Polytop-Modell

Tim Habermaas

26. Juni 2010

Inhaltsverzeichnis

1 Einleitung

Bei Prozessoren wurde die Steigerung der Leistungsfähigkeit bisher unter anderem durch mehr Transistoren und Erhöhung der Taktfrequenz erreicht. Doch diese Art der Leistungssteigerung hat den Nachteil der größeren Abwärme, welche nicht mehr handbar ist. Dadurch ist man dazu übergegangen einem Prozessor mehrere Kerne zu spendieren. Diese Mehrkernprozessoren haben bei gleichbleibender Taktfrequenz theoretisch eine n-fache Leistungssteigerung bei n Kernen. Um möglichst nahe an diese theoretische Grenze zu kommen ist es wichtig, dass die Software gut parallelisiert ist.

Um Software zu parallelisieren werden grob zwei Ansätze unterschieden. Einmal gibt es die Möglichkeit Software explizit zu parallelisieren (z.B. durch Schlüsselwörter in Programmiersprachen), andererseits lässt sie sich mit Hilfe des Compilers auch automatisch parallelisieren. Der Vorteil des automatischen Verfahrens ist dabei, dass

der Programmierer sich kaum Gedanken über die Parallelität machen muss und – was mindestens genauso wichtig ist – dass alte Software nicht umgeschrieben werden muss um sie parallel auszuführen. Es genügt dabei den alten Code einfach mit einem neuen Compiler zu kompilieren.

Das Polytop-Modell setzt bei dem automatischen Verfahren an und dient dem Parallisieren von Schleifen. Dabei besteht die Grundidee beim Polytop-Modell darin das vorliegende Schleifenprogramm als Polytop (Abschnitt 2) aufzufassen. Auf diesem Polytop lassen sich nun Transformationen ausführen, die dafür sorgen, dass das resultierende Schleifenprogramm parallelisierbar ist. Die Transformationen beschreiben dabei Basiswechsel der Indexvariablen und dienen dazu das Programm neu zu ordnen, so dass Datenabhängigkeiten erhalten bleiben und möglichst viele Iterationen der Schleife parallel ausgeführt werden können.

2 Polytop

Ein *Polytop* ist ein verallgemeinertes Polygon in beliebiger Dimension. Insbesondere lassen sich konvexe Polytope als Ungleichungssystem der folgenden Form darstellen: $A * \vec{x} \geq \vec{b}$. Dabei ist A eine $n \times m$-Matrix. Da jede Zeile dieses Ungleichungssystems einen Halbraum beschreibt, besteht das Polytop somit aus n Halbräumen und hat die Dimension m. Für $m = 2$ bzw. $m = 3$ nennt man das Polytop auch *Polygon* bzw. Polyeder.

Im zweidimensionalen lässt sich zum Beispiel das konvexe Polygon aus Abbildung 1 mit Hilfe von folgenden fünf Halbräumen definieren:

$$\begin{aligned} x &\geq 0 \\ x &\leq 4 \\ y &\geq 0 \\ y &\leq 5 \\ x - y &\geq -5 \end{aligned}$$

Beschreibt man das Polytop in der Form $A * \vec{x} \geq \vec{b}$ ergibt sich:

$$A * \vec{i} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 1 & -1 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \geq \begin{bmatrix} 0 \\ -4 \\ 0 \\ -5 \\ -5 \end{bmatrix}$$

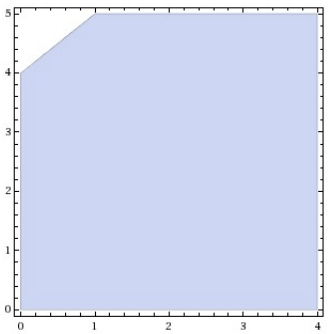


Abbildung 1: Schnitt von fünf Halbräumen ergibt konvexes Polytop

3 Quellprogramm

Als Quellprogramm wird das Schleifenprogramm bezeichnet, welches mittels des Polytop-Modells parallelisiert werden soll. Um ein Schleifenprogramm mittels eines Polytops zu beschreiben, muss es einige Bedingungen erfüllen [5]:

