$ sind affine Ausdrücke in Schleifenindizes und Strukturparametern[4]. Somit kann eine Schleife i_j von einer äußeren Schleife i_l, $l < j$, abhängen. Produkte von Schleifenindizes und Strukturparametern sind laut Definition von Feautrier [8] nicht affin, und damit nicht erlaubt.

Durch die Vereinigung der k Ziel-Programmteile ergeben sich als mögliche Unter- bzw. Obergrenzen:

$$\begin{aligned} l_j &= \min(l_{j_1}, \ldots, l_{j_k}) \\ u_j &= \max(u_{j_1}, \ldots, u_{j_k}) \end{aligned}$$

Zur Berechnung der Schrittweiten im Zielprogramm benötigen wir noch eine Definition, die den größten gemeinsamen Teiler auf $\mathbb{Q}$ verallgemeinert.

Definition 2.4.10 (Größter gemeinsamer Teiler auf $\mathbb{Q}$, $gcd_{\mathbb{Q}}$) Sei $\frac{a_1}{b_1}$, $\frac{a_2}{b_2} \in \mathbb{Q}$. Sei $\frac{a'_1}{b}$, $\frac{a'_2}{b}$ die Darstellung mit gemeinsamen Hauptnenner b. Dann berechnet sich der *größte gemeinsame Teiler* von $\frac{a_1}{b_1}$ und $\frac{a_2}{b_2}$ wie folgt:

$$gcd_{\mathbb{Q}}\left(\frac{a_1}{b_1}, \frac{a_2}{b_2}\right) = \frac{gcd(a'_1, a'_2)}{b}$$

Die Schrittweiten der Schleifen der Ziel-Programmteile sind Elemente aus $\mathbb{Q}$. Für das vollständige Zielprogramm ergibt sich als Schrittweite für Schleife j:

$$s_j = gcd_{\mathbb{Q}}(s_{j_1}, \ldots, s_{j_k}),\ s_j \in \mathbb{Q}$$

2.5 Grundlegende Begriffe zur Partitionierung

Kommen wir nun zu einigen grundsätzlichen Definitionen der Partitionierung. In dieser Arbeit soll entgegen gängiger Literatur[5] die Partitionierung paralleler Schleifensätze diskutiert werden. Somit ist im folgenden die Gruppierung von parallel auszuführenden Berechnungspunkten gemeint, wenn wir von Partitionierung sprechen.

[4] *Strukturparameter* definieren Konstanten, die in einem Programm durch einen Bezeichner deklariert werden. Sie werden dazu benutzt, Problemgrößen zu erfassen, wie z.B. in der Multiplikation zweier $n \times n$-Matrizen

[5] In der Literatur werden üblicherweise Verfahren unter dem Begriff *Tiling* zusammengefaßt, die auf rein sequentielle Programme angewandt werden. Da unser Anwendungsgebiet aber bei parallelen Programmen liegt, benutzen wir zur Abgrenzung den Begriff *Partitionierung.*

Definition 2.5.1 (Partitionierung) Wir bezeichnen ein Verfahren, das elementare Berechnungspunkte eines parallel perfekt verschachtelten Programms in Blöcke T_i gruppiert, als *Partitionierung*, wenn gilt:

1. Die geometrische Form aller Blöcke ist gleich.
2. $D^{\text{par}} \subseteq \bigcup_{i \in \mathbb{N}} T_i$
3. $\bigcap_{i \in \mathbb{N}} T_i = \emptyset$

Blöcke, die diese Eigenschaften erfüllen, werden als *Tiles* bezeichnet. Die Beschreibung der geometrischen Form eines Tiles, dessen lexikografisches Minimum im Nullpunkt liegt, durch ein Ungleichungssystem nennt man *Tile-Form*. Mit dem *Tile-Offset* $\vec{c}$ beschreiben wir die Translation der Tile-Form zum *Basis-Tile*, durch das die Lage aller übrigen Tiles eindeutig definiert ist.

Anmerkung 2.5.2 Die Restriktion für die Lage des lexikografischen Minimums einer Tile-Form hat lediglich normierenden Charakter. Es wäre auch ein anderes Kriterium denkbar, um das beschreibende Ungleichungssystem eindeutig festzulegen.

Die Partitionierung hat folgende Ziele:

1. Da bei der Allokation von einer Verteilung der Ausführungspunkte auf beliebig viele virtuelle Prozessoren ausgegangen wird, kann das transformierte Zielprogramm nicht auf einem Parallelrechner mit beschränkter Anzahl realer Prozessoren ausgeführt werden, falls die Anzahl an realen Prozessoren kleiner als die Anzahl an virtuellen Prozessoren ist. Aufgabe der Partitionierung ist es nun, die n Berechnungspunkte einer parallelen Schleife auf $m < n$ real vorhandene Prozessoren zu verteilen. Dadurch kann das partitionierte Zielprogramm auf einer beliebigen real existierenden Rechnerarchitektur ausgeführt werden. Im folgenden Abschnitt stellen wir zwei Techniken vor, die mit jeweils unterschiedlichen Ansätzen diese Verteilung durchführen.

2. Die Berechnungspunkte sollen möglichst optimal gruppiert werden, so daß der Kommunikationsaufwand minimiert wird. Im folgenden wird davon ausgegangen, daß Kommunikation nur durch Abhängigkeiten verursacht wird.

Die in Definition 2.5.1 eingeführten Begriffe sollen in einem Beispiel veranschaulicht werden.

Beispiel 2.5.3 Gegeben sei folgendes Ziel-Programm:

for $t := 0$ **to** 10 **do**
  parfor $p_1 := 0$ **to** 20 **do**
    parfor $p_2 := 0$ **to** $\min(15, 25 - p_1)$ **do**
      {Schleifenkörper}

Für den parallelen Indexraum D^{par}, der in Abbildung 2 dargestellt ist, ergeben sich die Ungleichungen:

$$\begin{array}{rcl} 0 \leq & p_1 & \leq 20 \\ 0 \leq & p_2 & \leq 15 \\ & p_2 & \leq 25 - p_1 \end{array}$$

Jedes der sechseckigen Polytope, die den Indexraum partitionieren, ist ein Tile. Die sechseckige Form der Tiles wurde völlig willkürlich gewählt. Die Tile-Form wird durch folgendes Ungleichungssystem beschrieben:

$$\begin{array}{rcl} 0 \leq & p_1 & < 4 \\ 0 \leq & p_2 & < 4 \\ p_1 - 2 \leq & p_2 & < p_1 + 2 \end{array}$$

Dieses Ungleichungssystem gibt die Grenzen des Tiles an, dessen lexikografisches Minimum im Nullpunkt liegt (s. Tile A der Abbildung). Durch die Wahl des Tile-Offsets $\vec{c} = \vec{0}$ entspricht Tile A dem Basis-Tile. Wegen Definition 2.5.1 muß der Schnitt aller Tiles gleich der leeren Menge sein, weshalb wir in diesem Beispiel für die Obergrenzen echte Ungleichungen benutzen.

Wie bereits oben erwähnt, unterscheidet man grundsätzlich zwei Verfahren zur Partitionierung des Indexraums:

LPGS: Bei dieser Technik werden alle Berechnungspunkte innerhalb eines Tiles parallel ausgeführt (**l**okal **p**arallel), die Tiles werden sequentiell

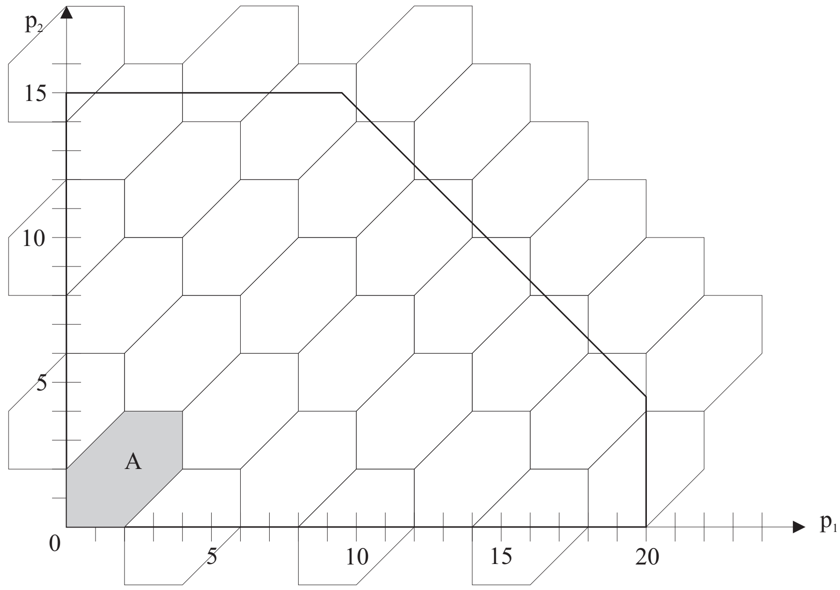

Abbildung 2: Mögliche Partitionierung des Indexraums aus Beispiel 2.5.3

aufgezählt (**g**lobal **s**equentiell). Die Tile-Größe entspricht der Anzahl an realen Prozessoren. Sei n die Anzahl an Berechnungspunkten, die von einer parallelen Schleife aufgezählt werden, m die Anzahl der real vorhandenen Prozessoren. Bei der Partitionierung gemäß des LPGS-Verfahrens wird der parallele Indexraum, der n Punkte enthält, in Tiles mit je m parallel auszuführenden Berechnungspunkten aufgeteilt. Die sich ergebenden $\lceil \frac{n}{m} \rceil$ Tiles werden sequentiell aufgezählt. Programmtechnisch wird durch dieses Verfahren eine parallele Schleife in zwei Schleifen transformiert, wobei davon eine Parallelschleife jeweils die Berechnungspunkte innerhalb eines Tiles abarbeitet und eine sequentielle Schleife die Tiles aufzählt. Die Reihenfolge der parallelen und sequentiellen Schleifen wird durch synchrone oder asynchrone Programmerzeugung festgelegt.

Beispiel 2.5.4 Gegeben sei folgende parallele Schleife:

$$\textbf{parfor } i := 0 \textbf{ to } 19 \textbf{ do}$$
$$\quad \ddots$$

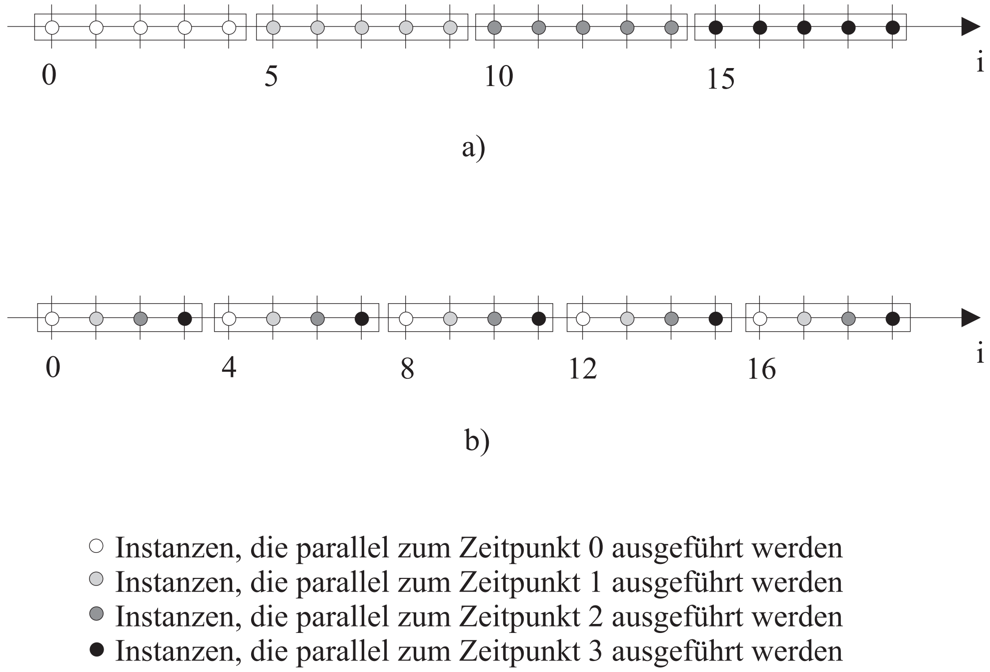

Abbildung 3: Partitionierung des Indexraums für Beispiel 2.5.4 und 2.5.5

Die Anzahl an real vorhandenen Prozessoren sei 5. Nach dem LPGS-Verfahren wird diese Schleife wie in Abbildung 3 a) partitioniert. Im synchronen Fall ergibt sich folgender Ziel-Code:

$$
\begin{array}{l}
\textbf{for } tp := 0 \textbf{ to } 3 \textbf{ do} \\
\quad \textbf{parfor } rp := 0 \textbf{ to } 4 \textbf{ do} \\
\quad\quad\quad i := 5tp + rp \qquad\qquad \{\mathrm{S}\} \\
\quad\quad\quad\quad \ddots
\end{array}
$$

Die durch Statement S gegebene Beziehung wird in Abschnitt 4.1.2 hergeleitet.

LSGP: Hier wird jedes Tile auf einen Prozessor abgebildet. Die Punkte innerhalb eines Tiles werden sequentiell (**l**okal **s**equentiell) auf einem

Prozessor ausgeführt, die Tiles werden parallel von den realen Prozessoren aufgezählt (**g**lobal **p**arallel). Somit entspricht die Anzahl an Tiles in einer Dimension der Anzahl an realen Prozessoren in dieser Dimension. Bei diesem Verfahren wird programmtechnisch jede parallele Schleife in zwei Schleifen transformiert, wobei davon eine Parallelschleife die Tiles aufzählt und eine sequentielle Schleife die Ausführung der Berechnungspunkte innerhalb eines Tiles übernimmt. Auch hier ist die Schleifenordnung von der Art der Programmerzeugung abhängig.

Beispiel 2.5.5 Die Partitionierung nach dem LSGP-Verfahren für die Schleife aus Beispiel 2.5.4 wird in Abbildung 3 b) dargestellt. Im synchronen Fall ergibt sich nachstehender Ziel-Code:

$$
\begin{array}{l}
\textbf{for } tp := 0 \textbf{ to } 3 \textbf{ do} \\
\quad \textbf{parfor } rp := 0 \textbf{ to } 4 \textbf{ do} \\
\qquad\quad i := 4rp + tp \qquad\qquad \{\mathrm{S}\} \\
\qquad\qquad\quad \ddots
\end{array}
$$

Da LooPo auch while-Schleifen[6] verarbeiten kann [9], ist es von besonderem Interesse, daß auch dieses Iterationskonstrukt partitioniert werden kann. Diese Möglichkeit bietet aber ausschließlich das LPGS-Verfahren, da wegen der zur Compilezeit unbekannten Obergrenze die Anzahl an benötigten virtuellen Prozessoren nicht ermittelt werden kann. Diese Größe wird jedoch bei der LSGP-Partitionierung benötigt, um die Größe eines Tiles zu bestimmen, die sich folgendermaßen ergibt:

$$\left\lceil \frac{\text{Anzahl virtuelle Prozessoren}}{\text{Anzahl reale Prozessoren}} \right\rceil$$

Das LPGS-Verfahren dagegen partitioniert den parallelen Indexraum immer in Tiles mit n Berechnungspunkten, die von den n vorhandenen Prozessoren parallel ausgeführt werden. Diese Tiles werden im Falle einer while-Schleife solange aufgezählt, bis die Bedingung nicht mehr erfüllt ist. Da die

[6]While-Schleifen werden in Loopo zunächst in for-Schleifen mit einem variablen Parameter als Obergrenze transformiert. Diese for-Schleife wird dann durch Schedule und Allokation in eine zeitliche sowie räumliche Dimension zerlegt, wobei die Ausdehnung in diesen Dimensionen zur Compilezeit nicht bekannt ist.

Tile-Größe durch die Anzahl an real vorhandenen Prozessoren festgelegt ist, stellt die unbekannte Obergrenze einer while-Schleife bei diesem Verfahren kein Problem dar.

Das LSGP-Verfahren wurde in der Literatur weitreichend diskutiert [3], [4], [20], [24], jedoch nur für sequentielle Programme. Verfahren für LPGS finden sich dagegen kaum. Gegenstand dieser Arbeit ist die Vorstellung eines LPGS-Verfahrens mit Optimierung des Kommunikationsaufwands in Kapitel 4, sowie die Erweiterung sequentieller LSGP-Verfahren auf parallele Programme in Kapitel 5. Neben der theoretischen Behandlung dieser Verfahren wurden außerdem Algorithmen für LPGS im LooPo-Projekt implementiert. Die Randbedingungen, die für diese Implementierung gelten und sich auch auf den theoretischen Entwurf auswirken, werden im nächsten Kapitel erläutert.

3 Eingliederung der Partitionierung in das LooPo-Projekt

In diesem Kapitel soll kurz auf die Überlegungen eingegangen werden, die nötig sind, um die Implementierung der Partitionierung in LooPo einzugliedern.

3.1 Module von LooPo

Der Prozeß der automatischen Parallelisierung durch LooPo gliedert sich in folgende Kernmodule, wie in Abbildung 4 dargestellt:

- Durch den *Parser* wird das Quell-Programm eingelesen und auf seine syntaktische Korrektheit hin überprüft [11]. Als Ergebnis erhält man einen Parsebaum.

- Die *Dependence Analysis* untersucht den Parsebaum nach Abhängigkeiten zwischen einzelnen Statements. Außerdem generiert sie die Ungleichungssysteme zur Beschreibung des Indexraums jedes Statements [13].

- *Scheduler* und *Allokator* berechnen anhand des Indexraums und der Abhängigkeiten die zeitliche und räumliche Ausführung der Statements und beschreiben diese durch verallgemeinerte affine Abbildungen [16], [23].

- *TargetLoops* transformiert die Quell-Indexräume und -Statements anhand von Schedule und Allokation in Ziel-Indexräume und -Statements und generiert daraus ein Ziel-Programm, das aus Zeit- und Raumschleifen besteht, dargestellt durch einen Ziel-Parsebaum [22].

- *Target Code* erzeugt aus dem Ziel-Parsebaum ein Ziel-Programm in einer wählbaren parallelen Programmiersprache und fügt nach einer Untersuchung der Kommunikation die notwendigen Anweisungen zum Datenaustausch ein [7].

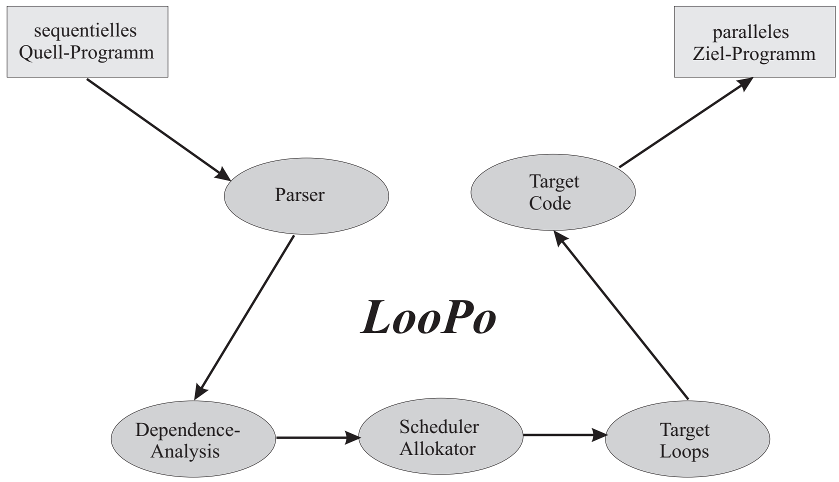

Abbildung 4: Die Struktur von LooPo

3.2 Implementierung im Modul Target Loops

Da wir bei der Partitionierung bereits von einem parallelen Programm ausgehen, muß ihre Implementierung nach der Erstellung des Ziel-Indexraums eingefügt werden. Naheliegend wäre zunächst, die Partitionierung als eigenes Modul hinter Target Loops einzureihen. Target Loops generiert durch die Vereinigung der transformierten Programmteile (siehe Kapitel 2.4) den Ziel-Indexraum, der geometrisch ein quasi-konvexes Polytop darstellt. Da aber quasi-konvexe Polytope nicht durch for- bzw. parfor-Schleifen aufgezählt werden können, wird eine konvexe Menge aufgezählt, die durch das Minimum aller unteren bzw. das Maximum aller oberen Schleifengrenzen der Programmteile definiert wird. Das beschreibende Ungleichungssystem dieser Menge kann aufgrund der Minima bzw. Maxima nicht in der Matrixschreibweise, die wir in Anmerkung 2.2.6 eingeführt haben, dargestellt werden. Wie wir aber in Kapitel 4.1 sehen werden, benötigen wir zur Aufzählung der Tiles die Beschreibung des Indexraums in dieser Darstellung. Diese Einschränkung wird von dem Verfahren, das zur Berechnung der neuen Schleifengrenzen eingesetzt wird, gefordert.

Aus diesem Grund wurde die Entscheidung getroffen, die Partitionierung in das Modul Target Loops einzubinden. Sie wird dort an der Stelle angesiedelt, an der die transformierten Programmteile bereits existieren. Nach ihrer Definition besitzen Programmteile einen Indexraum, der durch ein Polytop darstellbar ist und somit auch in Matrix-Schreibweise. Jeder Programmteil erfüllt also für sich die oben genannte Einschränkung.

3.3 Restriktionen für die Tile-Form

Die Entscheidung in Abschnitt 3.2 hatte einige Konsequenzen für die Wahl der Tile-Form.

Wie erwähnt, wird die Partitionierung je Programmteil durchgeführt. Damit stellt sich die Frage, ob für jeden Programmteil dieselbe Tile-Form gewählt wird. Da die eigentliche Absicht in der Partitionierung des vollständigen Programms lag, entscheiden wir uns für eine gemeinsame Tile-Form für alle Programmteile.

Als weiterer Punkt ist die Lage der Tiles im Indexraum zu klären. Mit dem gleichen Argument wie oben wählen wir denselben Tile-Offset zur Translation der Tiles bezüglich der Tile-Form für alle Programmteile. Als Wert für den Tile-Offset entscheiden wir uns für den Nullvektor, d.h. das Basis-Tile ist kongruent mit der Tile-Form.

Anmerkung 3.3.1 Keine dieser Restriktionen ist von theoretischer Seite her zwingend. D.h. es wäre durchaus möglich, jeden Programmteil mit einer eigenen Tile-Form und eigenem Tile-Offset zu partitionieren, sofern bei der Tile-Größe die Anzahl an real vorhandenen Prozessoren berücksichtigt wird. Die Berechnung des Offsets erzwingt jedoch in der Praxis einen Vektor, der keine Strukturparameter enthalten darf. Ursache dafür ist, daß sich mit dem später vorzustellenden Verfahren zur Aufzählung der Tiles nicht lineare Ungleichungssysteme ergeben würden, für deren Auflösung keine allgemein anwendbaren algorithmischen Verfahren zur Verfügung stehen. Deshalb wählen wir stets den Tile-Offset $\vec{0}$.

Folgendes Beispiel soll diese Punkte veranschaulichen:

Beispiel 3.3.2 Gegeben sei ein Zielprogramm, das in folgende zwei transformierte Programmteile aufgespalten ist:

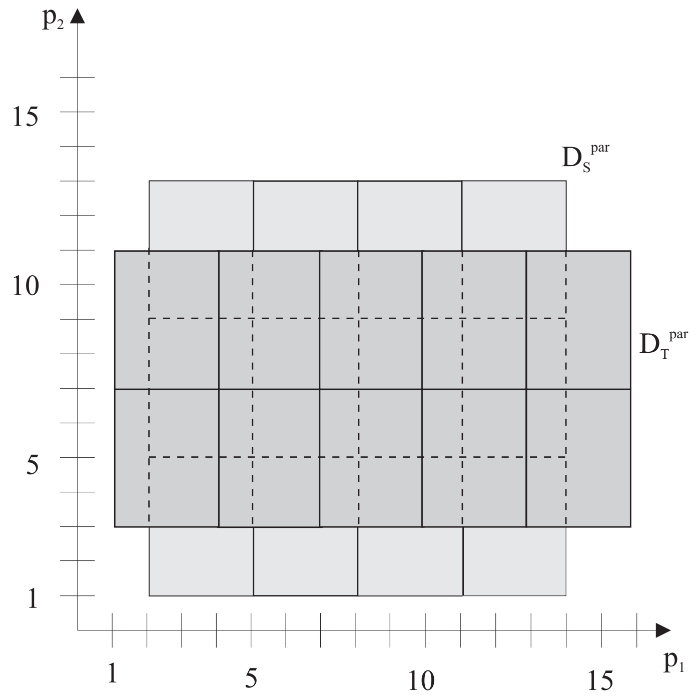

Abbildung 5: Mögliche Partitionierung mit derselben Tile-Form und unterschiedlichem Tile-Offset

Programmteil 1:

for $t := 0$ **to** 10 **do**
  parfor $p_1 := 2$ **to** 14 **do**
    parfor $p_2 := 1$ **to** 13 **do**
      {Statement S}

Programmteil 2:

for $t := 2$ **to** 9 **do**
  parfor $p_1 := 1$ **to** 16 **do**
    parfor $p_2 := 3$ **to** 11 **do**
      {Statement T}

Die zugehörigen Indexräume beider Programmteile mit zwei möglichen Partitionierungen sind in den Abbildungen 5 und 6 dargestellt.

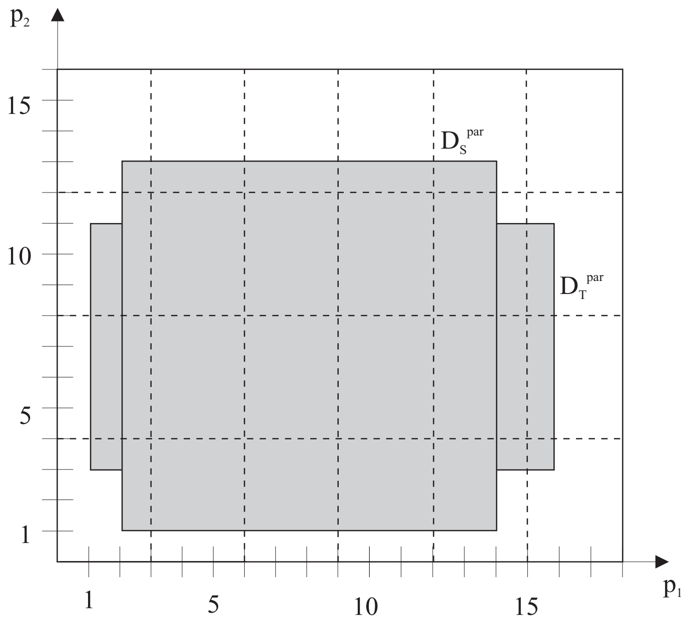

Abbildung 6: Mögliche Partitionierung mit derselben Tile-Form und gemeinsamem Tile-Offset $\vec{0}$

3.4 Synchrone Programmerzeugung

Weiter ergibt sich ein Problem für die Partitionierung, falls Target Loops ein asynchrones Programm erzeugt. Wie im Abschnitt 2.4 dargelegt wird, können innere Schleifen von äußeren Schleifen abhängen, d.h. die Schleifengrenzen von inneren Schleifen können Schleifenindizes äußerer Schleifen enthalten. Somit besteht die Möglichkeit, daß in einem asynchronen Programm in sequentiellen Schleifengrenzen Schleifenindizes äußerer paralleler Schleifen auftreten, z.B. :

parfor $p_1 := 0$ **to** 10 **do**
  for $t_1 := p_1$ **to** $p_1 + 5$ **do**
    {Schleifenkörper}

Wird der Indexraum — in unserem Beispiel von der parallelen Schleife mit Index p_1 erzeugt — partitioniert, ergeben sich aus der p_1-Schleife zwei neue Schleifen mit jeweils einem neuen Index:

parfor $rp := \ldots$ **to** $\ldots$ **do**
  for $tp := \ldots$ **to** $\ldots$ **do**

In Abschnitt 4.1.2 werden wir sehen, daß außerdem die von Target Loops generierten sequentiellen Schleifen, die aus dem Schedule resultieren, immer die äußersten sequentiellen Schleifen des partitionierten Zielprogramms bleiben müssen. Ansonsten würden eventuell Abhängigkeiten verletzt. In unserem Beispiel ergibt sich also folgender Schleifensatz:

parfor $rp := \ldots$ **to** $\ldots$ **do**
  for $t_1 := p_1$ **to** $p_1 + 5$ **do**
    for $tp := \ldots$ **to** $\ldots$ **do**

Aus dem sich ergebenden Schleifensatz wird das Problem sofort ersichtlich. Die sequentielle t_1-Schleife, die durch die Partitionierung unverändert bleibt, benutzt in ihren Grenzen weiterhin den Index p_1, der aber nicht mehr deklariert ist. Wie sich später herausstellen wird, gibt es zwar eine Beziehung, durch die sich aus den Indizes rp und tp der ursprüngliche Index p_1 ergibt, da aber beim Zugriff auf p_1 in den Grenzen von t_1 der Schleifenindex tp noch nicht bekannt ist, kann die Abbildung von rp und tp auf p_1 nicht vor der Schleife t_1 erfolgen.

Eine Lösung dieses Problems erreicht man dadurch, daß das Modul Target Loops intern bis zum Auruf der Partitionierung immer synchronen Ziel-Code erzeugen muß. Dadurch ist es nicht mehr möglich, daß sequentielle Schleifen in ihren Schleifengrenzen Indizes von äußeren parallelen Schleifen benutzen.

Andererseits führt es zu keinen Problemen, wenn parallele Schleifen von äußeren sequentiellen Schleifen abhängen, was im synchronen Fall möglich ist. Da von Target Loops generierte sequentielle Schleifen durch die Partitionierung unverändert bleiben und in einem synchronen Programm auch immer die äußersten Schleifen sein müssen, ist garantiert, daß alle sequentiellen Indizes bereits deklariert sind, wenn sie in innereren Schleifengrenzen auftauchen.

Die Partitionierung formt die synchronen Programmteile dann wieder in asynchrone um, die anschließend zu einem asynchronen Zielprogramm vereinigt werden. Durch die Tatsache, daß parallele Schleifen nie Abhängigkeiten tragen können, ist dabei sichergestellt, daß keine Abhängigkeiten verletzt werden, wenn die parallelen Schleifen über die sequentiellen Schleifen nach außen gezogen werden, um wieder ein asynchrones Zielprogramm zu erhalten.

Da nun grundlegende Begriffe zur Partitionierung und ihre Rolle im gesamten LooPo-Projekt geklärt sind, werden wir uns in den nachfolgenden Kapiteln mit den beiden Kategorien der Partitionierung, LPGS und LSGP, beschäftigen.

4 LPGS-Partitionierung

Zunächst werden wir als Schwerpunkt dieser Arbeit das LPGS-Verfahren zur Partitionierung von parallelen Programmen diskutieren. Wie bereits in Abschnitt 2.5 erläutert, partitioniert dieses Verfahren den parallelen Indexraum in Tiles, deren Berechnungspunkte parallel auf den real vorhandenen Prozessoren ausgeführt werden, die Tiles werden sequentiell aufgezählt.

4.1 Aufzählung von Tile-Formen

Dieser Abschnitt stellt ein Verfahren zum Aufzählen von Polytopen mit for-Schleifen vor [1]. Da dieses Verfahren auf rein sequentiellen Programmen arbeitet, wird es in Abschnitt 4.1.2 so erweitert, daß es Tiles derart aufzählen kann, daß ein nach dem LPGS-Verfahren partitioniertes Programm entsteht. Wir gehen hierzu zunächst von einer gegebenen Tile-Form[7] aus. Anschließend betrachten wir die Probleme der Aufzählung bei Schrittweiten $\neq 1$ und stellen Lösungsansätze vor.

4.1.1 Das Verfahren von Ancourt/Irigoin

Prinzip des hier vorzustellenden Verfahrens [1] ist es, zunächst ein Ungleichungssystem zu konstruieren, das implizit die Grenzen der Schleifen angibt, die zum einen die Tiles, zum anderen die Tile-Inhalte aufzählen. Die Berechnung der Grenzen erfolgt mittels Fourier-Elimination.

Da die Form eines Tiles in dieser Arbeit immer einem Parallelotop und damit einem Polyeder entspricht, ist es möglich, die Tile-Form in folgender Matrix-Darstellung anzugeben:

$$S\vec{\imath} \leq \vec{s}$$

Dabei sei $\vec{\imath}$ der Koordinatenvektor des Indexraums D, S die Koeffizientenmatrix des Ungleichungssystems, das die Tile-Form beschreibt und $\vec{s}$ der konstante Anteil. Gleiches gilt für die Darstellung des Indexraums D:

$$B\vec{\imath} \leq \vec{b} \qquad (1)$$

[7]Bei LooPo hat der Benutzer neben der automatischen Berechnung einer Tile-Form auch die Möglichkeit, selbst eine bestimmte Tile-Form anzugeben, indem er n linear unabhängige Vektoren angibt, wobei n die Dimension der Allokation sei. Diese Vektoren definieren jeweils die Ecken der Tile-Form.

Formal bedeutet die Aufzählung von Tiles wiederum die Aufzählung einer Menge, dem *Tile-Indexraum* T. Wählt man für jedes Tile einen Bezugspunkt $\vec{\imath}_0$ — z.B. sein lexikografisches Minimum — so erlaubt es folgende Abbildung, eine Beziehung zwischen T und D herzustellen:

$$L : T \rightarrow D,\ L\vec{t} + \vec{c} = \vec{\imath}_0 \tag{2}$$

Diese Abbildung weist jedem Tile mit Koordinaten $\vec{t}$ in T einen Indexpunkt $\vec{\imath}_0$ in D zu, wobei $\vec{c}$ der Tile-Offset ist. Wir bezeichnen L als *Gitter*. Implizit definiert das Gitter also die Reihenfolge der Aufzählung. Die Aufzählung muß unter Berücksichtigung der Tile-Form gemäß Definition 2.5.1 so erfolgen, daß die Tiles zum einen lückenlos aufgezählt werden, sich andererseits aber nicht überlappen. Die Abbildung L ist jedoch nicht eindeutig bestimmt, da jeder Punkt eines Tiles als Bezugspunkt gewählt werden kann und zudem die Reihenfolge, in der die Tiles aufgezählt werden, nicht deterministisch ist. Einzige Einschränkung ist, daß Abhängigkeiten zwischen Tiles berücksichtigt werden müssen.

Anmerkung 4.1.1 In einem Programm mit n Schleifen erhält man für $\vec{c} = 0$ ein Gitter mit diesen Eigenschaften beispielsweise durch die Berechnung von n linear unabhängigen Vektoren, deren Linearkombinationen die Bezugspunkte aller Tiles ergeben. Dazu bestimmen wir alle Bezugspunkte, die im Rahmen des Ungleichungssystems, das die Tile-Form repräsentiert, liegen, und wählen unter den beschreibenden Vektoren n linear unabhängige aus. Diese definieren die Spalten des Gitters.

Das Gitter ermöglicht nun die Darstellung eines beliebigen Tiles $\vec{t}$ im Indexraum D. Dazu benötigen wir zunächst folgendes Ungleichungssystem:

$$S(\vec{\imath} - \vec{\imath}_0) \leq \vec{s},$$

das die Indexpunkte eines Tiles mit Bezugspunkt $\vec{\imath}_0$ beschreibt. Ersetzt man $\vec{\imath}_0$ durch die Gleichung in (2), so ergibt sich:

$$S(\vec{\imath} - L\vec{t} - \vec{c}) \leq \vec{s} \tag{3}$$

Durch dieses Ungleichungssystem sind implizit die Grenzen von $\vec{t}$ zum Aufzählen der Tiles, sowie die Grenzen von $\vec{\imath}$ — eventuell in Abhängigkeit von $\vec{t}$ — zum Aufzählen der Tile-Inhalte gegeben. Um zu erreichen, daß nur Tiles aufgezählt werden, deren Schnitt mit dem Indexraum nicht leer ist, ergänzen wir Ungleichung (3) durch Ungleichung (1). Mit diesem Ungleichungssystem ist es nun möglich, den Tile-Indexraum T formal zu definieren:

$$T := \{\vec{t} : (\exists i : i \in D : B\vec{\imath} \leq \vec{b} \wedge S(\vec{\imath} - L\vec{t} - \vec{c}) \leq \vec{s})\}$$

Zur Berechnung der Schleifengrenzen muß ein Polyeder über $D \times T$ auf den Unterraum T projeziert werden. Das erreicht man durch Anwenden der Fourier-Elimination.

Jede Komponente von $\vec{t}$ entspricht dann einer neuen sequentiellen Schleife, die nicht-leere Tiles aufzählt. Durch die Bestimmung der Grenzen für $\vec{\imath}$ erhält man die Schleifen, die die Tile-Inhalte aufzählen.

Das folgende Beispiel soll das Vorgehen dieses Verfahrens deutlich machen:

Beispiel 4.1.2 Gegeben sei folgender Schleifensatz:

for $i := 0$ **to** 15 **do**
  for $j := 0$ **to** $15 - i$ **do**
    {Schleifenkörper}

Die Tile-Form sei beschrieben durch folgende Ungleichungen:

$$0 \leq j < 3, \; j \leq i < j + 3$$

In Matrix-Darstellung ergibt sich:

$$\begin{pmatrix} 0 & -1 \\ -1 & 1 \\ 0 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 2 \\ 2 \end{pmatrix}$$

Als Gitter wählen wir folgende Abbildung:

$$L = \begin{pmatrix} 0 & 3 \\ 3 & 3 \end{pmatrix}$$

L ist in diesem Beispiel unter der Annahme, daß keine Abhängigkeiten berücksichtigt werden müssen, willkürlich gewählt. Da wir uns in Kapitel 3.3 auf

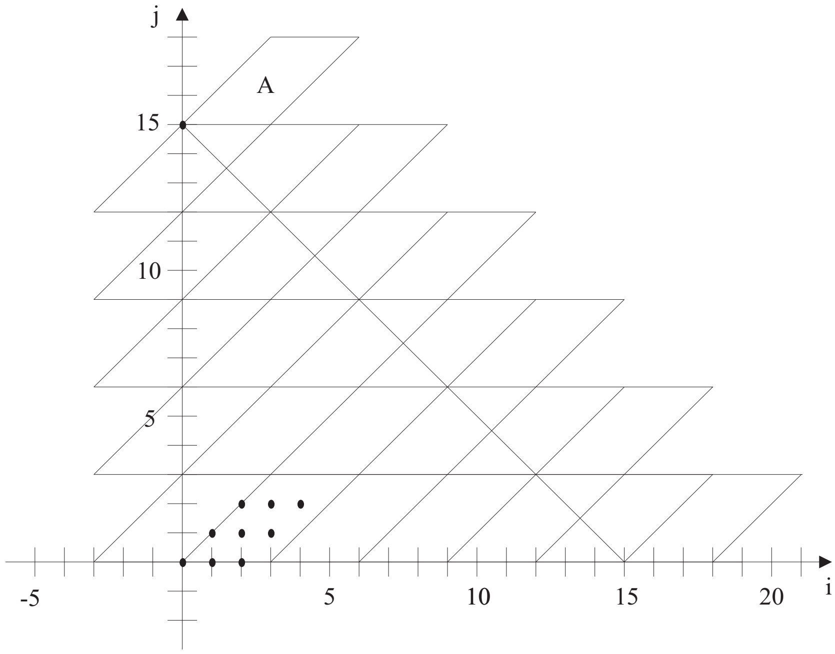

Abbildung 7: Partitionierung des Indexraums für Beispiel 4.1.2

den Tile-Offset $\vec{0}$ festgelegt haben, wird auch in diesem Beispiel von einem Offset $\vec{c} = \vec{0}$ ausgegangen.
Der Indexraum ist definiert durch:

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 15 \end{pmatrix}$$

Schließlich ergibt sich die Menge der Indexpunkte, die zu einem bestimmten Tile mit Tile-Koordinaten (t_1, t_2) gehören, folgendermaßen:

$$\begin{pmatrix} 0 & -1 \\ -1 & 1 \\ 0 & 1 \\ 1 & -1 \end{pmatrix} \left(\begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} 0 & 3 \\ 3 & 3 \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \right) \leq \begin{pmatrix} 0 \\ 0 \\ 2 \\ 2 \end{pmatrix}$$

Schneidet man nun jedes Tile mit Tile-Koordinaten (t_1, t_2) mit dem Indexraum, erhält man alle Tiles, die mindestens einen Punkt enthalten. Man beachte, daß beispielsweise Tile A aus Abbildung 7 ein nicht-leeres Tile ist.

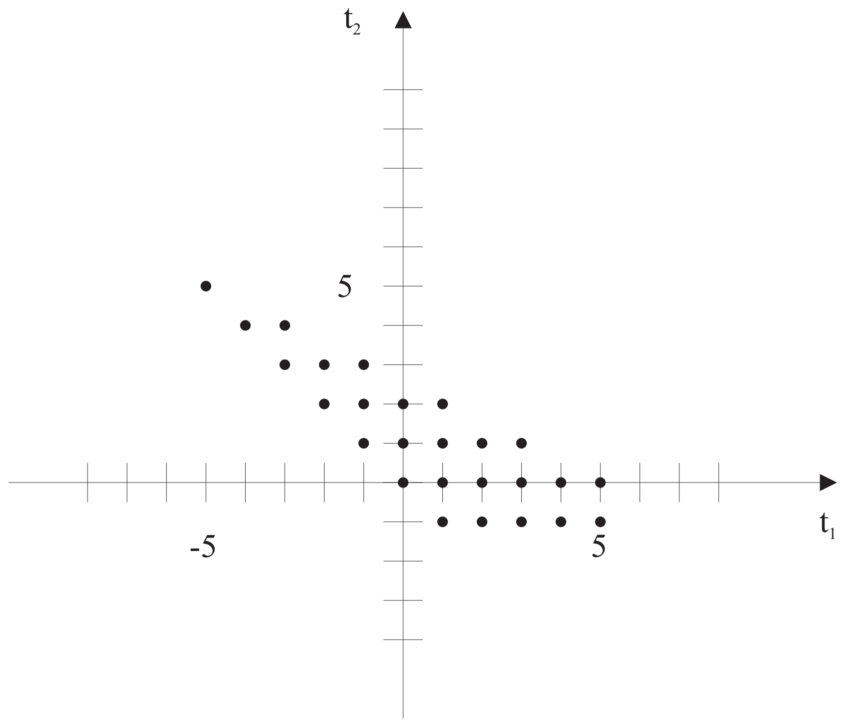

Abbildung 8: Darstellung des Tile-Indexraums für Beispiel 4.1.2

Es enthält genau einen Punkt des Indexraums, und zwar den Punkt $(0, 15)$. Ein Tile, das vollständig im Indexraum liegt, zählt 9 Indexpunkte auf, wie z.B. das Basis-Tile.

Wendet man die Fourier-Elimination auf das Ungleichungssystem an, das sich durch den Schnitt des Indexraums mit allen Tiles ergibt, erhält man folgende neue Schleifengrenzen:

for $t_1 := -5$ **to** 5 **do**
  for $t_2 := \max(\lceil -t_1 - \frac{2}{3} \rceil, -1)$ **to** $\min(5, 5 - t_1, \lfloor \frac{5}{2} - \frac{1}{2}t_1 \rfloor)$ **do**
    for $i := \max(0, -3t_1, 3t_2)$ **to** $\min(15, 15 - 3t_1 - 3t_2, 3t_2 + 4)$ **do**
      for $j := \max(0, 3t_1 + 3t_2, 3t_1 + i - 2)$ **to**
        $\min(15 - i, 3t_1 + i, 3t_1 + 3t_2 + 2)$ **do**

Der Tile-Indexraum T, dessen Punkte jeweils ein Tile repräsentieren, wird in Abbildung 8 dargestellt. So stellt beispielsweise der Punkt mit den Tile-Koordinaten $(5, 0)$ das Tile A in Abbildung 7 dar.

4.1.2 Erweiterung des Verfahrens

In diesem Abschnitt wird das soeben vorgestellte Verfahren um einige Punkte erweitert, so daß es auf die Partitionierung paralleler Programme anwendbar ist.

Ein grundlegender Unterschied ist, daß im Verfahren von Irigoin/Ancourt nur sequentielle Programme partitioniert werden, während wir im folgenden ausschließlich die Partitionierung von parallelen Schleifen betrachten. Wenden wir das Verfahren wie beschrieben auf diese Schleifen an, ist es naheliegend, die sich ergebenden neuen $\vec{t}$-Schleifen sequentiell auszuführen, weil sie die Tiles aufzählen (global sequentiell). Die transformierten $\vec{\imath}$-Schleifen bleiben parallele Schleifen zum Aufzählen der Tile-Inhalte (lokal parallel). Wir gehen im Rahmen dieser Arbeit davon aus, daß mit dem Wert eines parallelen Schleifenindex i_j der reale Prozessor in Dimension j angesprochen wird, d.h. daß beispielsweise die 0-te Iteration auf dem realen Prozessor 0 ausgeführt wird. Betrachtet man jedoch die Grenzen eines transformierten Schleifenindex i, so sieht man, daß dieser, gruppiert in Tiles, wieder den ursprünglichen Indexraum und damit die virtuellen Prozessoren aufzählt. Um zu erreichen, daß bei P real vorhandenen Prozessoren von der Parallelschleife ausschließlich Werte zwischen 0 und $P-1$ angenommen werden, ergänzen wir das Ungleichungssystem aus 4.1.1 zunächst um einen neuen Schleifenindex $\vec{rp}$ mit den folgenden Ungleichungen ($\vec{P}$ sei im folgenden der Vektor, dessen Komponenten die Anzahl an realen Prozessoren je Dimension angeben.):

$$\begin{aligned} \vec{rp} &\geq \vec{0} \\ \vec{rp} &\leq \vec{P} - \vec{1} \end{aligned}$$

Die Komponenten von $\vec{rp}$ repräsentieren die Indizes der parallelen Schleifen zur Aufzählung der realen Prozessoren. Wie bereits erwähnt, gehen wir bezüglich der Rechnerarchitektur von einem Prozessorarray aus. Wir setzen weiter keine konkrete Vernetzung voraus, nehmen aber an, daß der Datenaustausch zwischen Prozessoren entweder von der Hardware des Rechners gesteuert wird, oder daß das Zielprogramm in Abhängigkeit von einer bestimmten Vernetzung durch ein nachgeordnetes Verfahren um die notwendigen Kommunikationsanweisungen erweitert wird, wie z.B. in [7].