- Als Statements des Quellprogramms sind nur for-Schleifen und Zuweisungen erlaubt.
- Als Datentypen sind nur Arrays und Skalare erlaubt.
- Sowohl Schleifengrenzen, als auch Array-Indizes dürfen nur affine Funktionen in den Indizes umgebener Schleifen sein. Zusätzlich sind noch konstante Parameter in den affinen Funktionen erlaubt. Die unteren Schleifengrenzen dürfen außerdem durch eine Maximums-Funktion und die oberen durch eine Minimums-Funktion beschrieben werden.

Affine Funktionen in den Schleifengrenzen sind notwendig, damit das Schleifenprogramm als konvexes Polytop beschrieben werden kann. Affine Funktionen in den Array-Indizes sind für die Berechnung von Schedule und Allokation notwendig (Abschnitt 5.1).

3.1 Begriffe

Zunächst werde ich erst einmal die Begriffe *Indexraum*, *Indexvektor* und *Operation* erklären. Als Grundlage für die Erläuterung dient dieses Schleifenprogramm:

for $i_1 = u_1$ to o_1 **do**
  for $i_2 = u_2$ to o_2 **do**
    ...
    for $i_n = u_n$ to o_n **do**
      S
    end for
  end for
  S'
end for

3.1.1 Indexraum und Indexvektor

Der Indexraum wird in [5] folgendermaßen definiert:

Hat S die n umgebenden Schleifenindizes $i_1, \ldots, i_n$ und ist u_j bzw. o_j die untere bzw. obere Grenze des Index i_j, dann gilt:

$$\begin{aligned} u_1 \leq \quad & i_1 \quad \leq o_1 \\ u_2 \leq \quad & i_2 \quad \leq o_2 \\ u_3 \leq \quad & i_3 \quad \leq o_3 \\ & \ldots \\ u_n \leq \quad & i_n \quad \leq o_n \end{aligned}$$

alle i_j bilden dabei den *Indexraum*. Dieser Indexraum lässt sich auch kurz schreiben als:

$$D_S := \{(i_1, \ldots, i_n) : (i_1, \ldots, i_n) \in \mathbb{Z}^n \wedge (\forall j : 1 \leq j \leq n : u_j \leq i_j \leq o_j)\}$$

Ein Element $\vec{i} = (i_1, i_2, \ldots, i_n) \in D_S$ heißt dann *Indexvektor* von S.

3.1.2 Operation

Eine Operation bezeichnet ein Statement S an der Stelle $\vec{i} = (i_1, i_2, \ldots, i_n)$. Siehe auch Abbildung 2.

3.2 Beispiel

Ein Schleifenprogramm, das alle Bedingungen erfüllt, ist zum Beispiel folgendes:

for $i = 0$ to 5 **do**
  for $j = 0$ to 4 **do**
    $A[i, j] = A[i - 1, j] + 10 * A[i, j - 1]$
  end for
end for

4 Quellpolytop

Aus dem Quellprogramm lässt sich direkt das Quellpolytop ableiten, indem man den Indexraum betrachtet und ihn als Polytop in der Form $A * \vec{x} \geq \vec{b}$ beschreibt.

Beispiel Der Indexraum zu dem Schleifenprogramm aus Abschnitt 3.2 ergibt sich dann zu:

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} * \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 0 \\ -5 \\ 0 \\ -4 \end{bmatrix}$$

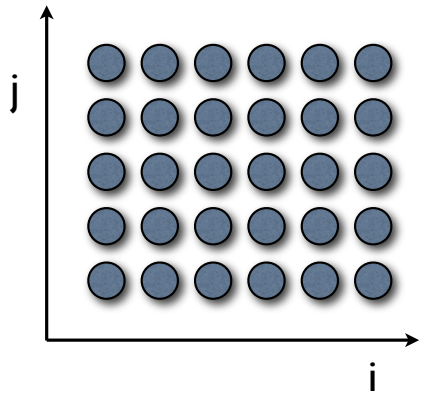


Abbildung 2: Der Indexraum zu Beispiel Abschnitt 3.2. Jeder Kreis steht dabei für eine Operation (Abschnitt 3.1.2).

5 Zielpolytop

Nun besteht die Aufgabe darin, das Quellpolytop so zu transformieren, dass das resultierende Programm parallelisierbar ist. Um das zu erreichen, muss das Polytop durch eine Matrixmultiplikation verschoben werden ohne die Datenabhängigkeiten zu verletzen.

Um die Transformation zu beschreiben, werden die zwei Funktionen Schedule und Allokation benötigt, die – wie auch schon die Schleifeninidizes – nur affine Funktionen sein dürfen.

5.1 Schedule

Die Funktion, die jeder Operation Ω einen festen Zeitpunkt zuordnet, wird Schedule genannt [3]:

$$t : \Omega \to \mathbb{Z}$$

Dabei ist ein Schedule genau dann gültig, wenn er die Datenabhängigkeiten E erhält:

$$\forall \vec{i}, \vec{j} : \vec{i}, \vec{j} \in \Omega \wedge (\vec{i}, \vec{j}) \in E : t(\vec{i}) < t(\vec{j})$$

Damit ist gewährleistet, dass eine Operation B nicht vor einer anderen Operation A ausgeführt wird, falls B von A abhängt.

5.2 Allokation

Während der Schedule die zeitliche Komponente der Transformation darstellt, kann man sich die Allokation die räumliche Komponente vorstellen. Dabei weist sie einer Operation einen bestimmten Prozessor zu [3]:

$$a : \Omega \to \mathbb{Z}^r$$

Die Zielmenge dieser Funktion kann auch mehrdimensional sein ($r > 1$). Dabei beschreibt $\mathbb{Z}^r$ dann einen Prozessor-Cluster.

5.3 Transformation

Die eigentliche Transformation des Polytops $A * \vec{i} \geq \vec{b}$ lässt sich nun mit Hilfe der Koeffizienten der zwei Funktionen t und a berechnen. Die Transformationsmatrix sieht folgendermaßen aus:

$$T = \begin{bmatrix} \vec{\lambda} \\ \vec{\sigma} \end{bmatrix}$$

wobei $t(S, \vec{i}) = \vec{\lambda} * \vec{i} + \vec{c}$ und $a(S, \vec{i}) = \vec{\sigma} * \vec{i} + \vec{d}$ die zwei affinen Funktionen sind.

Man gewinnt die neuen Indizes $\vec{j}$ des Polytops durch $\vec{j} = T * \vec{i}$. Die neuen Grenzen des Polytops berechnen sich dann mit Hilfe von

$$A * (T^{-1} * \vec{j}) \geq \vec{b}$$

Die inverse Matrix T^{-1} muss allerdings nicht immer existieren. Wie man mit solchen singulären Matrix umgeht, beschreibt das Verfahren aus [3]. Martin Griebl und sein Team lösen das Problem, indem sie zuerst die linear abhängigen Zeilen der Matrix eliminieren und danach die Matrix wieder zu einer quadratischen Matrix mit vollem Rang ergänzen. Die eliminierten Zeilen werden später wieder mittels Code in das Zielprogramm eingefügt.