In einem nächsten Schritt benötigen wir eine Beziehung zwischen realen und virtuellen Prozessoren, oder anders ausgedrückt, eine Abbildung zwischen den Indizes der parallelen $\vec{rp}$-Schleifen und den Indizes der Schleifen

$\vec{\imath}$, die die virtuellen Prozessoren aufzählen. Diese Beziehung ergibt sich folgendermaßen: Wie wir im letzten Abschnitt in Gleichung (2) bereits gesehen haben, weist die Abbildung $L\vec{t} + \vec{c} = \vec{\imath}_0$ jedem Tile mit Koordinaten $\vec{t}$ einen Bezugspunkt $\vec{\imath}_0$ im Indexraum zu. Als Tile-Offset wird in dieser Arbeit immer der Nullvektor gewählt, wie bereits in Abschnitt 3.3 vereinbart, als Bezugspunkt $\vec{\imath}_0$ wählen wir o.B.d.A. das lexikografische Minimum eines Tiles. Somit berechnet man durch den ersten Teil dieser Gleichung, $L\vec{t}$, das lexikografische Minimum eines Tiles. Mit der Addition von $\vec{rp}$ gelangt man zu allen übrigen Punkten, die in diesem Tile liegen. Also erhalten wir als Beziehung zwischen $\vec{rp}$ und $\vec{\imath}$:

$$L\vec{t} + \vec{rp} = \vec{\imath}$$

und integrieren diese in das Ungleichungssystem des Verfahrens durch:

$$L\vec{t} + \vec{rp} \leq \vec{\imath} \tag{4}$$
$$L\vec{t} + \vec{rp} \geq \vec{\imath} \tag{5}$$

Um im partitionierten Zielprogramm die Beziehung zwischen $\vec{\imath}$ und $\vec{rp}$ herzustellen, müßten nun in allen Array-Zugriffen des Schleifenkörpers alle Vorkommen von i_j durch $Lt_j + rp_j$ ersetzt werden. Da es aber in Bezug auf die Laufzeit des Programms effizienter ist, die Berechnung dieser Abbildung nur einmal auszuführen, wird am Anfang des Schleifenkörpers für jeden Index i_j die Zuweisung $i_j = Lt_j + rp_j$ eingefügt, die Array-Zugriffe bleiben damit unverändert.

Zusammengefaßt ergibt sich als erweitertes Ungleichungssystem zum Aufzählen von Tiles in parallelen Programmen ($\vec{P}$ sei hierbei der Vektor, dessen Komponenten die Anzahl an realen Prozessoren je Dimension angeben.):

$$\begin{array}{rcl} B\vec{\imath} & \leq & \vec{b} \\ S(\vec{\imath} - L\vec{t}) & \leq & \vec{s} \\ \vec{rp} & \geq & \vec{0} \\ \vec{rp} & \leq & \vec{P} - \vec{1} \\ L\vec{t} + \vec{rp} & \geq & \vec{\imath} \\ L\vec{t} + \vec{rp} & \leq & \vec{\imath} \end{array} \tag{6}$$

Die Reihenfolge der Aufzählung der Tiles ist frei wählbar, da die Tiles im parallelen Indexraum keine Abhängigkeiten besitzen. Wir bestimmen das Gitter gemäß Anmerkung 4.1.1.

Anmerkung 4.1.3 Im Falle einer rechteckigen Tile-Form ist dieses Ungleichungssystem überbestimmt, so daß die Ungleichung $S(\vec{\imath}-L\vec{t})\leq\vec{s}$ redundant wird.
Beweis: $S, \vec{s}$ und L haben in diesem Fall immer die folgende Form:

$$S = \begin{pmatrix} 1 & 0 & & \\ & \ddots & & \\ & & 0 & 1 \\ -1 & 0 & & \\ & \ddots & & \\ & & 0 & -1 \end{pmatrix}, \ \vec{s} = \begin{pmatrix} P_1 - 1 \\ \vdots \\ P_m - 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ L = \begin{pmatrix} P_1 & 0 & \dots & 0 \\ & & \ddots & \\ 0 & \dots & 0 & P_m \end{pmatrix}$$

Somit ergibt sich mit $S(\vec{\imath}-L\vec{t})\leq\vec{s}$ für jede Komponente j von $\vec{\imath}$:

$$i_j - P_j t_j \leq P_j - 1 \quad \Rightarrow \quad i_j \leq P_j t_j + (P_j - 1) \tag{7}$$
$$-i_j + P_j t_j \geq 0 \quad \Rightarrow \quad i_j \geq P_j t_j \tag{8}$$

Mit $L\vec{t}+r\vec{p}=\vec{\imath}$ erhalten wir:

$$\begin{aligned} P_j t_j + rp_j &\geq i_j \quad \text{und} \\ P_j t_j + rp_j &\leq i_j \end{aligned}$$

und in Verbindung mit $rp_j \geq 0$ und $rp_j \leq P_j - 1$ die Ungleichungen (7) und (8).

Wie bereits erwähnt, können die Tiles im parallelen Indexraum in beliebiger Reihenfolge aufgezählt werden, da hier keine Abhängigkeiten berücksichtigt werden müssen. Wohl aber ist es notwendig, daß bei Ausführung der parallelen Iterationen zum Zeitpunkt t alle Daten der Iterationen zum Zeitpunkt $t-1$ bereits vorhanden sind. Die vom Scheduler resultierenden sequentiellen Schleifen bleiben von der Partitionierung unberührt, da nur der parallele Indexraum partitioniert wird. Beim Wiedereinfügen von sequentiellen Schleifen nach der Partitionierung muß beachtet werden, daß sie als äußerste der

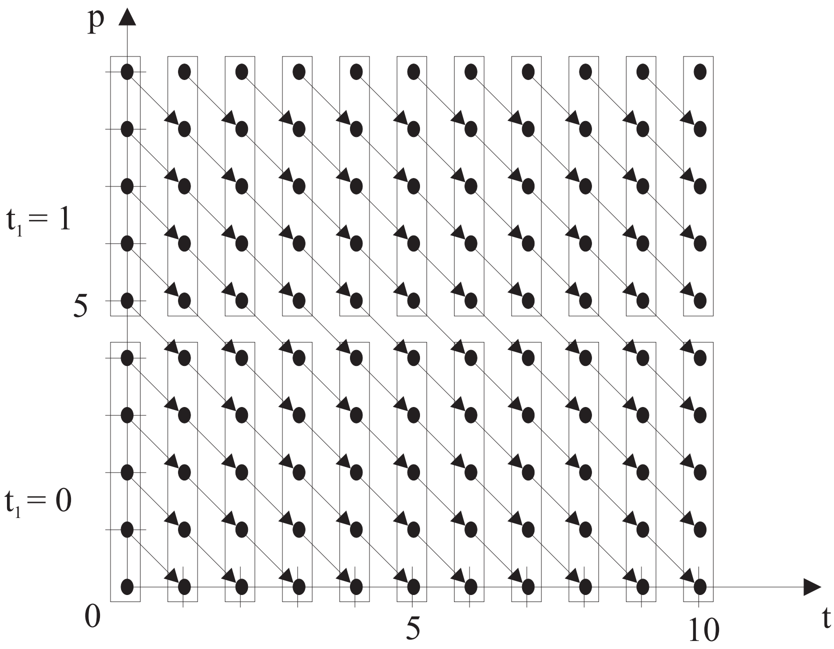

Abbildung 9: Partitionierung unter Berücksichtigung sequentieller Schleifen

sequentiellen Schleifen in den neuen sich ergebenden Schleifensatz eingeordnet werden. Dies muß gewährleistet sein, um keine Abhängigkeiten im Programm zu verletzen. Wird z.B. in Abbildung 9 die sich ergebende t_1-Schleife zum Aufzählen der Tiles die äußerste sequentielle Schleife des partitionierten Programms, ergibt sich folgendes Programmsegment:

for $t_1 := 0$ **to** 1 **do**
    for $t := 0$ **to** 10 **do**
        parfor $rp_1 := 0$ **to** 4 **do**

Dabei wird z.B. Tile $t_1 = 0$ für $t = 1$ vor Tile $t_1 = 1$ für $t = 0$ ausgeführt. Somit sind die Daten von Iteration $(0, 5)$ noch nicht bekannt, wenn sie für die Berechnung von Iteration $(1, 4)$ benötigt werden (siehe Abbildung 9). Fügt man die t-Schleife als äußerste der sequentiellen Schleifen ein,

for $t := 0$ **to** 10 **do**
  for $t_1 := 0$ **to** 1 **do**
    parfor $rp_1 := 0$ **to** 4 **do**

werden alle parallelen Iterationen zum Zeitpunkt t ausgeführt, bevor die parallelen Iterationen zum Zeitpunkt $t+1$ ausgeführt werden. Dadurch ist gewährleistet, daß alle Daten, die zum Zeitpunkt $t+1$ benötigt werden, bereits zum Zeitpunkt t berechnet wurden und somit keine Abhängigkeiten verletzt werden.

Hängen parallele Scheifen von Schleifenindizes äußerer sequentieller Schleifen ab, wird der sequentielle Schleifenindex während der Partitionierung zu einem Parameter, um den parallelen Indexraum begrenzen zu können. Nach der Partitionierung werden die sequentiellen Schleifen unverändert wieder dazugefügt. Damit bewegen sich die partitionierten parallelen Schleifen wieder korrekt in den von der sequentiellen Schleife abhängigen Grenzen.

Folgendes Beispiel soll das erweiterte Verfahren zum Aufzählen von Tiles veranschaulichen:

Beispiel 4.1.4 Gegeben sei folgendes synchrones Programm:

for $t := 0$ **to** 7 **do**
  parfor $i_1 := 3$ **to** $t + 10$ **do**
    parfor $i_2 := 0$ **to** 15 **do**
      {Schleifenkörper}

In der ersten Dimension seien 4 Prozessoren vorhanden, in der zweiten Dimension 5 Prozessoren. Die unter Berücksichtigung der vorhandenen Prozessoren willkürlich gewählte Tile-Form ist wie folgt beschrieben:

$$0 \le i_1 \le 3$$
$$0 \le i_2 \le 4$$

In Abbildung 10 ist der Indexraum für dieses Programm dargestellt, wobei exemplarisch die Partitionierung für den Zeitpunkt $t = 0$ skizziert ist.

Unter Berücksichtigung von Anmerkung 4.1.3 ergibt sich folgendes Ungleichungssystem zum Bestimmen der neuen Schleifengrenzen:

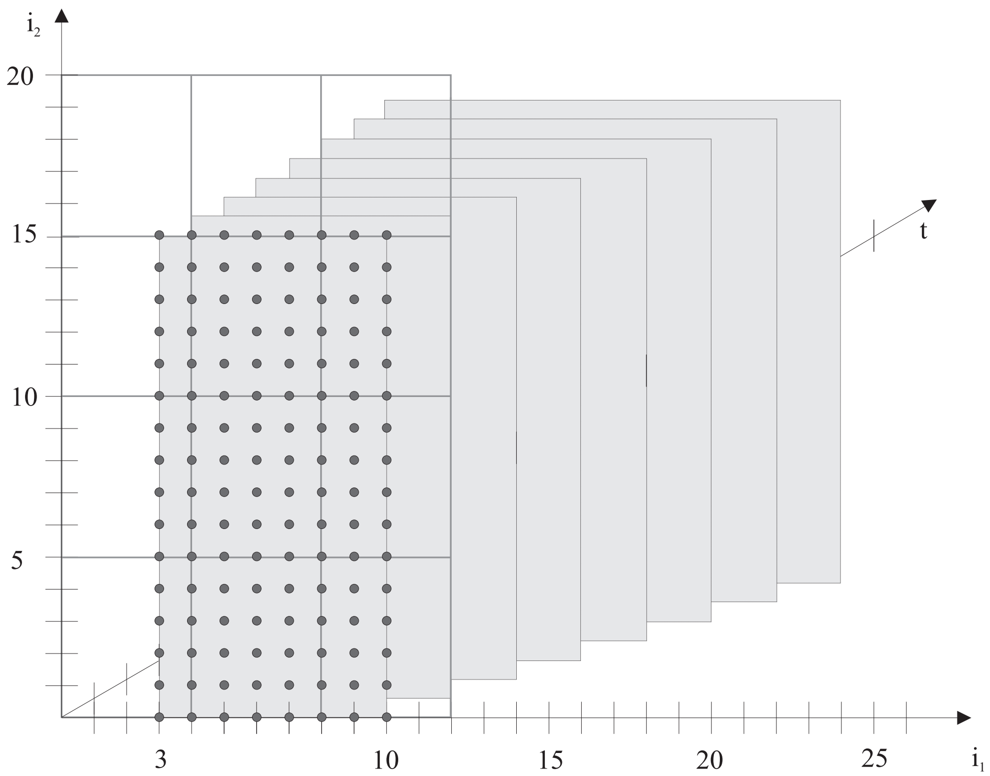

Abbildung 10: Indexraum für Beispiel 4.1.4 mit Partitionierung des parallelen Indexraums für $t = 0$

Beschreibung des Indexraums:

$$\begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \leq \begin{pmatrix} 3 \\ 10+t \\ 0 \\ 15 \end{pmatrix}$$

Begrenzungen für $\vec{rp}$:

$$\begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} rp_1 \\ rp_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 3 \\ 0 \\ 4 \end{pmatrix}$$

Beziehung zwischen realen und virtuellen Prozessoren:

$$\begin{pmatrix} 4 & 0 \\ 0 & 5 \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} + \begin{pmatrix} rp_1 \\ rp_2 \end{pmatrix} = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix}$$

Nach Anwendung der Fourier-Elimination, die im Rahmen des LooPo-Projektes implemeniert wurde, ergibt sich automatisch folgendes Ungleichungssystem:

$$\begin{array}{rcl} 0 \leq & rp_1 & \leq 3 \\ 0 \leq & rp_2 & \leq 4 \\ \lceil -\frac{1}{4} rp_1 + \frac{3}{4} \rceil \leq & t_1 & \leq \lfloor -\frac{1}{4} rp_1 + \frac{10}{4} + \frac{1}{4} t \rfloor \\ \lceil -\frac{1}{5} rp_2 \rceil \leq & t_2 & \leq \lfloor -\frac{1}{5} rp_2 + 3 \rfloor \\ 4t_1 + rp_1 \leq & i_1 & \leq 4t_1 + rp_1 \\ 5t_2 + rp_2 \leq & i_2 & \leq 5t_2 + rp_2 \end{array}$$

Daraus erhalten wir die Schleifengrenzen für das asynchrone Zielprogramm, das sich nach Wiedereinfügen der ursprünglichen sequentiellen Schleife mit Index t als äußerste der sequentiellen Schleifen wie folgt ergibt:

parfor $rp_1 := 0$ **to** 3 **do**
  parfor $rp_2 := 0$ **to** 4 **do**
    for $t := 0$ **to** 7 **do**
      for $t_1 := \lceil -\frac{1}{4}rp_1 + \frac{3}{4} \rceil$ **to** $\lfloor -\frac{1}{4}rp_1 + \frac{10}{4} + \frac{1}{4}t \rfloor$ **do**
        for $t_2 := \lceil -\frac{1}{5}rp_2 \rceil$ **to** $\lfloor -\frac{1}{5}rp_2 + 3 \rfloor$ **do**
          $i_1 := 4t_1 + rp_1$
          $i_2 := 5t_2 + rp_2$
          {Schleifenkörper}

Anmerkung 4.1.5 Da die Indizes der sequentiellen Schleifen des nicht partitionierten Programms während der Fourier-Elimination als Parameter betrachtet werden, ist es möglich, daß sie in den Grenzen der rp-Schleifen auftreten. Für eine asynchrone Programmerzeugung darf aber eine derartige Abhängigkeit zwischen rp- und t-Schleifen vernachlässigt werden, weil derartige Grenzen redundant sind, wie folgende Überlegung zeigt:

Sei $f_l(t)$ bzw. $f_u(t)$ ein affiner Ausdruck in t für eine zusätzliche untere bzw. obere Grenze einer rp-Schleife. Diese Grenzen beschränken rp nur dann, wenn gilt:

$$\begin{array}{rcl} 0 & < & f_l(t) \\ f_u(t) & < & P_i - 1 \end{array}$$

Wäre dies der Fall, würden aber zum Zeitpunkt t alle Tiles weniger als die real vorhandenen P_i Prozessoren benutzen. Dies widerspricht jedoch der Konstruktion des Ungleichungssystems, das zumindest für Tiles, die vollständig im parallelen Indexraum liegen, die Ausnutzung aller P_i Prozessoren garantiert.

Somit ist es korrekt, die rp-Schleifen im asynchronen Fall immer als äußerste Schleifen mit Grenzen 0 und $P_i - 1$ aufzuzählen und die sequentielle t-Schleife nach innen zu ziehen.

Analog erhalten wir durch Fourier-Elimination mit entsprechend geänderter Eliminationsreihenfolge folgendes synchrone Zielprogramm:

for $t := 0$ **to** 7 **do**
  for $t_1 := 0$ **to** $\lfloor \frac{5}{2} + \frac{1}{4}t \rfloor$ **do**
    for $t_2 := 0$ **to** 3 **do**
      parfor $rp_1 := \max(0, 3 - 4t_1)$ **to** $\min(3, 10 + t - 4t_1)$ **do**

$$
\begin{array}{l}
\textbf{parfor } rp_2 := \max(0, -5t_2) \textbf{ to } \min(4, 15 - 5t_2) \textbf{ do} \\
\quad i_1 := 4t_1 + rp_1 \\
\quad i_2 := 5t_2 + rp_2 \\
\quad \{\text{Schleifenkörper}\}
\end{array}
$$

4.1.3 Schrittweiten $\neq 1$

Bei dem Verfahren von Irigoin/Ancourt und dessen Erweiterung sind wir stets von Schrittweite 1 ausgegangen, wie es in der gängigen Literatur [1], [21] üblich ist. Wie aber in Abschnitt 2.4 dargestellt, sind die Schrittweiten der Schleifen der Ziel-Programmteile Elemente aus $\mathbb{Q}$.

Sind die Schrittweiten ganzzahlig, aber echt größer als 1, werden durch die Partitionierung zu viele Punkte aufgezählt, denn die Schrittweiten der aus der Partitionierung resultierenden Schleifen sind immer 1. Daher testen wir in diesem Fall vor jedem auszuführenden Statement S, ob eine Iteration einem Punkt im Indexraum entspricht. Dies erfolgt offensichtlich durch folgende Abfrage:

$$
\begin{array}{l}
\textbf{if } ((i - l_i) \mod step_i) = 0) \\
\qquad \{\text{Statement } S\}
\end{array}
$$

wobei i der Schleifenindex, l_i die untere Grenze der Schleife und $step_i$ die Schrittweite der Schleife ist.

Für rationale Schrittweiten ist dieses Vorgehen nicht ausreichend. Die Grenzen von Schleifen mit rationalen Schrittweiten haben folgende allgemeine Form [22]:

$$
\textbf{parfor } i := \frac{\lceil a \rceil}{d} \textbf{ to } \frac{\lfloor b \rfloor}{d} \textbf{ step } \frac{c}{d} \textbf{ do}
$$

Hier wäre es zunächst naheliegend, untere und obere Schleifengrenzen, sowie die Schrittweite, mit dem Nenner d zu multiplizieren, wodurch sich eine Schleife ergäbe, die genausoviele Punkte aufzählt wie die ursprüngliche und eine ganzzahlige Schrittweite besitzt. Damit könnte die Schleife mit oben genannter Lösung partitioniert werden. Laut Interpretation der Allokation repräsentieren aber alle in einer parallelen Schleife aufgezählten Punkte den virtuellen Prozessor, der sich durch Abrunden des jeweiligen Wertes ergibt. D.h. mit Indexpunkt $\frac{a}{d}$ wird der virtuelle Prozessor $\lfloor\frac{a}{d}\rfloor$ angesprochen. Dadurch können verschiedene Indexpunkte auf dieselben virtuellen Prozessoren abgebildet werden, was eine Gruppierung der parallelen Schleife bedeutet:

$$\textbf{parfor } i := \tfrac{1}{3} \textbf{ to } \tfrac{14}{3} \textbf{ step } \tfrac{1}{3} \textbf{ do}$$

Indexpunkt i	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{3}{3}$	$\frac{4}{3}$	$\frac{5}{3}$	$\frac{6}{3}$	$\frac{7}{3}$	$\frac{8}{3}$	$\frac{9}{3}$	$\frac{10}{3}$	$\frac{11}{3}$	$\frac{12}{3}$	$\frac{13}{3}$	$\frac{14}{3}$
virt. Proz. $\lfloor i\rfloor$	0	0	1	1	1	2	2	2	3	3	3	4	4	4

Das Problem besteht also darin, eine Parallelschleife mit gebrochener Schrittweite so zu partitionieren, daß diese Gruppierung und damit die Interpretation der Allokation erhalten bleibt. Die Idee zur Lösung dieses Problems ist, für die Parallelschleife zur Aufzählung der realen Prozessoren die gegebene gebrochene Schrittweite zu benutzen. Für obiges Beispiel zeigt folgende Tabelle Werte für rp, die die genannten Forderungen bei einer Partition mit 2 Prozessoren erfüllen:

rp	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{3}{3}$	$\frac{4}{3}$	$\frac{5}{3}$	0	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{3}{3}$	$\frac{4}{3}$	$\frac{5}{3}$	0	$\frac{1}{3}$	$\frac{2}{3}$
$\lfloor rp\rfloor$	0	0	1	1	1	0	0	0	1	1	1	0	0	0
tp			0						1				2	

Da sich die Schleifengrenzen algorithmisch durch Anwendung eines Projektionsverfahren (z.B. Fourier-Elimination) angewandt auf Ungleichungssystem (6) ergeben, ist für die Verarbeitung rationaler Schrittweiten eine Verallgemeinerung des Systems notwendig. Um zu erreichen, daß die $\vec{rp}$-Schleifen alle Punkte $\vec{0} \le \vec{rp} < \vec{P}$ aufzählen, benutzen wir als neue Obergrenze für $\vec{rp}$:

$$\vec{rp} \le \begin{pmatrix} P_0 - \frac{1}{d_0} \\ \vdots \\ P_n - \frac{1}{d_n} \end{pmatrix}$$

wobei d_i den Nenner der Schrittweite der i-ten Schleife bezeichnet.

Für die Untergrenzen der $\vec{rp}$-Schleifen muß ebenfalls eine neue Festlegung gefunden werden, da sich bei rationalen Schrittweiten eventuell rationale Bezugspunkte für die Tiles ergeben. Um darauf formal eingehen zu können, benötigen wir eine Verallgemeinerung des Modulo-Operators auf $\mathbb{Q}$:

Definition 4.1.6 (Modulo-Operator auf $\mathbb{Q}$) Sei $a, b \in \mathbb{Q}$, und $a = \frac{a_1}{a_2}$, $b = \frac{b_1}{b_2}$ mit $a_1, a_2, b_1, b_2 \in \mathbb{Z}$. Dann ist der Modulo-Operator auf $\mathbb{Q}$, $\mathrm{mod}_{\mathbb{Q}}$, folgendermaßen definiert:

$$a \ \mathrm{mod}_{\mathbb{Q}} \ b \ := a - \lfloor \frac{a_1 b_2}{a_2 b_1} \rfloor b$$

Offensichtlich ergibt sich ein rationaler Bezugspunkt bezüglich einer Schleife i, wenn gilt:

$$P_i \ \mathrm{mod}_{\mathbb{Q}} \ step_i \neq 0$$

Anmerkung 4.1.7 Wie in Anmerkung 4.1.1 erläutert, ist die Berechnung eines Gitters von den Bezugspunkten im Rahmen des Ungleichungssystems, das die Tile-Form repräsentiert, abhängig. Rationale Bezugspunkte liegen jedoch nicht notwendigerweise im Rahmen. Um dennoch eine Berechnung zu ermöglichen, könnte man zunächst die Bezugspunkte aller angrenzenden Tiles zur Auswahl für die Spalten des Gitters heranziehen. Damit erhalten wir ein rationales Gitter, das aber die Bezugspunkte aller Tiles nicht korrekt aufzählt, wie folgendes Beispiel veranschaulicht:

$$\textbf{parfor } i := 0 \textbf{ to } \tfrac{22}{3} \textbf{ step } \tfrac{2}{3} \textbf{ do}$$

Sei $P = 3$. Wählen wir als Bezugspunkt eines angrenzenden Tiles den Punkt $\frac{10}{3}$, erhalten wir: $L = \left(\frac{10}{3}\right)$. Wie in untenstehender Tabelle zu sehen, beginnt das Tile für $t = 2$ mit dem Indexpunkt $\frac{18}{3}$, während $L(2) = \frac{20}{3}$ ergibt, und damit den Indexpunkt $\frac{18}{3}$ nicht aufzählt.

i	0	$\frac{2}{3}$	$\frac{4}{3}$	$\frac{6}{3}$	$\frac{8}{3}$	$\frac{10}{3}$	$\frac{12}{3}$	$\frac{14}{3}$	$\frac{16}{3}$	$\frac{18}{3}$	$\frac{20}{3}$	$\frac{22}{3}$
$\lfloor i \rfloor$	0	0	1	2	2	3	4	4	5	6	6	7
$\lfloor rp \rfloor$	0	0	1	2	2	0	1	1	2	0	0	1

Eine korrekte Berechnung erhalten wir, indem wir das Gitter wie in Anmerkung 4.1.1 berechnen, aber während der Berechnung alle Bezugspunkte als nach unten gerundet betrachten. Damit ergibt sich zwar auch nicht der korrekte Wert des Bezugspunktes, wir erhalten aber stets den abgerundeten Wert dieses Punktes:

$$L\vec{t} = \lfloor \vec{\imath}_0 \rfloor$$

Um die Differenz zwischen rationalem Bezugspunkt eines Tiles t und $L(t)$ auszugleichen, bestimmen wir einen Offset für rp, sodaß die Gleichung $Lt + rp = i$ auch für rationale Schrittweiten gültig ist. Dazu berechnen wir die Differenz zwischen Bezugspunkt und erstem ganzzahligen Punkt eines Tiles. Wir wollen zunächst den rationalen Bezugspunkt $\vec{\imath}_0$ eines Tiles $\vec{t}$ berechnen. Falls die zu partitionierende i-te Schleife bei 0 zu zählen beginnt, ergibt sich die i-te Komponente des Bezugspunktes durch:

$$(\vec{\imath}_0)_i = \left\lceil \frac{\left(L\vec{t}\right)_i}{\left(\overrightarrow{step}\right)_i} \right\rceil \left(\overrightarrow{step}\right)_i$$

wobei $\overrightarrow{step}$ den Vektor aller Schrittweiten bezeichnet.

Ist die Untergrenze lb_i der i-ten Schleife ungleich 0, muß zusätzlich die Verschiebung $\left(\overrightarrow{olb}\right)$ des Bezugspunktes aufgrund der Untergrenze berechnet werden:

$$\left(\overrightarrow{olb}\right)_i = lb_i \bmod_{\mathbb{Q}} \left(\overrightarrow{step}\right)_i$$

Zusammengefaßt erhält man den Offset $\vec{o}$:

$$\vec{o} = \left(\vec{\imath}_0 + \overrightarrow{olb}\right) - \lfloor \vec{\imath}_0 \rfloor$$

Addieren wir diesen Offset auf die bisherige ganzzahlige Untergrenze 0 von rp, erhalten wir für rp den gleichen Werteverlauf der Rundungsdifferenz wie für i und erreichen damit, daß die Interpretation der Allokation gültig bleibt. Für das Beispiel aus Anmerkung 4.1.7 berechnen wir für rp:

rp	0	$\frac{2}{3}$	$\frac{4}{3}$	$\frac{6}{3}$	$\frac{8}{3}$	$\frac{1}{3}$	$\frac{3}{3}$	$\frac{5}{3}$	$\frac{7}{3}$	0	$\frac{2}{3}$	$\frac{4}{3}$
$\lfloor rp \rfloor$	0	0	1	2	2	0	1	1	2	0	0	1

Abschließend ergibt sich folgendes verallgemeinertes Ungleichungssystem zur Partitionierung von Schleifen mit rationalen Schrittweiten:

$$\begin{aligned} \vec{rp} &\geq \vec{o} \\ \vec{rp} &\leq \begin{pmatrix} P_0 - \frac{1}{d_0} \\ \vdots \\ P_n - \frac{1}{d_n} \end{pmatrix} \\ B\vec{\imath} &\leq \vec{b} \\ S(\vec{\imath} - L\vec{t}) &\leq \vec{s} \\ L\vec{t} + \vec{rp} &\geq \vec{\imath} \\ L\vec{t} + \vec{rp} &\leq \vec{\imath} \end{aligned} \tag{9}$$

Nach Berechnung der Schleifengrenzen müssen die sich ergebenden $\vec{rp}$-Schleifen mit der rationalen Schrittweite der ursprünglichen Schleife aufgezählt werden.

4.1.4 Rationale Schrittweiten in der Praxis

Zur Berechnung der Schleifengrenzen setzt LooPo Verfahren (z.B. Fourier-Elimination) ein, die keine $floor$- bzw. *ceil*-Operatoren verarbeiten können. Daher kann das Ungleichungssystem (9) aus Abschnitt 4.1.3 in der Praxis nicht gelöst werden, weil $floor/ceil$ zum einen in den Untergrenzen der rp-Schleifen, zum anderen in der Beschreibung des Indexraums auftreten.

Wir werden in diesem Abschnitt ein Verfahren zeigen, mit dessen Hilfe wir Programme mit rationalen Schrittweiten, deren parallele Schleifen einen rechteckigen Indexraum aufspannen, bei synchroner Programmerzeugung partitionieren können.

- $floor/ceil$ in den Untergrenzen der rp-Schleifen
  Zur Lösung dieses Problems belassen wir die Ungleichung für die rp-Untergrenzen wie in Gleichung (6): $\vec{rp} \geq \vec{0}$ und addieren den Offset erst auf die sich durch Fourier-Elimination ergebende Untergrenze 0 der rp-Schleife. Würde man den Offset im System belassen, so könnte er neben den Untergrenzen für die rp-Schleifen auch Einfluß auf die Grenzen der t-Schleifen haben. Folgende Überlegung zeigt jedoch, daß dieser Fall nicht auftreten kann: Aufgrund der Konstruktion des Offsets entsteht lediglich eine Verschiebung der Untergrenze der rp-Schleife in

Abhängigkeit von t, die Anzahl der Tiles und damit die Grenzen der t-Schleifen bleiben davon jedoch unberührt. Also ist es auch für diese Grenzen korrekt, das System ohne den Offset für rp zu formulieren.

Da der Offset von den Tile-Koordinaten t abhängt, beschränkt sich die Partitionierung bei rationalen Schrittweiten auf den synchronen Fall.

- $ceil$ bzw. $floor$ in den Unter- bzw. Obergrenzen der Schleifen, die den Indexraum begrenzen
  Um ein Verfahren zu entwickeln, das Programme mit rechteckigem parallelen Indexraum partitioniert, benötigen wir zunächst folgendes Lemma:

Lemma 4.1.8 Bei Programmen, deren parallele Schleifen ganzzahlig aufgezählt werden, kann zur Berechnung der durch die Partitionierung entstehenden Schleifengrenzen mittels Fourier-Elimination während der Elimination $floor/ceil$ vernachlässigt werden. Durch anschließendes Einfügen von $floor/ceil$ um die neuen Grenzen, werden wieder genau die ganzzahligen Indexpunkte des Ausgangssystems aufgezählt.

Beweis:

1. Durch Vernachlässigung von $floor$ und $ceil$ vergrößert sich aufgrund der ganzzahligen Schrittweite der Indexraum in Dimension i gegebenenfalls in $\mathbb{Q}$, jedoch nicht in $\mathbb{Z}$.
2. Gleiches gilt für rp und t:
   (a) Angenommen, sowohl rp als auch t hätten in $\mathbb{Z}$ einen größeren Wertebereich, so würde dies wegen der im System enthaltenen Gleichung

   $$Lt + rp = i$$

   Punkt 1) widersprechen, da damit auch i einen größeren Wertebereich besitzen würde.

   (b) Angenommen, rp vergrößert sich bezüglich $\mathbb{Z}$, t verkleinert sich entsprechend, und umgekehrt:
   Im asynchronen Fall erhalten wir aufgrund der Eliminationsreihenfolge für rp genau die vorgegebenen Grenzen $(0, P-1)$,

sodaß t bei Einhaltung der Gleichung $Lt + rp = i$ in $\mathbb{Z}$ weder vergrößert noch verkleinert werden kann. Da aber die Eliminationsreihenfolge keinen Einfluß auf die Lösungsmenge des Ungleichungssystems hat, gilt dies auch im synchronen Fall.

Wie bereits erwähnt, haben die Schleifengrenzen bei rationalen Schrittweiten folgende allgemeine Form:

$$\textbf{parfor } i := \frac{\lceil a \rceil}{d} \textbf{ to } \frac{\lfloor b \rfloor}{d} \textbf{ step } \frac{c}{d} \textbf{ do}$$

Da in unserem Fall also $floor/ceil$ nur in den Zählern der Schleifengrenzen auftreten, und wir außerdem von gebrochenen Schrittweiten ausgehen, skalieren wir zunächst den Indexraum und alle Tiles mit dem Nenner d. In Ungleichungssystem (9) bewirkt dies eine Multiplikation der Beschreibung des Indexraums, des Gitters, der Grenzen von rp und der Tile-Form mit d.

Dieses System partitioniert den skalierten Indexraum auf dieselbe Weise wie den ursprünglichen Indexraum, nur mit einer Vergrößerung des gesamten Systems um den Nenner der Schrittweite (siehe Abbildung 11). Um nach der Fourier-Elimination Schleifengrenzen zu erhalten, die den ursprünglichen Indexraum partitionieren, werden diese Grenzen zurückskaliert, d.h. alle Grenzen der rp-Schleifen und die Vorkommen von L müssen wieder durch den Nenner der Schrittweite dividiert werden. Da sich t einzig aus den Beziehungen im System ergibt, also vorher nicht skaliert wurde, werden diese Grenzen nicht zurückskaliert. Durch die Skalierung des gesamten Systems werden im skalierten Indexraum genausoviele Tiles aufgezählt, wie im ursprünglichen.

Wir wollen uns das Vorgehen an einem Beispiel verdeutlichen:

Beispiel 4.1.9 Gegeben sei folgende Parallelschleife mit gebrochener Schrittweite:

$$\textbf{parfor } i := \frac{\lceil (\frac{16}{7}) \rceil}{5} \textbf{ to } \frac{\lfloor (\frac{92}{3}) \rfloor}{5} \text{ step } \frac{4}{5} \textbf{ do}$$

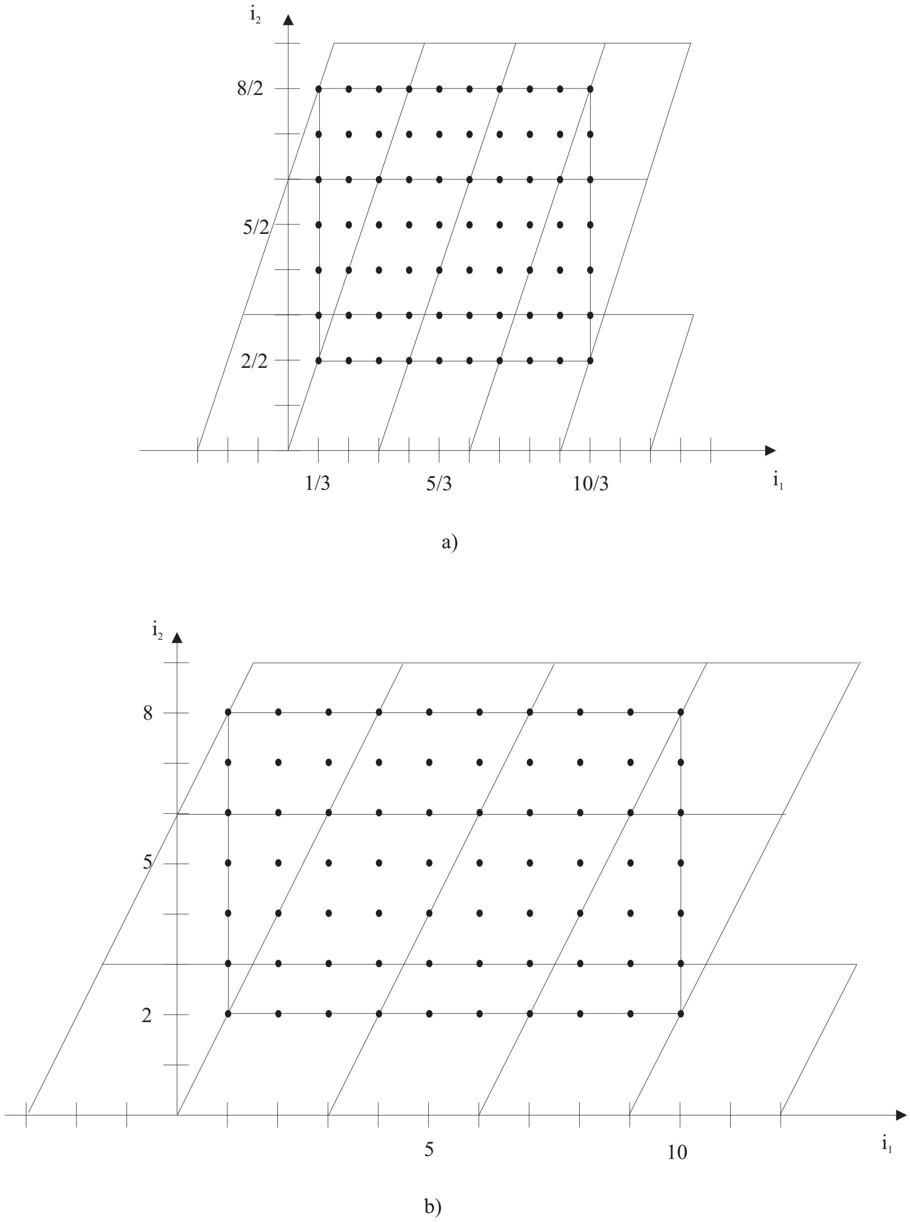

Abbildung 11: Partitionierung a) des ursprünglichen Indexraums und b) des skalierten Indexraums

Es seien 3 Prozessoren vorhanden. Durch Skalieren des Systems mit dem Nenner ergibt sich:

$$\begin{aligned} i &\geq \lceil \frac{16}{7} \rceil \\ i &\leq \lfloor \frac{92}{3} \rfloor \\ rp &\geq 0 \\ rp &\leq (3*5)-1 \\ i &\geq (3*5)t+rp \\ i &\leq (3*5)t+rp \end{aligned}$$

Durch Fourier-Elimination gemäß Lemma 4.1.8 ergeben sich folgende Grenzen:

$$\begin{array}{rcl} \lceil -\frac{82}{105} \rceil \leq & t & \leq \lfloor \frac{92}{45} \rfloor \\ \lceil \frac{16}{7} - 15t \rceil \leq & rp & \leq \lfloor \frac{92}{3} - 15t \rfloor \\ 0 \leq & rp & \leq 14 \\ 15t+rp \leq & i & \leq 15t+rp \end{array}$$

Dieser Schleifensatz partitioniert den skalierten Indexraum. Um eine Partitionierung des ursprünglichen Indexraums zu erreichen, müssen wir diese Grenzen zurückskalieren:

$$\begin{array}{rcl} \lceil -\frac{82}{105} \rceil \leq & t & \leq \lfloor \frac{92}{45} \rfloor \\ \frac{\lceil \frac{16}{7} - 15t \rceil}{5} \leq & rp & \leq \frac{\lfloor \frac{92}{3} - 15t \rfloor}{5} \\ 0 \leq & rp & \leq \frac{14}{5} \\ \frac{15}{5}t+rp \leq & i & \leq \frac{15}{5}t+rp \end{array}$$

Um einen korrekten Verlauf der rp-Schleife zu erhalten, muß auf die Untergrenze 0 von rp noch der Offset

$$o = \left(\left(\lfloor \frac{3t}{\frac{4}{5}} \rfloor * \frac{4}{5} + \frac{3}{5} \right) - (3t) \right)$$

addiert werden.

Somit ergibt sich nach der Partitionierung folgender Schleifensatz:

$$
\begin{array}{l}
\textbf{for } t := 0 \textbf{ to } 2 \textbf{ step } 1 \textbf{ do} \\
\quad \textbf{parfor } rp := \max\left(0 + \left(\left(\lfloor \frac{3t}{\frac{4}{5}} \rfloor * \frac{4}{5} + \frac{3}{5}\right) - (3t)\right), \frac{\lceil \frac{16}{7} - 15t \rceil}{5}\right) \textbf{ to} \\
\qquad\qquad\qquad\qquad \min\left(\frac{14}{5}, \frac{\lfloor \frac{92}{3} - 15t \rfloor}{5}\right) \textbf{ step } \frac{4}{5} \textbf{ do} \\
\qquad\qquad i := \frac{15}{5} t + rp
\end{array}
$$

Nachfolgende Tabelle soll den Zusammenhang zwischen rp, dem zugehörigen realen Prozessor $\lfloor rp \rfloor$, i und dem zugehörigen virtuellen Prozessor $\lfloor i \rfloor$ veranschaulichen:

t	rp	$\lfloor rp \rfloor$	i	$\lfloor i \rfloor$
0	$\frac{3}{5}$	0	$\frac{3}{5}$	0
	$\frac{7}{5}$	1	$\frac{7}{5}$	1
	$\frac{11}{5}$	2	$\frac{11}{5}$	2
1	0	0	$\frac{15}{5}$	3
	$\frac{4}{5}$	0	$\frac{19}{5}$	3
	$\frac{8}{5}$	1	$\frac{23}{5}$	4
	$\frac{12}{5}$	2	$\frac{27}{5}$	5
2	—	—		

Hier ist anzumerken, daß für $t = 2$ die rp-Schleife leer ist. Das ergibt sich dadurch, daß bei der Berechnung der Schleifengrenzen zum Aufzählen der Tiles durch Fourier-Elimination keine Schrittweiten berücksichtigt werden, d.h. es wird immer von Schrittweite 1 ausgegangen. Im skalierten Indexraum ist die Obergrenze $\frac{92}{3}$, somit müßten mit Schrittweite 1 im Tile $t = 2$ noch die Punkte 28 bis 30 aufgezählt werden, oder wieder zurückskaliert die Punkte $\frac{28}{5}$ bis $\frac{30}{5}$. Durch die Schrittweite $\frac{4}{5}$ wäre der nächste aufzuzählende Punkt $\frac{31}{5}$, der aber nicht mehr im Indexraum liegt. Um keine Punkte aufzuzählen, die nicht im Indexraum liegen, ergibt sich in diesem Fall die rp-Schleife für $t = 2$ automatisch zu einer leeren Schleife.

Wenn parallele Schleifen von äußeren parallelen Schleifen abhängen, liefert das eben vorgestellte Vorgehen keine korrekten Ergebnisse.