5.4 Beispiel

Für das Beispiel aus Abschnitt 3.2 ergibt sich dann der Schedule $t(S, \vec{i}) = i + j$ und die Allokation $a(S, \vec{i}) = i$, was zu der Transformationsmatrix

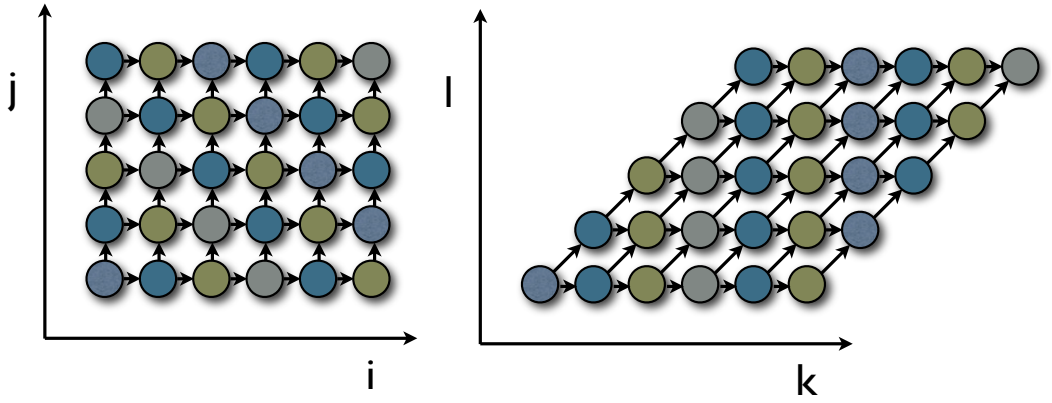
$$T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

und den neuen Indizes

$$\begin{bmatrix} k \\ l \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i+j \\ i \end{bmatrix}$$

führt.

Der Effekt, den die Transformation auf das Polytop hat, ist auf den folgenden zwei Abbildungen zu sehen.



(a) Abhängigkeitsgraph und Indexraum (b) Verschobenes Polytop, welches die Abhängigkeiten erhält

Abbildung 3: Transformation des Polytops

6 Zielprogramm

Nachdem man nun das transformierte Polytop gewonnen hat, muss es wieder zurück in ein gültiges Schleifenprogramm gewandelt werden. Für das Beispiel aus Abschnitt 3.2 erhält man dabei folgendes Polytop:

$$\begin{bmatrix} l \\ -l \\ k-l \\ l-k \end{bmatrix} \geq \begin{bmatrix} 0 \\ -5 \\ 0 \\ -4 \end{bmatrix}$$

Will man nun aus diesem Polytop wieder ein Schleifenprogramm gewinnen, muss man l und k als Schleifenindizes auffassen und versuchen passende Schleifengrenzen zu finden. Dies gelingt bei diesem Beispiel aber nicht. Löst man die Ungleichungen nämlich nach l bzw. k auf, erhält man für die letzten beiden Zeilen einen Ausdruck in Abhängigkeit von k bzw. l. Somit lassen sich keine äußeren Schleifengrenzen finden.

Fourier-Motzkin-Elimination Dieses Problem lässt sich mit Hilfe der *Fourier-Motzkin-Elimination* lösen. Diese elimiert aus einem Ungleichungssystem einzelne Variablen und man erhält so ein neues Ungleichungssystem, welches nicht mehr von diesen Variablen abhängig ist [1]. Dabei werden die j Ungleichungen nach der zu eliminierenden Variable x_j aufgelöst und es entstehen drei Gruppen von Unglei-

chungen:

$$\begin{aligned} x_j &\geq \sum_{i=1}^{j-1} a_i * x_i \ (Gruppe\ A) \\ x_j &\leq \sum_{i=1}^{j-1} a_i * x_i \ (Gruppe\ B) \\ x_j & \quad\ \ spielt\ keine\ Rolle\ (\phi) \end{aligned}$$

Das ursprüngliche Gleichungssystem ist somit äquivalent zu: $max(A_1, \ldots, A_{n_A}) \leq x_j \leq min(B_1, \ldots, B_{n_B}) \wedge \phi$, wobei n_A bzw. n_B die Anzahl der Ungleichungen in der Gruppe A bzw. B angeben und A_i bzw. B_i die rechte Seite einer Ungleichung ist.