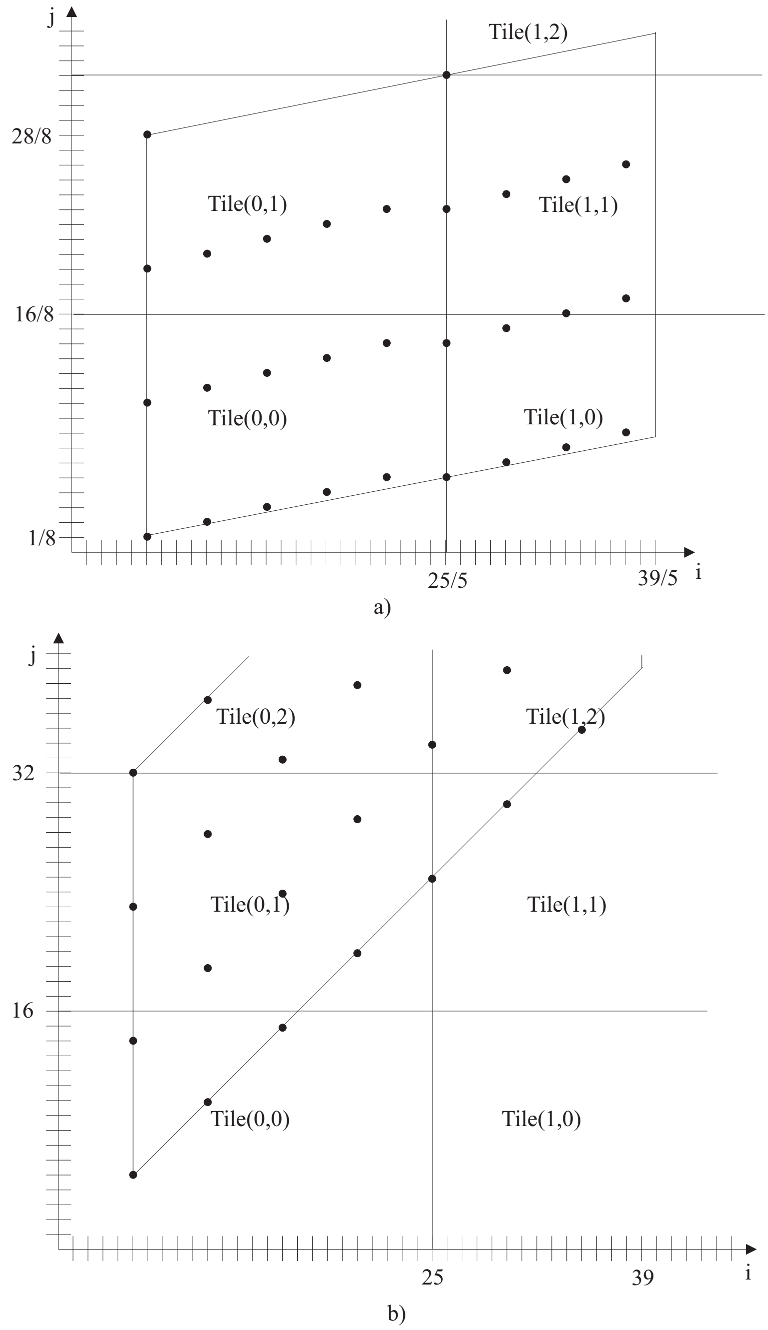

Abbildung 12: Mögliche Partitionierung a) im Indexraum und b) im skalierten Indexraum

$$\textbf{parfor } i := \frac{\lceil a \rceil}{d} \textbf{ to } \frac{\lfloor b \rfloor}{d} \textbf{ step } \frac{c}{d} \textbf{ do}$$
$$\quad \textbf{parfor } j := \frac{\lceil i \rceil}{e} \textbf{ to } \frac{\lfloor i+g \rfloor}{e} \textbf{ step } \frac{f}{e} \textbf{ do}$$

Da aufgrund der allgemeinen Form von Schleifengrenzen bei gebrochenen Schrittweiten der Index i nur im Zähler der Grenzen von j auftauchen kann, wird bei der Fourier-Elimination der Indexraum bezüglich j nach unten durch $\lceil a \rceil$ begrenzt, statt durch $\lceil \frac{\lceil a \rceil}{d} \rceil$. Dadurch wird der Indexraum in Dimension j verzerrt. Das folgende Beispiel soll die Problematik veranschaulichen:

Beispiel 4.1.10 Gegeben sei folgender Schleifensatz:

$$\textbf{parfor } i := \frac{\lceil \frac{30}{7} \rceil}{5} \textbf{ to } \frac{\lfloor \frac{118}{3} \rfloor}{5} \textbf{ step } \frac{4}{5} \textbf{ do}$$
$$\quad \textbf{parfor } j := \frac{\lceil i \rceil}{8} \textbf{ to } \frac{\lfloor i+27 \rfloor}{8} \textbf{ step } \frac{9}{8} \textbf{ do}$$

Für die Partitionierung stehen willkürlich gewählt in der ersten Dimension 5 Prozessoren, in der zweiten Dimension 2 Prozessoren zur Verfügung. Vergleicht man in Abbildung 12 den ursprünglichen Indexraum mit dem skalierten Indexraum, wird sofort ersichtlich, daß der skalierte Indexraum durch die Abhängigkeit der j-Schleife von den Grenzen der i-Schleife viel steiler in den Begrenzungen von j wird, und somit die Tiles nicht korrekt wie im ursprünglichen Indexraum aufgezählt werden. Beispielsweise liegt kein Punkt von Tile $(1, 0)$ im skalierten Indexraum, im ursprünglichen Indexraum werden 6 Punkte dieses Tiles aufgezählt.

Wie wir in Abschnitt 4.1.3 gesehen haben, ist die Partitionierung von Schleifen mit rationalen Schrittweiten von theoretischer Seite vollständig herzuleiten. Durch die uns bekannten Projektionsverfahren zur Bestimmung der Schleifengrenzen (z.B. Fourier-Elimination) beschränkt sie sich in der Praxis jedoch ausschließlich auf synchrone Programmerzeugung, und dabei nur auf Schleifen, die einen rechteckigen parallelen Indexraum aufspannen.

4.2 Automatische Generierung einer Tile-Form

In den bisherigen Untersuchungen sind wir stets von einer gegebenen Tile-Form ausgegangen. In diesem Abschnitt werden wir einen Algorithmus entwickeln, der automatisch eine Tile-Form generiert, die den Zielen der Partitionierung, wie in Abschnitt 2.5 erläutert, möglichst nahe kommt. Wir wollen also eine Tile-Form berechnen, durch die Kommunikationsaufwand minimiert wird, jedoch stets unter der Voraussetzung, daß die Größe des Tiles es erlaubt, das partitionierte Programm auf einer Rechnerarchitektur mit beschränkter Anzahl an Prozessoren auszuführen. Während es viele Forschungsergebnisse gibt, um eine möglichst optimale Tile-Form bei der LSGP-Partitionierung automatisch zu generieren [3], [4], [24], bleibt die automatische Tile-Form-Generierung im Rahmen der LPGS-Partitionierung, auf die wir hier näher eingehen werden, in der Literatur weitgehend unerwähnt.

Verfahren zum Generieren einer bestmöglichen Tile-Form

Wir gehen in dieser Arbeit stets davon aus, daß Kommunikation nur durch Abhängigkeiten verursacht wird. Deshalb benötigen wir zunächst die transformierten Abhängigkeiten des Zielprogramms. Dazu führen wir den Begriff der h-Transformation ein [8]:

Definition 4.2.1 (h-Transformation) Seien $S_1, S_2 \in \Omega_P$ zwei Statements eines Programms P, wobei S_2 von S_1 abhängig sei, i.Z. $S_1 \delta S_2$. Eine Abbildung $h \in D_{S_2} \to D_{S_1}$ heißt *h-Transformation* dieser Abhängigkeit, falls sie jeden Indexpunkt einer abhängigen Instanz von S_2 auf den Indexpunkt der zugehörigen Instanz von S_1 abbildet. Formal:

$$(\forall\, \vec{\imath}, \vec{\jmath} \,:\, \vec{\imath} \in D_{S_1} \wedge \vec{\jmath} \in D_{S_2} \wedge (S_1, \vec{\imath})\delta(S_2, \vec{\jmath}) \,:\, h(\vec{\jmath}) = \vec{\imath})$$

Diese Darstellung von Abhängigkeiten ermöglicht es uns, auch nicht-uniforme Abhängigkeiten in einer geschlossenen Form zu notieren. Wir benutzen im weiteren die Begriffe h-Transformation und Abhängigkeit synonym.

Offensichtlich gilt für eine transformierte Abhängigkeit h^{tr}:

$$h^{\mathrm{tr}} T_{S_2} = T_{S_1} h$$

wobei T_{S_1}, T_{S_2} die Transformationen der Statements S_1, S_2 sind. Damit erhalten wir aufgrund der Invertierbarkeit von Transformationsmatrizen die transformierte Abhängigkeit durch

$$h^{\mathrm{tr}} = T_{S_1} h T_{S_2}^{-1}$$

Da parallele Schleifen keine Abhängigkeiten tragen, können zwischen Tiles innerhalb eines Zeitschritts i keine Abhängigkeiten existieren. Aus diesem

Grund kann der parallele Indexraum auch beliebig partitioniert werden, ohne dabei Abhängigkeiten zu verletzen. Wohl aber kann es Abhängigkeiten zwischen nicht notwendigerweise verschiedenen Tiles in einem Zeitschritt i und einem Zeitschritt $i+k$ geben. Diese verursachen aber nur dann Kommunikation, wenn die abhängigen Instanzen von unterschiedlichen Prozessoren ausgeführt werden. Wir werden sehen, daß Kommunikation, die durch Abhängigkeiten zwischen Indexpunkten zu verschiedenen Zeitpunkten entsteht, durch die Tile-Form im parallelen Indexraum u.U. eliminiert werden kann.

Betrachten wir dazu Abbildung 13. Im dreidimensionalen Indexraum, der durch die Parallelschleifen mit Indizes i_1 und i_2, sowie durch die sequentielle Schleife mit Index t erzeugt wird, existieren zwei uniforme Abhängigkeiten, $h_1^{\mathrm{tr}}(i_1, i_2, t) = (i_1 - 3, i_2, t-1)$ und $h_2^{\mathrm{tr}}(i_1, i_2, t) = (i_1 - 2, i_2 - 2, t-1)$, die durch folgende Distanzvektoren dargestellt werden können: $\vec{d_1} = (3,0,1)$, $\vec{d_2} = (2,2,1)$.
In Teil a) der Abbildung wurde für die Partitionierung willkürlich eine rechteckige Tile-Form gewählt. In diesem Fall verursachen beide Abhängigkeiten Kommunikation: Durch die Abhängigkeit mit Distanzvektor $\vec{d_2}$ entsteht z.B. Kommunikation zwischen dem realen Prozessor $(0,0)$, der in Tile $(0,0)$ zum Zeitpunkt $t=0$ den Indexpunkt $(0,0,0)$ ausführt, und dem realen Prozessor $(2,0)$, der in Tile $(0,1)$ zum Zeitpunkt $t=1$ den Indexpunkt $(2,2,1)$ ausführt. Die Abhängigkeit mit Distanzvektor $\vec{d_1}$ bewirkt Kommunikation innerhalb des Tiles $(0,0)$ zum Zeitpunkt $t=0$ und $t=1$. Hier benötigt Prozessor $(3,0)$ Daten von Prozessor $(0,0)$. Die gewählte Tile-Form bewirkt somit keine Optimierung in Bezug auf Minimierung des Kommunikationsaufwands. Dagegen wird durch die Wahl der Tile-Form in Teil b) der Abbildung durch keine der beiden Abhängigkeiten Kommunikation bewirkt. Die Punkte $(0,0,0), (3,0,1)$ und $(2,2,1)$, zwischen denen die Abhängigkeiten existieren, werden alle von Prozessor $(0,0)$ ausgeführt. Durch die Uniformität der Abhängigkeiten und die gleiche Partitionierung der Tiles in jedem Zeitschritt verursachen diese Abhängigkeiten, die in der Abbildung repräsentativ für den Nullpunkt dargestellt sind, im gesamten Indexraum keine Kommunikation.

Abbildung 14 zeigt die Tile-Form aus Abbildung 13b) im parallelen In-

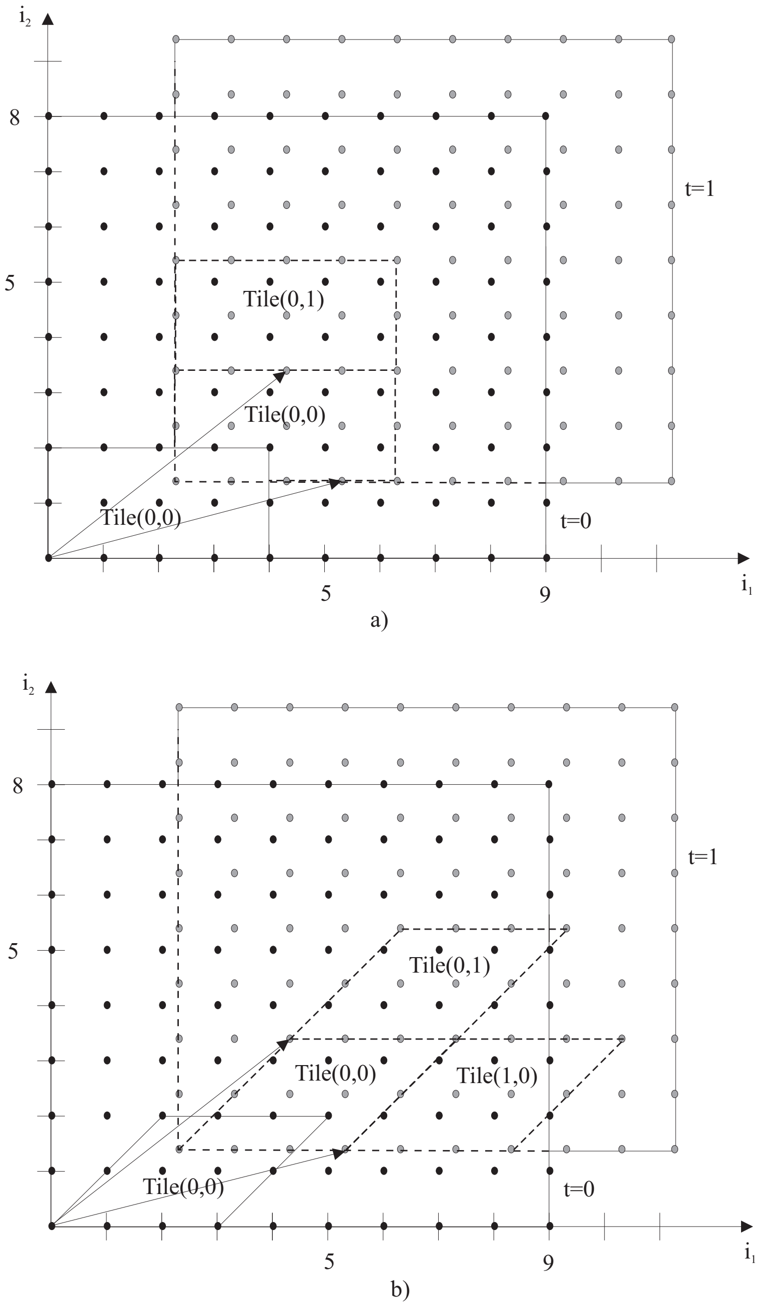

Abbildung 13: a) Partitionierung ohne Minimierung des Kommunikationsaufwands, b) Partitionierung mit minimaler Kommunikation

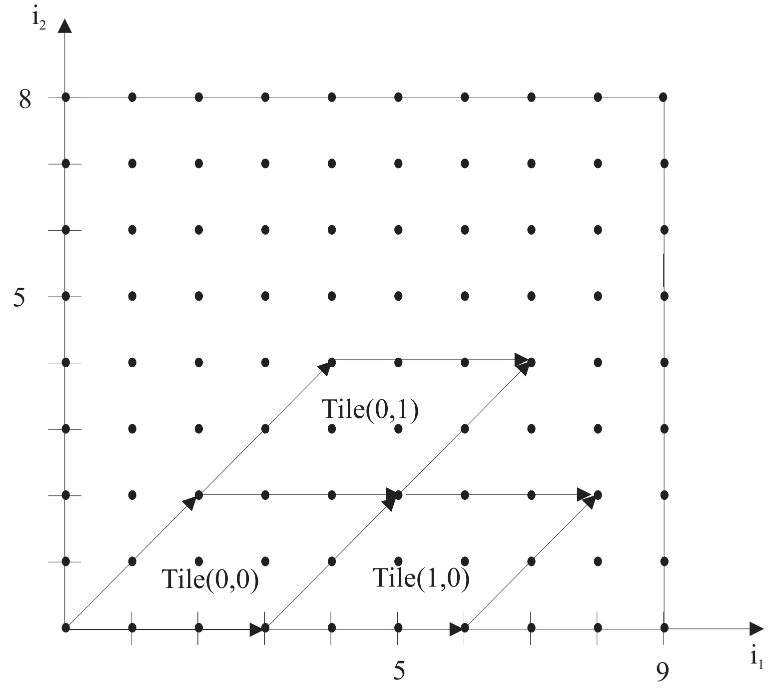

Abbildung 14: Partitionierung des parallelen Indexraums anhand der auf die parallelen Komponenten projezierten Abhängigkeiten

dexraum und zusätzlich die auf die parallelen Komponenten projezierten[8] Abhängigkeiten $\vec{d_1}^{\mathrm{p}}$ und $\vec{d_2}^{\mathrm{p}}$. Daraus wird ersichtlich, daß die Tile-Form genau so gewählt wurde, daß sich die projezierten Distanzvektoren als Spalten des zugehörigen Gitters ergeben: $L = (\vec{d_1}^{\mathrm{p}}, \vec{d_2}^{\mathrm{p}})$

Wir werden im nächsten Satz formal beweisen, daß allgemein durch diese Wahl für die Spalten des Gitters eine Tile-Form generiert wird, die den Kommunikationsaufwand minimiert. Die Konstruktion des Gitters durch die projezierten Abhängigkeitsvektoren ermöglicht nur eine Eliminierung der Kommunikation, die durch uniforme Abhängigkeiten entsteht. Es können zwar durch die Partitionierung keine affinen Abhängigkeiten verletzt werden, da wir nur den parallelen Indexraum partitionieren, durch sie verursachte Kommunikation kann jedoch durch die gleiche Form aller Tiles nicht verringert werden. Aus diesem Grund werden für die automatische Generierung einer optimalen Tile-Form nur uniforme Abhängigkeiten berücksichtigt.

[8] Im folgenden bezeichnen wir mit der Notation $\vec{v}^{\mathrm{p}}$ die Projektion eines Vektors $\vec{v}$ auf die parallelen Komponenten des Indexraums. Analog bezeichne M^{p} die Matrix, deren Spaltenvektoren $\vec{v_i}^{\mathrm{p}}$ aus der Projektion der Spaltenvektoren $\vec{v_i}$ einer Matrix M bestehen.

Für den Beweis der Optimalität des am Beispiel vorgeführten Verfahrens zur Berechnung des Gitters benötigen wir zunächst folgendes Lemma, das aussagt, daß bei oben genannter Konstruktion des Gitters abhängige Indexpunkte immer in benachbarten Tiles liegen.

Lemma 4.2.2 Seien $\vec{d}_1, \ldots, \vec{d}_n$ die n linear unabhängigen Distanzvektoren eines Programms und sei zusätzlich $n = \dim(D^{\mathrm{par}})$. Für ein Gitter $L = (\vec{d}_1^{\,\mathrm{p}}, \ldots, \vec{d}_n^{\,\mathrm{p}})$ gilt: Existiert eine uniforme Abhängigkeit h^{tr} mit Distanzvektor $\vec{d}_l, 1 \leq l \leq n$, zwischen zwei Instanzen $\vec{\imath}$ und $\vec{\jmath}$ des Indexraums, so ist die Differenz der zugehörigen Tile-Koordinaten für $\vec{\imath}^{\mathrm{p}}$ und $\vec{\jmath}^{\mathrm{p}}$ stets gleich dem l-ten Einheitsvektor $\vec{e}_l$.

Beweis:

1. Nach Voraussetzung gilt:

$$
\begin{aligned}
& h^{\mathrm{tr}}(\vec{\jmath}) = \vec{\imath} \\
\Leftrightarrow \quad & \{ \ (\exists\, \vec{t}_i, \vec{rp}_i \, : \, \vec{t}_i \in \mathbb{Z}^n, \vec{0} \leq \vec{rp}_i < L\vec{1} \, : \, L\vec{t}_i + \vec{rp}_i = \vec{\imath}^{\mathrm{p}}) \ \} \\
& h^{\mathrm{tr}}(\vec{\jmath})^{\mathrm{p}} = L\vec{t}_i + \vec{rp}_i \\
\Leftrightarrow \quad & \{ \ \vec{\jmath}^{\mathrm{p}} - \overrightarrow{h^{\mathrm{tr}}(\jmath)}^{\mathrm{p}} = \vec{d}_l^{\,\mathrm{p}} \ \} \\
& \vec{\jmath}^{\mathrm{p}} - \vec{d}_l^{\,\mathrm{p}} = L\vec{t}_i + \vec{rp}_i \\
\Leftrightarrow \quad & \{ \ (\exists\, \vec{t}_j, \vec{rp}_j \, : \, \vec{t}_j \in \mathbb{Z}^n, \vec{0} \leq \vec{rp}_j < L\vec{1} \, : \, L\vec{t}_j + \vec{rp}_j = \vec{\jmath}^{\mathrm{p}}) \ \} \\
& L\vec{t}_j + \vec{rp}_j - \vec{d}_l^{\,\mathrm{p}} = L\vec{t}_i + \vec{rp}_i \\
\Rightarrow \quad & \{ \ \vec{\jmath}^{\mathrm{p}} \geq L\vec{t}_j \ \} \\
& L\vec{t}_j - \vec{d}_l^{\,\mathrm{p}} \leq L\vec{t}_i + \vec{rp}_i \\
\Rightarrow \quad & \{ \ \vec{rp}_i < L\vec{1} \ \} \\
& L\vec{t}_j - \vec{d}_l^{\,\mathrm{p}} < L\vec{t}_i + L\vec{1} \\
\Leftrightarrow \quad & \{ \ \vec{d}_l^{\,\mathrm{p}} = L\vec{e}_l \ \} \\
& L\vec{t}_j - L\vec{e}_l < L(\vec{t}_i + \vec{1}) \\
\Leftrightarrow \quad & \{ \text{ arithmetische Umformung } \} \\
& L(\vec{t}_j - \vec{t}_i) < L(\vec{e}_l + \vec{1}) \\
\Leftrightarrow \quad & \{ \text{ L invertierbar } \} \\
& \vec{t}_j - \vec{t}_i < \vec{e}_l + \vec{1} \\
\Leftrightarrow \quad & \{ \text{ arithmetische Umformung } \} \\
& \vec{t}_j - \vec{t}_i \leq \vec{e}_l
\end{aligned}
$$

2. Nach Voraussetzung gilt:

$$
\begin{array}{ll}
 & h^{\mathrm{tr}}(\vec{\jmath}) = \vec{\imath} \\
\Leftrightarrow & \{\ (\exists\, \vec{t_i}, \vec{rp}_i\ :\ \vec{t_i} \in \mathbb{Z}^n, \vec{0} \leq \vec{rp}_i < L\vec{1}\ :\ L\vec{t_i} + \vec{rp}_i = \vec{\imath}^{\mathrm{p}})\ \} \\
 & h^{\mathrm{tr}}(\vec{\jmath})^{\mathrm{p}} = L\vec{t_i} + \vec{rp}_i \\
\Leftrightarrow & \{\ \vec{\jmath}^{\mathrm{p}} - h^{\mathrm{tr}}(\vec{\jmath})^{\mathrm{p}} = \vec{d_l}^{\mathrm{p}}\ \} \\
 & \vec{\jmath}^{\mathrm{p}} - \vec{d_l}^{\mathrm{p}} = L\vec{t_i} + \vec{rp}_i \\
\Rightarrow & \{\ \vec{rp}_i < L\vec{1}\ \} \\
 & L\vec{t_j} + L\vec{1} - \vec{d_l}^{\mathrm{p}} > L\vec{t_i} + \vec{rp}_i \\
\Rightarrow & \{\ \vec{rp}_i \geq \vec{0}\ \} \\
 & L(\vec{t_j} + \vec{1}) - \vec{d_l}^{\mathrm{p}} > L\vec{t_i} \\
\Leftrightarrow & \{\ \vec{d_l}^{\mathrm{p}} = L\vec{e_l}\ \} \\
 & L(\vec{t_j} + \vec{1}) - L\vec{e_l} > L\vec{t_i} \\
\Leftrightarrow & \{\ \text{arithmetische Umformung}\ \} \\
 & L(\vec{t_j} - \vec{t_i}) > L(\vec{e_l} - \vec{1}) \\
\Leftrightarrow & \{\ \text{L invertierbar}\ \} \\
 & \vec{t_j} - \vec{t_i} < \vec{e_l} - \vec{1} \\
\Leftrightarrow & \{\ \text{arithmetische Umformung}\ \} \\
 & \vec{t_j} - \vec{t_i} \geq \vec{e_l}
\end{array}
$$

Aus 1. und 2. ergibt sich:

$$\vec{t_j} - \vec{t_i} = \vec{e_l}$$

Mit Hilfe dieses Lemmas läßt sich beweisen, daß durch die Tile-Form im n-dimensionalen parallelen Indexraum ein Programm mit n linear unabhängigen uniformen Abhängigkeiten so partitioniert werden kann, daß sämtliche Kommunikation eliminiert wird.

Satz 4.2.3 (Partitionierung ohne Kommunikationsaufwand) Der n-dimensionale parallele Indexraum eines Programms mit n linear unabhängigen Abhängigkeiten $\vec{d_1}, \ldots, \vec{d_n}$ wird durch das Gitter $L = (\vec{d_1}^{\mathrm{p}}, \ldots, \vec{d_n}^{\mathrm{p}})$ so partitioniert, daß im partitionierten Programm keine Kommunikation entsteht.

Beweis: Seien $\vec{\imath}, \vec{\jmath}$ zwei abhängige Instanzen mit $h(\vec{\jmath}) = \vec{\imath}$, und sei $\vec{d_l}, 1 \leq l \leq n$, der Distanzvektor von h^{tr}.
Es gilt:

1. $(\exists\, \vec{t_i}, \vec{rp_i} \,:\, \vec{t_i} \in \mathbb{Z}^n, \vec{0} \leq \vec{rp_i} < L\vec{1} \,:\, L\vec{t_i} + \vec{rp_i} = \vec{\imath}^{\mathrm{p}})$

2. $(\exists\, \vec{t_j}, \vec{rp_j} \,:\, \vec{t_j} \in \mathbb{Z}^n, \vec{0} \leq \vec{rp_j} < L\vec{1} \,:\, L\vec{t_j} + \vec{rp_j} = \vec{\jmath}^{\mathrm{p}})$

$$
\begin{array}{ll}
& L\vec{t_j} + \vec{rp_j} - L\vec{t_i} - \vec{rp_i} = \vec{\jmath}^{\mathrm{p}} - \vec{\imath}^{\mathrm{p}} \\
\Leftrightarrow & \{\ \vec{d_l}^{\mathrm{p}} = \vec{\jmath}^{\mathrm{p}} - \vec{\imath}^{\mathrm{p}}\ \} \\
& L\vec{t_j} + \vec{rp_j} - L\vec{t_i} - \vec{rp_i} = \vec{d_l}^{\mathrm{p}} \\
\Leftrightarrow & \{\ \text{arithmetische Umformung}\ \} \\
& L(\vec{t_j} - \vec{t_i}) + \vec{rp_j} - \vec{rp_i} = \vec{d_l}^{\mathrm{p}} \\
\Leftrightarrow & \{\ \vec{d_l}^{\mathrm{p}} = L\vec{e_l}\ \} \\
& L(\vec{t_j} - \vec{t_i}) + \vec{rp_j} - \vec{rp_i} = L\vec{e_l} \\
\Leftrightarrow & \{\ \text{Lemma 4.2.2}\ \} \\
& L\vec{e_l} + \vec{rp_j} - \vec{rp_i} = L\vec{e_l} \\
\Leftrightarrow & \{\ \text{arithmetische Umformung}\ \} \\
& \vec{rp_j} = \vec{rp_i}
\end{array}
$$

Das heißt, je zwei abhängige Iterationen werden auf demselben Prozessor $\vec{rp}$ ausgeführt, wodurch keine Abhängigkeit Kommunikation bewirkt.

Durch die Darstellung des Gitters ergibt sich eindeutig die Beschreibung der Tile-Form.

Anmerkung 4.2.4 Bei näherer Betrachtung des Verfahrens sieht man, daß die Anzahl an real vorhandenen Prozessoren nicht in die Konstruktion des Gitters miteinfließt. Für den Fall, daß in einer Dimension mehr Prozessoren vorhanden sind als von der Tile-Form benötigt, könnte man zunächst meinen, daß durch eine Streckung der Tile-Form in dieser Dimension um ein ganzzahliges Vielfaches die Optimalität des Gitters erhalten bleibt und gleichzeitig die Prozessoren besser ausgelastet werden. Würde man beispielsweise die Tile-Form aus Abbildung 14 in Dimension i_2 verdoppeln, d.h. würden Tile $(0,0)$ und $(0,1)$ zusammen die neue Tile-Form beschreiben, würden zwar Indexpunkt $(0,0)$ und $(4,4)$ auf demselben realen Prozessor $(0,0)$ ausgeführt, dazwischen läge aber ein anderer Prozessor, der die Daten berechnet, die Prozessor $(0,0)$ benötigt, um Iteration $(4,4)$ zu berechnen. Somit würde die Kommunikation, die durch die Abhängigkeit $\vec{d_1}$ verursacht wird, nicht eliminiert.

Für den Fall, daß in einer Dimension weniger Prozessoren als benötigt zur Verfügung stehen, bleibt bei einer Stauchung um ein ganzzahliges Vielfaches in dieser Dimension die Optimalität erhalten. Voraussetzung dafür ist aber, daß das sich neu ergebende Gitter wieder nur ganzzahlige Einträge besitzt, da ein rationales Gitter nicht definiert ist. Skaliert man beispielsweise in Abbildung 14 das Gitter in Dimension i_2 mit dem Faktor $\frac{1}{2}$, so ergibt sich als Gitter:

$$L = \begin{pmatrix} 3 & 1 \\ 0 & 1 \end{pmatrix}$$

Mit diesem Gitter werden z.B. die Punkte $(0,0)$ und $(2,2)$, zwischen denen eine Abhängigkeit existiert, auf demselben Prozessor ausgeführt. Durch Stauchung wird aber zwischen der Ausführung von $(0,0)$ und $(2,2)$ auch der Punkt $(1,1)$ von diesem Prozessor berechnet. Dies ändert jedoch unter der Annahme von unbegrenzt vorhandenem lokalen Speicher für jeden Prozessor nicht die Optimalität des Gitters.

Da in den wenigsten Fällen ein Programm, das einen n-dimensionalen parallelen Indexraum beschreibt, auch genau n linear unabhängige uniforme Abhängigkeiten besitzt, werden wir im folgenden diskutieren, welche Ansatzpunkte es gibt, in diesem Fall eine bestmögliche Tile-Form zu generieren. Dazu betrachten wir zunächst die Matrix $D^{\mathrm{p}} = (\vec{d_1}^{\mathrm{p}}, \ldots, \vec{d_m}^{\mathrm{p}})$, deren Spalten aus den m vorhandenen projezierten Distanzvektoren des zu partitionierenden Programms bestehen und unterscheiden zwei Fälle:

1. Es existieren $m \geq n$ Distanzvektoren, für die gilt:

   $$\mathrm{rg}(D^{\mathrm{p}}) = \dim(D^{\mathrm{par}})$$

   Ist $m = n$, ergibt sich das Gitter durch Satz 4.2.3. Ist $m > n$, muß eine Auswahl getroffen werden, welche projezierten Distanzvektoren die Spalten des Gitters bilden. Wir werden im Anschluß Kriterien für diese Wahl vorstellen.

2. Der Rang von D^{p} ist kleiner als die Dimension des parallelen Indexraums:

   $$\mathrm{rg}(D^{\mathrm{p}}) < \dim(D^{\mathrm{par}})$$

   In diesem Fall kann kein Gitter nur aus den linear unabhängigen Distanzvektoren erzeugt werden. Wir erweitern deshalb zunächst die Matrix D^{p} um die Vektoren $P_i\vec{e}_i, 1 \leq i \leq n$. Dadurch erreichen wir eine optimale Prozessorauslastung in einer nicht durch vorhandene Distanzvektoren bestimmten Dimension. Mit dieser Erweiterung geht damit dieser Fall in der ersten über.

Kriterien zur Auswahl von Distanzvektoren

1. Welche Distanzvektoren verursachen am meisten Kommunikation?
   Kurze[9] Distanzvektoren verursachen natürlich häufiger Kommunikation als lange Distanzvektoren. Darum ist es vom Aspekt der Minimierung von Kommunikation naheliegend, möglichst kurze Distanzvektoren als Spalten des Gitters zu wählen. Allerdings sollte darüber das Ziel, ein paralleles Programm zu erhalten, nicht außer acht gelassen werden. Denn durch kurze Distanzvektoren in den Spalten des Gitters verringert sich auch das Berechnungsvolumen V_{comp}, d.h. die Anzahl an Indexpunkten, die in einem Tile und damit parallel ausgeführt werden. Da die Vektoren von L ein Parallelotop beschreiben, ergibt sich das Berechnungsvolumen V_{comp} durch:

   $$V_{\text{comp}} = \text{abs}(\det(L))$$

   In der Praxis treten sogar häufig Abhängigkeiten zur vorhergehenden Iteration auf, wodurch die Wahl der kürzesten Distanzvektoren im Extremfall zu einer Tile-Form mit $V_{\text{comp}} = 1$, d.h. einem rein sequentiellen Programm, führen kann.

2. Maximale Ausnutzung der real vorhandenen Prozessoren
   Durch die maximale Ausnutzung der real vorhandenen Prozessoren erreicht man maximale Parallelität im Zielprogramm. Dieses Ziel widerspricht in Programmen mit überwiegend kurzen Abhängigkeiten dem Ziel der Minimierung von Kommunikation. Daher muß ein Kompromiß gefunden werden: man sollte die Tile-Form so wählen, daß Kommunikation bei möglichst guter Ausnutzung der real vorhandenen Prozessoren minimiert wird. Dieser Kompromiß ließe sich vermeiden, wenn wir eine Zielfunktion einführen würden, die Parameter für die Kosten je Kommunikation und die Kosten je Berechnung enthält. Diese Zielfunktion wäre jedoch unter anderem von der Vernetzung der Prozessoren abhängig, weshalb wir darauf nicht näher eingehen.

[9] Wir bestimmen die Länge eines Vektors anhand der euklidschen Norm.

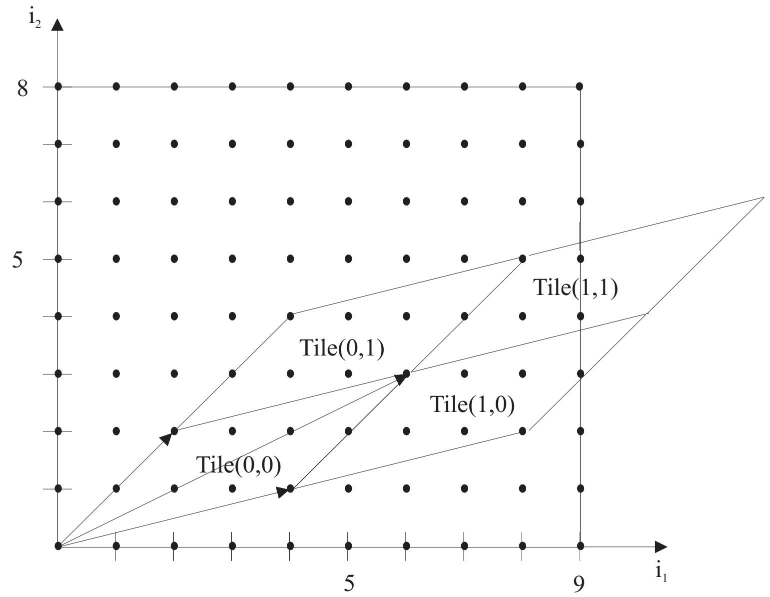

Abbildung 15: Partitionierung des parallelen Indexraums

3. Elimination von Kommunikation durch Abhängigkeiten, die nicht Spalten des Gitters sind
   Es besteht die Möglichkeit, auch Kommunikation, die durch Abhängigkeiten verursacht wird, deren Distanzvektoren nicht Spalten des Gitters L sind, bei geeigneter Wahl von L vollständig zu eliminieren. Betrachten wir dazu Abbildung 15. Die projezierten Distanzvektoren $\vec{d_1}^{\,\mathrm{p}} = (2,2)$ und $\vec{d_2}^{\,\mathrm{p}} = (4,1)$ bestimmen das Gitter

   $$L = \begin{pmatrix} 2 & 2 \\ 4 & 1 \end{pmatrix}$$

   Trotzdem wird offensichtlich auch die Kommunikation eliminiert, die durch den projezierten Distanzvektor $\vec{d_3}^{\,\mathrm{p}} = (6,3)$ entsteht, weil auch das Ziel der Abhängigkeit, z.B. der Indexpunkt $(6,3)$, auf demselben Prozessor ausgeführt wird wie die Quelle der Abhängigkeit, z.B. $(0,0)$. Allgemein erreicht man eine Elimination von Kommunikation durch derartige Abhängigkeiten, wenn der Distanzvektor dieser Abhängigkeit eine ganzzahlige Linearkombination der das Gitter beschreibenden Distanzvektoren ist. Im Beispiel gilt: $\vec{d_3}^{\,\mathrm{p}} = \vec{d_1}^{\,\mathrm{p}} + \vec{d_2}^{\,\mathrm{p}}$. Wählt man die

Spalten des Gitters also so, daß möglichst viele andere Distanzvektoren ganzzahlige Linearkombinationen der Distanzvektoren sind, die das Gitter beschreiben, wird dadurch zusätzliche Kommunikation eliminiert.
Projezierte Abhängigkeiten, die sich zu $\vec{0}$ ergeben, werden immer automatisch eliminiert, da die Positionierung der Tiles im parallelen Indexraum für alle Zeitschritte identisch ist.

Den unter 2. angesprochenen Kompromiß erreichen wir, indem wir zunächst Gitter auswählen, die aus Distanzvektoren bestehen, von denen möglichst viele andere Distanzvektoren ganzzahlig linear abhängig sind. Dadurch werden die Kriterien 1 und 3 erfüllt. Unter den ausgewählten Gittern entscheiden wir uns dann für diejenigen, die das größte Berechnungsvolumen aufweisen, wodurch wir Kriterium 2 entsprechen.

Anmerkung 4.2.5 Wenn wir anhand dieser Kriterien ein Gitter ohne Berücksichtigung der Anzahl an real vorhandenen Prozessoren konstruieren, ist es nach Anmerkung 4.2.4 nicht immer möglich, bei zu geringer Anzahl an Prozessoren das Gitter zu skalieren. Dieser Fall kann sich bei jeder Kombination von Distanzvektoren ergeben. Um im folgenden Alogrithmus auch hier ein Ergebnis zu erhalten, ergänzen wir die Distanzvektoren in jedem Fall um die Vektoren $P_i\vec{e}_i$, da damit gewährleistet ist, daß bei fehlender Möglichkeit zur Konstruktion eines optimalen Gitters zumindest eine optimale Prozessorauslastung erreicht wird.

Zusammengefaßt erhalten wir folgenden Algorithmus, der bei der automatischen Generierung einer Tile-Form im Rahmen der LPGS-Partitionierung den Zielen der Partitionierung unter Berücksichtigung der genannten Kriterien entspricht:

Algorithmus 4.2.6 (Automatische Tile-Form-Generierung)

Eingabe: Ein paralleles Programm mit perfekter Verschachtelung der parallelen Schleifen und die Anzahl an real vorhandenen Prozessoren je Dimension: $P_i, 1 \leq i \leq n$, wobei n die Dimension des parallelen Indexraums sei.

Ausgabe: Ein Gitter L, welches das Programm nach dem LPGS-Verfahren partitioniert und dabei Kommunikation minimiert.

Methode:

1. Berechne die transformierten Abhängigkeiten des Programms P. Konstruiere daraus die projezierten Distanzvektoren $\vec{d_j}^{\,\mathrm{p}}$ aller uniformer Abhängigkeiten und erstelle die Distanzmatrix D^{p}, deren Spalten die projezierten Distanzvektoren sind.
   (a) Entferne alle projezierten Distanzvektoren $\vec{0}$ aus D^{p}. (Dieser Schritt wurde lediglich aus Laufzeitgründen eingefügt, um die Anzahl der in Schritt 2 gebildeten Mengen zu reduzieren.)
   (b) Füge die n Vektoren $P_i\vec{e_i}$ zur Matrix D^{p}.
2. Bilde aus den $m \geq n$ sich ergebenden Vektoren von D^{p} die Menge $\mathcal{L}$ aller geordneten n-aus-m-Mengen L_i, $1 \leq i \leq \binom{m}{n}$:
   (a) Hat eine Menge $L_i \in \mathcal{L}$ linear abhängige Spalten, entferne sie aus $\mathcal{L}$, da ihre Vektoren keinen n-dimensionalen Raum aufspannen und damit kein Gitter ergeben.
   (b) Überprüfe für die verbleibenden $L_i \in \mathcal{L}$, ob außer den projezierten Distanzvektoren in L_i noch weitere projezierte Distanzvektoren in D^{p} existieren, die ganzzahlige Linearkombinationen der Vektoren von L_i sind. Zähle für jedes L_i die Anzahl an eliminierter Kommunikation, also die Anzahl linear abhängiger Distanzvektoren.
   (c) Bilde aus den L_i, die gemäß b) am meisten zusätzliche Kommunikation eliminieren, eine Menge $\mathcal{L}^{\mathrm{opt}}$.
3. Betrachte die Menge der L_i aus $\mathcal{L}^{\mathrm{opt}}$:
   (a) Für $|\mathcal{L}^{\mathrm{opt}}| = 1$: Sei L das Gitter, dessen Spalten sich aus den Distanzvektoren des einzigen Elements von $\mathcal{L}^{\mathrm{opt}}$ ergeben. Überprüfe, ob die Anzahl an real vorhandenen Prozessoren ausreicht.

      Falls ja: L ergibt sich als optimales Gitter $\Rightarrow$ Terminieren.

      Falls nein: Versuche L, wie in Anmerkung 4.2.4 erläutert, zu skalieren.

         Falls erfolgreich, ist das skalierte L das optimale Gitter $\Rightarrow$ Terminieren.

         Falls nicht erfolgreich: Streiche in $\mathcal{L}$ das Element von $\mathcal{L}^{\mathrm{opt}}$ und gehe zu 2b).

   (b) Für $|\mathcal{L}^{\text{opt}}| > 1$ gehe zu Schritt 4.

4. Falls $\mathcal{L}^{\text{opt}} = \emptyset$, gehe zu Schritt 2b).
   Ansonsten berechne für alle $L_i \in \mathcal{L}^{\text{opt}}$ das Berechnungsvolumen $V_{\text{comp}} = \text{abs}(\det(L_i))$. Bilde aus den L_i mit dem größten Berechnungsvolumen die Menge $\mathcal{L}^{\text{comp}}$.
   Konstruiere für jedes $L_i \in \mathcal{L}^{\text{comp}}$ wie in 3a) ein Gitter und überprüfe analog, ob die Anzahl an real vorhandenen Prozessoren jeweils ausreicht. Falls für ein L_i dies nicht zutrifft, und auch eine Skalierung erfolglos bleibt, streiche L_i aus den Mengen $\mathcal{L}^{\text{comp}}, \mathcal{L}^{\text{opt}}$ und $\mathcal{L}$.
   (a) $\mathcal{L}^{\text{comp}} = \emptyset \Rightarrow$ gehe zu Schritt 4.
   (b) $|\mathcal{L}^{\text{comp}}| = 1 \Rightarrow$ die zum einzigen Element von $\mathcal{L}^{\text{comp}}$ gehörige Matrix ergibt sich als optimales Gitter $\Rightarrow$ Terminieren.
   (c) $|\mathcal{L}^{\text{comp}}| > 1 \Rightarrow$ gehe zu Schritt 5.

5. Wähle das $L_i \in \mathcal{L}^{\text{comp}}$ als Gitter, das die vorhandenen Prozessoren am besten ausnutzt.
   Ist diese Auswahl immer noch nicht eindeutig, wähle aus den verbleibenden zufällig ein L_i, das sich als Gitter L ergibt.

Anmerkung 4.2.7 In Schritt 1a) des Algorithmus werden aus Laufzeitgründen alle projezierten Distanzvektoren entfernt, die gleich dem Nullvektor sind. Eine Elimination weiterer offensichtlich linear abhängiger Distanzvektoren (z.B. Vielfache anderer Vektoren) ist jedoch nicht sinnvoll. Existieren beispielsweise die Distanzvektoren $(3,3)$, $(6,6)$ und $(1,0)$ bei einer Prozessorenanzahl von $(3,2)$, so kann bei der Wahl des Vektors $(3,3)$ als Komponente des Gitters eine Skalierung nicht erfolgreich durchgeführt werden, während dies für den Vektor $(6,6)$ möglich ist.

Folgendes Beispiel soll Algorithmus 4.2.6 veranschaulichen:

Beispiel 4.2.8 Gegeben sei ein Programm mit zwei parallelen Schleifen mit Indizes i_1 und i_2, sowie einer sequentiellen Schleifen mit Index t. Es existieren folgende transformierte uniforme Abhängigkeiten:

$$\vec{d}_1 = \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix},\ \vec{d}_2 = \begin{pmatrix} 10 \\ 0 \\ 3 \end{pmatrix},\ \vec{d}_3 = \begin{pmatrix} 6 \\ 2 \\ 2 \end{pmatrix},\ \vec{d}_4 = \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix}$$

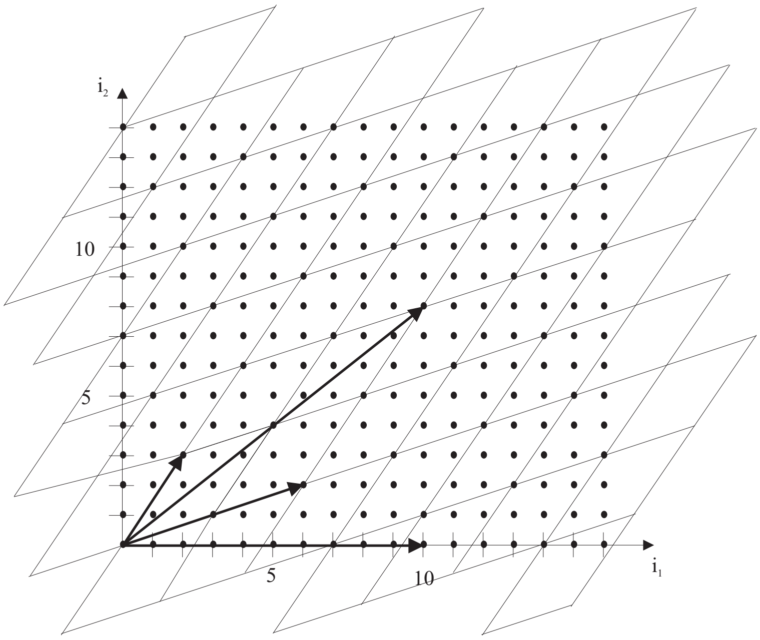

Abbildung 16: Partitionierung mit optimalem Kommunikationsaufwand für Beispiel 4.2.8

In jeder Dimension seien 5 Prozessoren vorhanden.

1. Daraus ergeben sich folgende projezierte transformierte Abhängigkeiten:

$$\vec{d_1}^{\,\mathrm{p}} = \begin{pmatrix} 2 \\ 3 \end{pmatrix},\ \vec{d_2}^{\,\mathrm{p}} = \begin{pmatrix} 10 \\ 0 \end{pmatrix},\ \vec{d_3}^{\,\mathrm{p}} = \begin{pmatrix} 6 \\ 2 \end{pmatrix},\ \vec{d_4}^{\,\mathrm{p}} = \begin{pmatrix} 10 \\ 8 \end{pmatrix}$$

(a) Es existieren keine Nullvektoren als projezierte Distanzvektoren.
(b) Somit ergibt sich folgende projezierte Distanzmatrix:

$$D^{\mathrm{p}} = \begin{pmatrix} 2 & 10 & 6 & 10 & 5 & 0 \\ 3 & 0 & 2 & 8 & 0 & 5 \end{pmatrix}$$

2. Es ergeben sich folgende 2-aus-6-Mengen:

$$L_1 = \left\{ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 10 \\ 0 \end{pmatrix} \right\} \quad L_2 = \left\{ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 6 \\ 2 \end{pmatrix} \right\} \quad L_3 = \left\{ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 10 \\ 8 \end{pmatrix} \right\}$$

$$L_4 = \left\{ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right\} \quad L_5 = \left\{ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 0 \\ 5 \end{pmatrix} \right\} \quad L_6 = \left\{ \begin{pmatrix} 10 \\ 0 \end{pmatrix} \begin{pmatrix} 6 \\ 2 \end{pmatrix} \right\}$$

$$L_7 = \left\{ \begin{pmatrix} 10 \\ 0 \end{pmatrix} \begin{pmatrix} 10 \\ 8 \end{pmatrix} \right\} \quad L_8 = \left\{ \begin{pmatrix} 10 \\ 0 \end{pmatrix} \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right\} \quad L_9 = \left\{ \begin{pmatrix} 10 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 5 \end{pmatrix} \right\}$$

$$L_{10} = \left\{ \begin{pmatrix} 6 \\ 2 \end{pmatrix} \begin{pmatrix} 10 \\ 8 \end{pmatrix} \right\} \quad L_{11} = \left\{ \begin{pmatrix} 6 \\ 2 \end{pmatrix} \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right\} \quad L_{12} = \left\{ \begin{pmatrix} 6 \\ 2 \end{pmatrix} \begin{pmatrix} 0 \\ 5 \end{pmatrix} \right\}$$

$$L_{13} = \left\{ \begin{pmatrix} 10 \\ 8 \end{pmatrix} \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right\} \quad L_{14} = \left\{ \begin{pmatrix} 10 \\ 8 \end{pmatrix} \begin{pmatrix} 0 \\ 5 \end{pmatrix} \right\} \quad L_{15} = \left\{ \begin{pmatrix} 5 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 5 \end{pmatrix} \right\}$$

(a) L_8 wird aus der Menge der L_i entfernt, da die Spalten linear abhängig sind.

(b)

L_i	zus. Elimination	#elim. Kommunikation
L_1	—	2
L_2	$\begin{pmatrix} 10 \\ 8 \end{pmatrix}$	3
L_3	$\begin{pmatrix} 6 \\ 2 \end{pmatrix}$	3
L_4	$\begin{pmatrix} 10 \\ 0 \end{pmatrix}$	2
L_5	$\begin{pmatrix} 10 \\ 0 \end{pmatrix}$	2
L_6	—	2
L_7	—	2
L_9	—	1
L_{10}	—	2
L_{11}	$\begin{pmatrix} 10 \\ 0 \end{pmatrix}$	2
L_{12}	—	1
L_{13}	$\begin{pmatrix} 10 \\ 0 \end{pmatrix}$	2
L_{14}	—	1
L_{15}	$\begin{pmatrix} 10 \\ 0 \end{pmatrix}$	1

(c) Somit ergibt sich $\mathcal{L}^{\text{opt}}$ zu:

$$\mathcal{L}^{\text{opt}} = \{L_2, L_3\}$$

3.
(a) $|\mathcal{L}^{\text{opt}}| \neq 1$
(b) $\Rightarrow$ 4.

4.

$$V_{\text{comp}}(L_2) = 14$$
$$V_{\text{comp}}(L_3) = 14$$

Somit ergibt sich $\mathcal{L}^{\text{comp}}$ zu:

$$\mathcal{L}^{\text{comp}} = \{L_2, L_3\}$$

L_2 benötigt 8 Prozessoren in Dimension 1, 5 in Dimension 2.
L_3 benötigt 12 Prozessoren in Dimension 1, 11 in Dimension2.
Da die Prozessoren sowohl für L_1 als auch für L_2 nicht ausreichen, versuchen wir die beiden möglichen Gitter zu skalieren:

- L_2 kann bezüglich des ersten Distanzvektors $(2, 3)$ nicht skaliert werden, bezüglich Vektor $(6, 2)$ ist eine Stauchung um die Hälfte möglich. Es ergibt sich:

  $$\tilde{L}_2 = \left\{ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \end{pmatrix} \right\}$$

  Für das Gitter, das sich aus $\tilde{L}_2$ ergibt, werden 5, bzw. 4 Prozessoren benötigt, d.h. durch die Skalierung erhalten wir ein mögliches Gitter, für das die vorhandenen Prozessoren ausreichen.

- L_3 kann ebenso nur bezüglich des zweiten Vektors $(10, 8)$ skaliert werden. Es ergibt sich:

  $$\tilde{L}_3 = \left\{ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 5 \\ 4 \end{pmatrix} \right\}$$

  Da aber hierbei in Dimension 1 und 2 jeweils 7 Prozessoren benötigt werden, und eine nochmalige Skalierung nicht möglich ist, wird L_3 aus $\mathcal{L}^{\text{comp}}$, $\mathcal{L}^{\text{opt}}$ und $\mathcal{L}$ entfernt.

(a) $|\mathcal{L}^{\text{comp}}| \neq \emptyset$
(b) $|\mathcal{L}^{\text{comp}}| = 1 \Rightarrow \tilde{L}_2$ ergibt sich als optimales Gitter L:

$$L = \begin{pmatrix} 2 & 3 \\ 3 & 1 \end{pmatrix}$$

Mit diesem Gitter verursacht nur noch eine von vier Abhängigkeiten Kommunikation bei nahezu optimaler Prozessorauslastung.

4.3 Partitionierung mit Anzahl Prozessoren als Parameter

Bisher gingen wir in dieser Arbeit bei der Anzahl an real vorhandenen Prozessoren stets von konkreten Werten aus. Um das partitionierte parallele Programm jedoch auf jedem Rechner, dessen Architektur ein Prozessorarray mit beliebiger Ausdehnung ist, ausführen zu können, wäre es wünschenswert, die Prozessorenanzahl bei der Partitionierung als Parameter mitzuführen. Zur Laufzeit des Programms würden dann, abhängig von der Rechnerarchitektur, den Parametern konkrete Werte zugewiesen. Dadurch erhalten wir aber in den Einträgen des Gitters Strukturparameter, was im Verfahren zum Aufzählen der Tiles gemäß Abschnitt 4.1.2 zu einer Multiplikation von Strukturparametern und Schleifenindizes führt, die laut Definition von Feautrier [8] nicht affin ist, z.B.

$$L = \begin{pmatrix} n & 0 \\ 0 & m \end{pmatrix} \Rightarrow n * t_1 + rp_1 = i_1, \quad m * t_2 + rp_2 = i_2$$

Da die in LooPo eingesetzten Verfahren zur Berechnung der Schleifengrenzen nur affine Ausdrücke im Sinne von Feautrier verarbeiten können, darf das Gitter keine Strukturparameter enthalten, um die Tiles aufzählen zu können.

Um dennoch zumindest eine rechteckige Partitionierung mit Parametern als Prozessorenanzahl zu erreichen, wurde dieser Fall durch eine primitive Partitionierung des parallelen Indexraums gelöst. Wir erreichen dies durch eine Aufteilung der i-ten parallelen Schleife in eine sequentielle Schleife, die

$$(upper_i \ \mathbf{div} \ n\_proc_i) - (lower_i \ \mathbf{div} \ n\_proc_i) + 1$$

Tiles in Dimension i aufzählt, sowie eine parallele Schleife, die den n_proc_i großen Tileinhalt in Dimension i ausführt:

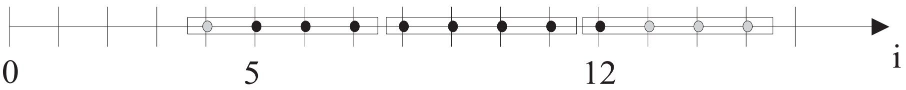

Abbildung 17: Partitionierung mit $n\_proc = 4$

$$\textbf{parfor } j := lower_i \textbf{ to } upper_i \textbf{ do}$$

Daraus erhalten wir im asynchronen Fall:

$$\begin{array}{l} \textbf{parfor } rp_i := 0 \textbf{ to } n\_proc_i - 1 \textbf{ do} \\ \quad \textbf{for } t_i := (lower_i \textbf{ div } n\_proc_i) \textbf{ to } (upper_i \textbf{ div } n\_proc_i) \textbf{ do} \\ \quad\quad j := n\_proc_i t_i + rp_i \end{array}$$

Für den synchronen Fall ergeben sich die Schleifengrenzen für t_i und rp_i analog.

Durch diese Art der Partitionierung erreicht man an den Grenzen des parallelen Indexraums keine exakte Aufzählung der Indexpunkte mehr, d.h. es werden an den Rändern eventuell zu viele Punkte aufgezählt:

$$\textbf{parfor } i := 5 \textbf{ to } 12 \textbf{ do}$$

Nehmen wir als konkreten Wert für n_proc beispielsweise 4 an. Dann ergibt sich durch die Partitionierung folgender Schleifensatz:

$$\begin{array}{c} \textbf{parfor } rp := 0 \textbf{ to } 3 \textbf{ do} \\ \textbf{for } t := 1 \textbf{ to } 3 \textbf{ do} \\ i := 4t + rp \end{array}$$

Wie aus Abbildung 17 ersichtlich wird, werden sowohl am unteren als auch am oberen Rand Punkte aufgezählt, die nicht mehr zum parallelen Indexraum gehören. Um wieder eine exakte Aufzählung des Indexraums zu erreichen, wird vor dem Statement S des partitionierten Programms die Abfrage eingefügt, ob der berechnete Punkt wirklich ein Punkt des ursprünglichen Indexraums ist:

$$\begin{array}{l} \textbf{if } (lower_1 \leq i_1 \leq upper_1) \wedge \ldots \wedge (lower_n \leq i_n \leq upper_n) \\ \qquad \{\text{Statement } S(i_1, \ldots, i_n)\} \end{array}$$

Somit wird Statement S nur ausgeführt, wenn $\vec{\imath}$ ein Punkt des ursprünglichen parallelen Indexraums ist.

Die Behandlung von Schrittweiten ungleich 1 erfolgt analog des in Abschnitt 4.1.4 vorgestellten Verfahrens.

5 LSGP-Partitionierung

Bei der LSGP-Partitionierung wird, wie bereits in Abschnitt 2.5 erläutert, der Indexraum in Tiles partitioniert, die parallel von den realen Prozessoren ausgeführt werden (**g**lobal **p**arallel), und deren Inhalte sequentiell von einem Prozessor aufgezählt werden (**l**okal **s**equentiell).

Im folgenden Abschnitt werden wir ein Verfahren [24] vorstellen, das bei gegebenem Berechnungsvolumen, d.h. der Anzahl an auszuführenden Indexpunkten je Tile, eine Tile-Form mit minimalem Kommunikationsaufwand generiert. Wie in der gängigen Literatur üblich, bezieht sich auch diese Arbeit auf perfekt verschachtelte sequentielle Programme mit ausschließlich affinen Schleifengrenzen und uniformen Abhängigkeiten. Wir werden jedoch an geeigneter Stelle auf entsprechende Modifikatonen zur Einbindung in das LooPo-Projekt eingehen.

Das Verfahren von Jingling Xue

In diesem Verfahren wird eine Tile-Form generiert, die für ein gegebenes Berechnungsvolumen ν minimalen Kommunikationsaufwand verursacht. Das in Abschnitt 4.2 eingeführte Berechnungsvolumen V_{comp} stellt im Gegensatz zu ν, das hier als Eingabeparameter betrachtet wird, das Berechnungsvolumen in Abhängigkeit von einer bestimmten Tile-Form dar.

Anmerkung 5.1 Da das Berechnungsvolumen ν ein Eingabeparameter für das vorzustellende Verfahren ist, gilt es zunächst, dieses anhand der Größe des Prozessorarrays, d.h. der Anzahl an Prozessoren je Dimension, und der Grenzen der Parallelschleifen zu bestimmen. Mittels folgender Formel ergibt sich das Berechnungsvolumen pro Dimension ν_i:

$$\nu_i = \lceil \frac{upper_i - lower_i + 1}{P_i} \rceil$$

wobei $upper_i$ bzw. $lower_i$ die obere bzw. untere Grenze der parallelen Schleife in Dimension i sei, P_i die Anzahl an realen Prozessoren in dieser Dimension. Das gesamte Berechnungsvolumen ν erhalten wir durch

$$\nu = \prod_{i=1}^{n} \nu_i$$

wobei n die Dimensionalität des Prozessorfeldes sei.

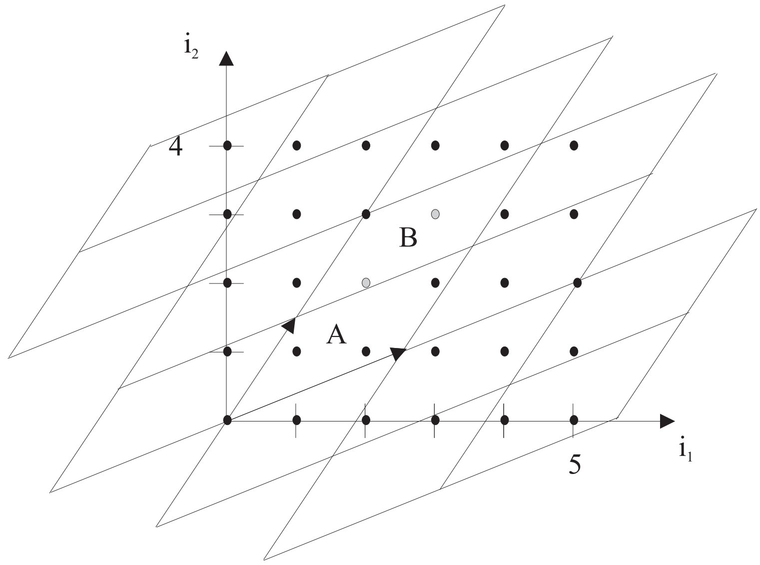

Abbildung 18: Partitionierung mit einer Tile-Form P, die rationale Einträge enthält

Die Tile-Form hat in diesem Verfahren zwei äquivalente Darstellungen:

- als Matrix P, deren Spalten die Vektoren sind, die die Seiten eines Tiles beschreiben,
- als Matrix H, deren Zeilen die Normalen-Vektoren der n Seiten eines Tiles sind.

Aus dieser Beziehung ergibt sich $P = H^{-1}$. H muß eine nicht-singuläre Matrix sein, damit ein Tile eine beschränkte Anzahl an Punkten enthält. Außerdem ist gefordert, daß P ganzzahlig ist, um zu erreichen, daß alle Tiles die gleiche Anzahl an Punkten enthalten. Betrachten wir dazu Abbildung 18. Die Tile-Form P ist durch die Matrix

$$\begin{pmatrix} 1 & \frac{5}{2} \\ \frac{3}{2} & 1 \end{pmatrix}$$

gegeben. Tile A enthält somit vier Indexpunkte, während Tile B z.B. nur zwei enthält.

Anmerkung 5.2 Die Matrix P, deren Spalten die Vektoren sind, die die Kanten der Tile-Form beschreiben, entspricht in der Notation dieser Arbeit dem Gitter L.

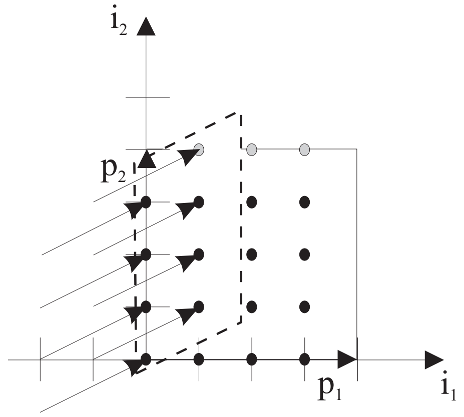

Abbildung 19: Annäherung der Anzahl an Kommunikation, die aufgrund der Abhängigkeit $\vec{d} = (2, 1)$ die Seite $\vec{p}_2$ kreuzt, durch das Volumen des Parallelotops, das sich durch $\vec{d}$ und $\vec{p}_2$ ergibt.

Anmerkung 5.3 Xue beschränkt sich bei seiner Tile-Form-Generierung auf perfekt verschachtelte Programme mit ausschließlich uniformen Abhängigkeiten; affine Abhängigkeiten dürfen nicht auftreten. Dies gilt jedoch nicht für die Anwendung des Verfahrens auf Parallelschleifen. Wie bei LPGS betrachten wir auch bei LSGP nur die projezierten uniformen Abhängigkeiten und wenden diese Methode ebenfalls nur auf den parallelen Indexraum an. Affine Abhängigkeiten können hier vernachlässigt werden, ohne die Semantik des Programms zu verändern, jedoch dadurch verursachte Kommunikation kann durch die Tile-Form nicht minimiert werden.

Das Berechnungsvolumen V_{comp} für eine gegebene Tile-Form ergibt sich durch:

$$V_{\text{comp}}(H) = \text{abs}(\det(P)) = \frac{1}{\text{abs}(\det(H))}$$

Da bei der LSGP-Partitionierung jede Abhängigkeit zwischen Tiles Kommunikation verursacht, ist das Kommunikationsvolumen V_{comm} durch die Anzahl an Abhängigkeiten, die in ein Tile hineingehen (oder aus einem Tile herausgehen) bestimmt. Seien $\vec{d}$ ein Distanzvektor und $\vec{p}_1, \ldots, \vec{p}_n$ die Vektoren, die die Seiten der Tile-Form beschreiben. Dann kann die Anzahl der durch $\vec{d}$ verursachten Abhängigkeiten, die die Seite $\vec{p}_2 \times \ldots \times \vec{p}_n$ der Tile-Form kreuzen, durch das Volumen des Parallelotops, das durch die Vektoren $\vec{d}, \vec{p}_2, \ldots, \vec{p}_n$ beschrieben wird, angenähert werden (siehe Abbildung 19). Man erhält:

$$\mathrm{abs}(\det([\vec{d}, \vec{p}_2, \ldots, \vec{p}_n])) = \frac{1}{\mathrm{abs}(\det(H))} \vec{h}_1 \vec{d}$$

Wir erreichen dadurch lediglich eine Annäherung des Kommunikationsvolumen, da nicht jedes Ziel einer Abhängigkeit, die eine Seite der Tile-Form kreuzt, im Tile liegen muß. Das gesamte Kommunikationsvolumen wird damit durch folgende Formel angenähert:

$$V_{\mathrm{comm}}(H) = \frac{1}{\mathrm{abs}(\det(H))} \sum_{i=1}^{n} \sum_{j=1}^{m} \vec{h}_i \vec{d}_j$$

Das Optimierungsproblem, eine Tile-Form mit minimalem Kommunikationsaufwand bei gegebenem Berechnungsvolumen ν zu generieren, läßt sich formal folgendermaßen darstellen:

$$\begin{array}{rl} \text{Minimiere} & V_{\mathrm{comm}}(H) = \frac{1}{\mathrm{abs}(\det(H))} \sum_{i=1}^{n} \sum_{j=1}^{m} (HD)_{i,j} \\ \text{unter der Bedingung} & V_{\mathrm{comp}}(H) = \frac{1}{\mathrm{abs}(\det(H))} = \nu \\ & HD \geq \vec{0} \end{array}$$

wobei D die Distanzmatrix ist, deren Spalten sich aus den Distanzvektoren $\vec{d}_i, 0 \leq i \leq m$, ergeben.
Die Bedingung $HD \geq \vec{0}$ entspricht der Forderung, daß vor dem Ausführen eines Tiles die gesamte Synchronisation abgeschlossen sein muß, d.h. daß alle Daten, die zur Ausführung des Tiles benötigt werden, bereits vorhanden sind [12].

Anmerkung 5.4 Die Bedingung $HD \geq \vec{0}$ kann bei der Anwendung dieses Verfahrens auf Parallelschleifen entfallen, da durch die Partitionierung des parallelen Indexraums die Synchronisation zwischen den Tiles nur in den Zeitschleifen stattfindet.

Es ergibt sich ein komplexes kombinatorisches Problem. Xue geht in seiner Arbeit zuerst auf einige Spezialfälle ein, um diese im Algorithmus laufzeit-optimal behandeln zu können. Wir wollen hier nur die Idee des allgemeinen Falles vorstellen, in dem eine vollrangige Distanzmatrix vorliegt.

Anmerkung 5.5 Besitzt die Matrix D den Rang $\rho < n$, so führt Xue eine Schleifentransformation durch, deren Ergebnis $n - \rho$ äußere Parallelschleifen und ρ innere sequentielle Schleifen sind. Die Partitionierung beschränkt sich dann auf die ρ inneren Schleifen.

Da wir uns aber mit der Partitionierung des parallelen Indexraums beschäftigen, benötigen wir hier ein anderes Vorgehen. Sei ρ der Rang der Distanzmatrix D^{p} der projezierten Abhängigkeiten. Wir führen zunächst wie Xue eine Schleifentransformation des parallelen Indexraums anhand D^{p} durch und erhalten $n - \rho$ äußere Parallelschleifen, sowie ρ innere parallele Schleifen. Da die $n - \rho$ äußeren Schleifen keine projezierten Abhängigkeiten tragen, sind sie für eine Optimierung des Kommunikationsaufwands nicht von Belang und können daher z.B. rechteckig partitioniert werden. Auf die ρ inneren Schleifen wenden wir das im folgenden vorgestellte und erweiterte Verfahren von Xue an.

Zur Herleitung dieser Methode benötigen wir zunächst nachfolgende Definition:

Definition 5.6 (endlich erzeugter Kegel) Eine nicht-leere Menge $C \subset \mathbb{R}^n$ heißt von den Vektoren $\vec{x}_1, \ldots, \vec{x}_m \in \mathbb{R}^n$ *endlich erzeugter Kegel*, wenn gilt:

$$C = \{\lambda_1 \vec{x}_1 + \ldots + \lambda_m \vec{x}_m | \lambda_i \geq 0\}$$

Notation: $C = cone(\vec{x}_1, \ldots, \vec{x}_m)$

Alle Distanzvektoren der Matrix D definieren einen Kegel, den *Abhängigkeits-Kegel* $C(D)$:

$$C(D) = cone(\vec{d}_1, \ldots, \vec{d}_m)$$

Ausgehend von der Distanzmatrix definieren wir einen weiteren Kegel, den *Tiling Kegel*, der durch die Normalenvektoren auf die Seiten des Abhängigkeits-Kegels erzeugt wird:

$$C(\vec{h}) = \{\vec{h} | \vec{h} D \geq \vec{0}\}$$

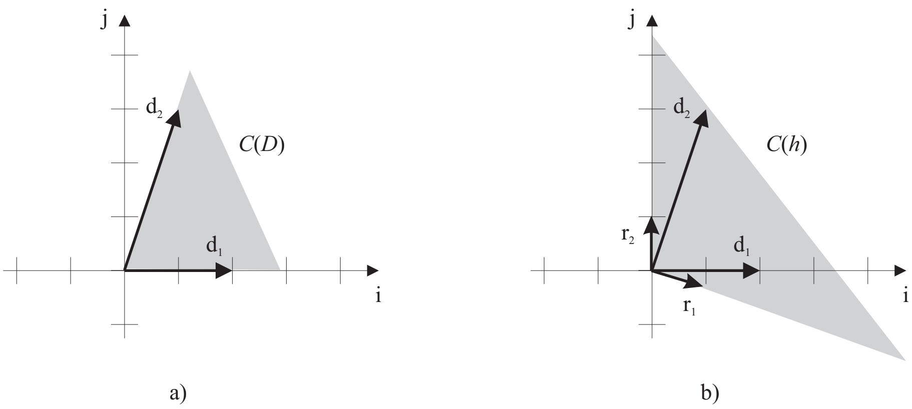

Abbildung 20: a) Abhängigkeits-Kegel und b) Tiling Kegel für die Distanzvektoren $\vec{d_1} = (1,3)$ und $\vec{d_2} = (2,0)$

wobei $\vec{h}$ ein Zeilenvektor der Matrix H sei. Diese beiden Kegel stehen in folgender Weise in Relation zueinander:

$$\begin{aligned} C(D) &= cone(\vec{d_1}, \ldots, \vec{d_m}) = \{\vec{\mu} | \vec{r}_1 \vec{\mu} \geq 0, \ldots, \vec{r}_p \vec{\mu} \geq \vec{0}\} \\ C(\vec{h}) &= \{\vec{h} | \vec{h} D \geq \vec{0}\} = cone(\vec{r}_1, \ldots, \vec{r}_p) \end{aligned}$$

$\vec{r}_1, \ldots, \vec{r}_p$ sind die Kanten, die den Tiling Kegel begrenzen. Anschaulich stellt z.B. der Vektor $\vec{r}_1$ die Senkrechte auf die von den Vektoren $\vec{d_2}, \ldots, \vec{d_m}$ erzeugte Seite des Abhängigkeits-Kegel dar (s. Abbildung 20). Auf die Konstruktion der Kegel, sowie der Kanten des Tiling Kegel soll in dieser Arbeit nicht weiter eingegangen werden, wir werden lediglich an einem Beispiel eine mögliche Methode vorstellen.

Beinhaltet die Distanzmatrix D n linear unabhängige Vektoren, die die Kanten des Abhängigkeits-Kegels darstellen, ergibt sich ein eindeutiges optimales Tiling mit einer Tile-Form, deren n Kanten parallel zu den n Kanten des Abhängigkeits-Kegels verlaufen. Im allgemeinen kann der Abhängigkeits-Kegel aber durch mehr als n Kanten beschrieben sein, wodurch sich mehrere optimale Tilings ergeben können.

In einem ersten Schritt schränkt Xue diese Menge ein, indem er folgendes

Lemma aufstellt:

Lemma 5.7 Sei die Distanzmatrix D vollrangig. Wenn H eine optimale Tile-Form ist, dann werden alle Seiten eines Tiles von derselben Anzahl an Abhängigkeiten gekreuzt. Formal ergibt sich: Sei $c(\vec{h}) = \sum_{j=1}^{m} (\vec{h}D)_j$. Wenn $\vec{h}$ ein Zeilenvektor von H ist, repräsentiert $\frac{c(\vec{h})}{\mathrm{abs}(\det(H))}$ das Kommunikationsvolumen durch eine Seite. Dann muß gelten:

$$c(\vec{h}_1) = \ldots = c(\vec{h}_n)$$

Der Beweis zu diesem Lemma stützt sich im wesentlichen auf die Aussagen über das Verhältnis von arithmetischem und geometrischem Mittel und kann in [24] nachgelesen werden.

Sei im folgenden G die Menge aller Tile-Formen, für die gilt, daß genau eine Abhängigkeit jede Kante eines Tiles kreuzt:

$$G = \left\{ \begin{bmatrix} \frac{\vec{a}_1}{c(\vec{a}_1)} \\ \vdots \\ \frac{\vec{a}_n}{c(\vec{a}_n)} \end{bmatrix} \mid \vec{a}_1, \ldots, \vec{a}_n \in C(\vec{h}) \right\}$$

Im nächsten Schritt wird ein Kriterium hergeleitet, das aussagt, welche Matrizen in G zur Erstellung optimaler Tile-Formen benutzt werden können.

Lemma 5.8 Sei D eine vollrangige Distanzmatrix. Eine optimale Tile-Form

$$H = \sqrt[n]{\frac{1}{\mathrm{abs}(\det(B))\nu}} B, B \in G$$

besitzt die größte Determinante $\mathrm{abs}(\det(B))$. Damit ergibt sich das geringste Kommunikationsvolumen

$$V_{\mathrm{comm}}(H) = n\nu \sqrt[n]{\frac{1}{\mathrm{abs}(\det(B))\nu}}$$

Anmerkung 5.9 Bei der Konstruktion der Matrizen aus G bleibt das Berechnungsvolumen $V_{\mathrm{comp}} = \nu$ unberücksichtigt. Um eine Tile-Form mit diesem Berechnungsvolumen zu erhalten, muß B mit dem Faktor $\sqrt[n]{\frac{1}{\mathrm{abs}(\det(B))\nu}}$ skaliert werden.

Die Menge G enthält unendlich viele Matrizen, doch auch diese Auswahl an optimalen Tile-Formen läßt sich noch weiter einschränken.

Lemma 5.10 Sei wieder D eine vollrangige Matrix, $B \in G$ sei so konstruiert, daß einige Zeilen keine Kanten des Tiling Kegels sind, $B' \in G$ so, daß alle Zeilen Kanten des Tiling Kegel sind. Dann gilt:

$$\mathrm{abs}(\det(B)) < \mathrm{abs}(\det(B'))$$

Mit diesem Lemma verringert sich die Menge der Matrizen aus G als mögliche optimale Tile-Formen auf die Menge G', die nur noch Matrizen enthält, deren Vektoren Kanten des Tiling Kegels sind:

Sei R die Menge aller Kanten des Tiling Kegels $C(\vec{h})$. Dann ist die Menge $G' \subseteq G$ definiert als

$$G' = \left\{ \begin{bmatrix} \frac{\vec{a}_1}{c(\vec{a}_1)} \\ \vdots \\ \frac{\vec{a}_n}{c(\vec{a}_n)} \end{bmatrix} \mid \vec{a}_1, \ldots, \vec{a}_n \in R \right\}$$

Zusammengefaßt ergeben Lemmata 5.7 bis 5.10 schließlich folgenden Satz, der beschreibt, wie sich bei gegebenem Berechnungsvolumen ν eine Tile-Form ergibt, die minimale Kommunikation erzeugt.

Satz 5.11 (Optimale Tile-Form) Sei D eine vollrangige Distanzmatrix. Eine optimale Tile-Form

$$H = \sqrt[n]{\frac{1}{\mathrm{abs}(\det(B))\nu}} B, \ B \in G'$$

hat die größte Determinante $\mathrm{abs}(\det(B))$, wodurch sich minimales Kommunikationsvolumen

$$V_{\mathrm{comm}}(H) = n\nu \sqrt[n]{\frac{1}{\mathrm{abs}(\det(B))\nu}}$$

ergibt.

Wir wollen dieses Vorgehen am folgenden Beispiel erläutern:

Beispiel 5.12 Gegeben seien folgende Distanzvektoren:

$$\vec{d_1} = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}, \vec{d_2} = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}, \vec{d_3} = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}, \vec{d_4} = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}, \vec{d_5} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

Man kann die Kanten des Tiling Kegels konstruieren, indem man für jeweils $n-1$ linear unabhängige Distanzvektoren $(\vec{x}_1, \ldots, \vec{x}_{n-1}) \in D$ die Normalenvektoren zu der Ebene, die von diesen $n-1$ Vektoren aufgespannt wird, berechnet. Eine mögliche Lösung ist:

$$\vec{r} = \vec{x}_1 \times \ldots \times \vec{x}_{n-1}$$

Wenn gilt:

- $\vec{r}\vec{d} \geq 0$ für alle $\vec{d} \in D$, dann ist $\vec{r}$ eine Kante des Tiling Kegels.
- $\vec{r}\vec{d} \leq 0$ für alle $\vec{d} \in D$, dann ist $-\vec{r}$ eine Kante des Tiling Kegels.

Im Beispiel besitzt der Tiling Kegel folgende 4 Kanten:

$$\vec{r}_1 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \vec{r}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \vec{r}_3 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \vec{r}_4 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

Als nächstes berechnen wir die Anzahl an Abhängigkeiten

$$c(\vec{r}_i) = \sum_{j=1}^{5} (\vec{r}_i D)_j$$

die jeweils durch eine Seite gehen, die von r_i beschrieben wird:

$$c(\vec{r}_1) = 3,\ c(\vec{r}_2) = 6,\ c(\vec{r}_3) = 6,\ c(\vec{r}_4) = 9$$

In der Menge G' ergeben sich damit $\binom{4}{3} = 4$ mögliche Tile-Formen:

$$G' = \{B_1, B_2, B_3, B_4\}$$

die folgendermaßen definiert sind:

$$B_1 = \begin{pmatrix} 0 & 0 & \frac{1}{3} \\ 0 & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 \end{pmatrix} \quad B_2 = \begin{pmatrix} 0 & 0 & \frac{1}{3} \\ 0 & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{9} & \frac{1}{9} & 0 \end{pmatrix}$$

$$B_3 = \begin{pmatrix} 0 & 0 & \frac{1}{3} \\ \frac{1}{6} & 0 & 0 \\ \frac{1}{9} & \frac{1}{9} & 0 \end{pmatrix} \quad B_4 = \begin{pmatrix} 0 & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 \\ \frac{1}{9} & \frac{1}{9} & 0 \end{pmatrix}$$

Berechnen wir die Determinanten der B_i:

$$\det(B_1) = -\frac{1}{108}, \quad \det(B_2) = -\frac{1}{162}, \quad \det(B_3) = \frac{1}{162}, \quad \det(B_4) = \frac{1}{324}$$

so ergibt sich B_1 mit der größten absoluten Determinante als optimale Tile-Form H, die nun noch so skaliert werden muß, daß die Anzahl der Punkte in einem Tile dem vorgegebenen Berechnungsvolumen ν entsprechen:

$$H = \sqrt[3]{\frac{108}{\nu}} B_1$$

Dieses Verfahren garantiert jedoch nicht, daß für eine optimale Tile-Form H die Forderung, daß $H^{-1} = P$ ganzzahlig ist, erfüllt ist. P wird dann ganzzahlig, wenn

- B^{-1} nur ganzzahlige Einträge hat und
- $\sqrt[n]{\text{abs}(\det(B))\nu} \in \mathbb{Z}$ ist.

Allgemein gilt:
Um H^{-1} so zu konstruieren, daß es nur ganzzahlige Einträge hat, muß das Berechnungsvolumen ν in der Menge

$$M = \left\{ \frac{x^n c^n}{\text{abs}(\det(B))} | x \in \mathbb{N}^+ \right\}$$

enthalten sein, wobei c die kleinste positive Zahl ist, so daß cB^{-1} ganzzahlig ist.

In unserem Beispiel muß gelten:

$$\nu \in \left\{ 108x^3 | x \in \mathbb{N}^+ \right\}$$

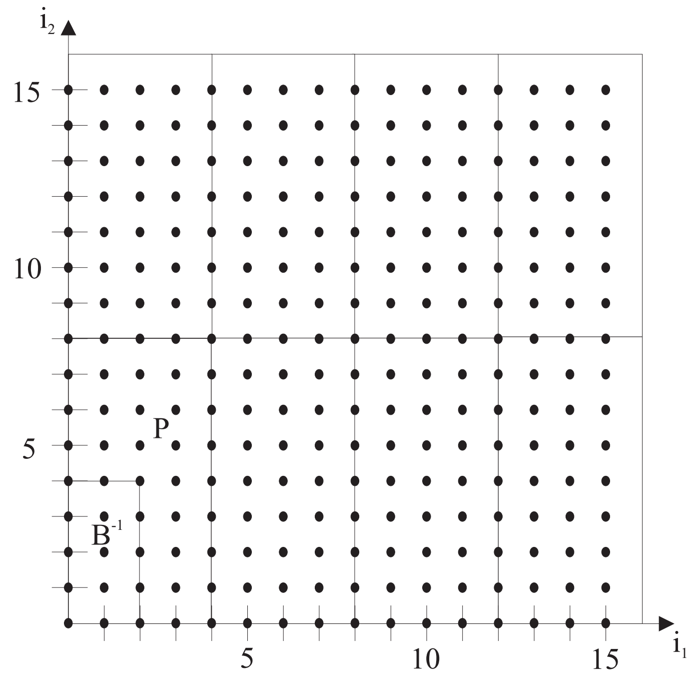

Abbildung 21: Partitionierung des Indexraums mit der Tile-Form P, die sich durch gleichmäßige Skalierung von B^{-1} in einem 4×4-Prozessorarray ergibt.

Für ein gegebenes Berechnungsvolumen ν' wählen wir also für ν:

$$\min(\nu : \nu \in M : \nu \geq \nu')$$

Anmerkung 5.13 Durch die Forderung, daß durch alle Seiten der sich ergebenden Tile-Form gleich viele Abhängigkeiten laufen müssen, um eine optimale Partitionierung zu erhalten, muß das sich ergebende optimale B in allen Dimensionen gleichmäßig skaliert werden. Dadurch haben wir aber bei der Anwendung des Verfahrens auf Parallelschleifen keinen Einfluß mehr auf die Berechnungspunkte je Dimension, d.h. bei einem Berechungsvolumen $\nu = 36$, das sich aus jeweils 6 Prozessoren in Dimension 1 und 2 ergibt, ist durch die Skalierung nicht gewährleistet, ob die Prozessoren je Dimension ausreichen. Beispielsweise wäre es auch denkbar, daß sich bei einem Berechungsvolumen von 36 Indexpunkten eine Verteilung auf 3 Prozessoren in Dimension 1 und 12 Prozessoren in Dimension 2 ergibt. Um dennoch eine Ausführung des partitionierten Programms auf einem realen Prozessorarray zu ermöglichen, gibt es zwei Möglichkeiten, die Tile-Form B zu skalieren:

- Wir skalieren B in allen Dimensionen gleichmäßig, indem wir den Wert als Skalierungsfaktor wählen, für den in allen n Dimensionen gewährleistet ist, daß die Anzahl an realen Prozessoren ausreicht. Durch diese Wahl kann sichergestellt werden, daß in mindestens einer Dimension eine optimale Prozessorauslastung erreicht wird. In den maximal $n-1$ übrigen Dimensionen kann sich jedoch eine massive Vergeudung an Prozessoren, und damit eine unnötige Sequentialisierung des partitionierten Programms ergeben. Betrachten wir dazu Abbildung 21. Seien in den Dimensionen i_1 und i_2 jeweils 4 Prozessoren vorhanden, und sei B^{-1} die zu skalierende optimale Tile-Form. Bei dieser Tile-Form würde man in Dimension i_1 8, in Dimension i_2 4 Prozessoren benötigen. Durch eine Skalierung mit dem Faktor 2 erreicht man, daß in Dimension i_1 die verfügbaren Prozessoren ausreichen. Mit der sich ergebenden Tile-Form P wird jedoch in Dimension i_2 und damit im gesamten Prozessorarray die Hälfte der vorhandenen Prozessoren vergeudet.

  Vorteil der gleichmäßigen Skalierung ist jedoch die Einhaltung der Bedingung, daß durch jede Kante der Tile-Form gleich viele Abhängigkeiten gehen müssen, um ein kommunikationsoptimales Tiling zu erreichen.

- Die andere Möglichkeit ist, die Tile-Form je Dimension zu skalieren. Dadurch werden die Prozessoren in allen Dimensionen besser ausgenutzt, die Optimalität bezüglich der Kommunikationsminimierung bleibt jedoch nicht erhalten.

Unabhängig davon, welche Art der Skalierung wir wählen, erhalten wir im Falle von Parametern als Grenzen der parallelen Schleifen in der Tile-Form P, und damit in unserer Notation im Gitter L, Parameter als Einträge. Bei der Aufzählung der Tiles ergeben sich dadurch nicht-affine Ausdrücke, für die wir im Rahmen des LooPo-Projektes keine Verfahren zum Berechnen der Schleifengrenzen haben. Da Parameter als Schleifengrenzen häufig auftreten, haben wir auf die Implementierung dieses Verfahrens zur Bestimmung einer optimalen LSGP-Tile-Form verzichtet.

6 Schlußbemerkungen

Mit den in dieser Arbeit vorgestellten Verfahren erreichen wir die Ausführung der durch LooPo automatisch parallelisierten Programme auf realen Rechnerarchitekturen, indem wir den theoretischen Ansatz bereits existierender Methoden für sequentielle Programme auf parallele erweitern und davon neue ableiten. Da uns für den Bereich LPGS in der Literatur keine Verfahren bekannt sind, haben wir in Abschnitt 4.2 einen neuen Algorithmus zur Partitionierung paralleler Programme hergeleitet. Sowohl im Falle der LPGS- als auch der LSGP-Partitionierung stellen wir Methoden vor, die eine Partitionierung mit minimalem Kommunikationsaufwand liefern. Durch die Beschränkung der Partitionierung auf den parallelen Indexraum können wir die starken Restriktionen der in der Literatur diskutierten Verfahren entschärfen, sodaß wir auch eine nicht-perfekte Verschachtelung der Eingabe-Programme, sowie affine Abhängigkeiten erlauben.

Die Implementierung der Partitionierung im LooPo-Projekt ist ein wesentlicher Bestandteil dieser Arbeit. Die auf theoretischer Seite herausgearbeiteten Ansätze zur Partitionierung von parallelen Programmen sind jedoch nicht auf von LooPo erzeugte parallele Programme beschränkt.

Problematisch erwies sich die Aufzählung der generierten Tile-Formen, da sich beim Aufstellen des Ungleichungssystems zur Bestimmung der neuen Schleifengrenzen des partitionierten Programms, wie in Abschnitt 4.1.2 vorgestellt, nicht-affine Ausdrücke ergeben, die von den bei LooPo eingesetzten Projektionsverfahren (z.B. Fourier-Elimination) nicht verarbeitet werden können. Durch mächtigere Projektionsverfahren, die auch nicht-affine Ausdrücke, sowie komplexere Operationen, wie z.B. $ceil$ und $floor$ unterstützen, würde man zum einen die Möglichkeit zur vollständigen Implementation der Aufzählung von Tile-Formen bei rationalen Schrittweiten erhalten. Zum anderen ergäbe sich damit auch die Möglichkeit der Minimierung des Kommunikationsaufwands im Falle der Partitionierung, bei der die Anzahl an Prozessoren als Parameter in das System miteinfließen (Abschnitt 4.3). Außerdem wäre damit auch eine Implementierung des in Kapitel 5 vorgestellten Verfahrens möglich, auf die aus eben genannten Gründen bisher verzichtet wurde, obwohl von theoretischer Seite die Grundlage dafür gegeben wäre.

Literatur

[1] Corinne Ancourt, Francois Irigoin. *Scanning Polyhedra with DO Loops*. Third ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, Vol. 26, No. 7, pp. 39-50, July 1991.

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

[3] P. Boulet, A. Darte, T. Risset, Y. Robert. *(Pen)-Ultimate Tiling?* INTEGRATION, the VLSI Journal, Vol. 17, pp. 33-51, 1994.

[4] Pierre-Yves Calland, Tanguy Risset. *Precise Tiling for Uniform Loop Nests*. Research Report No. 94-29, Ecole Normale Supérieure de Lyon, September 1994.

[5] H. S. M. Coxeter. *Regular Polytopes*. 3rd edition, New York: Dover, pp. 122-123, 1973.

[6] Erik H. D'Hollander. *Partitioning and Labeling of Index Sets in Do Loops with Constant Dependence Vectors*. International Conference on Parallel Processing, Vol. 2, pp. 139-144, 1989.

[7] Peter Faber. *Target Output*. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1997, in Vorbereitung.

[8] 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.

[9] Max Geigl. *Parallelization of Loop Nests with General Bounds in the Polyhedron Model*. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1997.

[10] Branko Grünbaum, G. C. Shephard, *Tiling and Patterns*. W. H. Freeman and Company, New York, 1987.

[11] Robert Günz. *The new LooPo Scanner and Parser*. Inernal Report, Fakultät für Mathematik und Informatik, Universität Passau, 1995.

[12] F. Irigoin, R. Triolet. *Supernode Partitioning*. In 15th Annual ACM Symposium on Principles of Programming Languages, pp. 319-329, San Diego, California, Jan. 1988

[13] Harald Keimer. *Datenabhängigkeitsanalyse zur Schleifenparallelisierung: Vergleich und Erweiterung zweier Ansätze*. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1997.

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

[15] 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.

[16] Wolfgang Meisl. *Practical Methods for Scheduling and Allocation in the Polytope Model*. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1996.

[17] Dan I. Moldovan, Jose A. B. Fortes. *Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays*. IEEE Transactions on Computers, Vol. c-35, No. 1, January 1986.

[18] J. Ramanujam, P. Sadayappan. *Tiling of Iteration Spaces for Multicomputers*. International Conference on Parallel Processing, Vol. 2, pp. 179-186, 1990.

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

[20] Jang-Pin Sheu, Tsu-Huei Tai. *Partitioning and Mapping Nested Loops on Multiprocessor Systems*. IEEE Transactions on Parallel and Distributed Systems, Vol. 2, No. 4, pp. 430-439, Oktober 1991.

[21] Jürgen Teich, Lothar Thiele. *Partitioning of Processor Arrays: a Piecewise Regular Approach*. INTEGRATION, the VLSI Journal 14, pp. 297-332, 1993

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

[23] Christian Wieninger. *Automatische Methoden zur Parallelisierung im Polyedermodell*. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1995.

[24] Jingling Xue. *Communication-Minimal Tiling of Uniform Dependence Loops*. Journal of Parallel and Distributed Computing, Vol. 42, pp. 42-59, 1997.

Passau, 30.09.1997

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, daß ich die vorliegende Diplomarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet habe, sowie daß diese Diplomarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt wurde.

(Martina Wieninger)