Wendet man die Fourier-Motzkin-Elimination auf unsere Ungleichung von oben an und löst nach l auf, erhält man

$$\begin{aligned} & max(0, k-4) \leq l \leq min(k, 5) \\ \Longrightarrow\ & 0 \leq k \wedge k \leq 9 \end{aligned}$$

Damit lässt sich k nun unabhängig von l beschreiben und wir können das resultierende Schleifenprogramm hinschreiben

for $k = 0$ to 9 **do**
  for $l = max(0, k-4)$ to $min(5, k)$ **do**
    $i = l$
    $j = k - l$
    $A[i, j] = A[i-1, j] + 10 * A[i, j-1]$
  end for
end for

wobei die innere Schleife nun parallel ausgeführt werden kann.

7 Schedule und Allokation

Beim Berechnen von Schedule (Abschnitt 5.1) bzw. Allokation (Abschnitt 5.2) unterscheidet man grundsätzlich zwischen dem *statementbasierten* und dem *schleifenbasierten* Ansatz. Wie der Name schon sagt berechnet der statementbasierte Ansatz für jedes Statement eine eigene Transformationsmatrix, wohingegen der schleifenbasierte Ansatz diese Matrix für die komplette Schleife berechnet.

7.1 Schleifenbasiert

Der schleifenbasierte Ansatz betrachtet eine Schleife als ganzes Element und berechnet dafür einen Schedule und eine Allokation. Aus diesem Grund darf das Quellprogramm nur aus perfekt verschachtelten Schleifen bestehen.

for $i_1 = u_1$ to o_1 **do**
  for $i_2 = u_2$ to o_2 **do**
    ...
    for $i_n = u_n$ to o_n **do**
      S
    end for
  end for
end for

Abbildung 4: Aufbau einer perfekt verschachtelten Schleife

In [5] stellt Christian Wieninger die *Hyperebenen-Methode von Lamport* vor, welche eine Transformationsmatrix zu einer gegebenen Schleife und den Abhängigkeiten der Statements berechnet. Die Hyperebenen-Methode ist nicht zwangsläufig optimal und funktioniert im Groben dadurch, dass versucht wird eine senkrechte Hyperebene zu den einzelnen Abhängigkeitsvektoren zu finden.

7.2 Statementbasiert

Bezüglich des statementbasierten Ansatzes gibt es dagegen keine Einschränkungen an das Quellprogramm. Für jedes Statement lässt sich ein Optimierungsproblem aufstellen und somit erhält man auch für jedes Statement eine eigene Transformationsmatrix. Die Herleitung dieses Optimierungsproblems ist sehr gut auf den Seiten 31 bis 41 von [5] beschrieben. Diese statementbasierte Methode ist im Gegensatz zu der schleifenbasierten optimal bezüglich der Anzahl der parallelisierten Operationen. Sie benötigt allerdings auch deutlich mehr Rechenaufwand.

7.2.1 Zusammenfügen von Schleifen

Dadurch, dass jedes Statement separat behandelt wird, erhält man am Ende auch mehrere einzelne Zielprogramme. Um diese wieder zu einem Schleifenprogramm zusammenzusetzen, stellt [3] zwei mögliche Ansätze vor. Zum einen können die Schleifenprogramme während der *Laufzeit* zusammengesetzt werden – das heißt man fügt Bedingungen in das Zielprogramm ein, welche überprüfen, ob die Operation Teil des Polytops ist.

Auf der anderen Seite gibt es den Ansatz die Schleifenprogramme während der *Kompilierzeit* zu verbinden. Grob gesprochen werden alle Permutationen der einzelnen Schleifengrenzen berechnet und für jede Permutation eine eigene Schleife generiert. Je nach Eingabeprogramm kann dadurch allerdings das Zielprogramm sehr groß werden und auch die Kompilierzeit steigt sehr stark an [4].

8 Fazit

Das Polytop-Modell, wie wir es hier kennen gelernt haben, stellt eine einfache und elegante Art dar, Schleifen automatisch zu parallelisieren. Durch das Polytop aus der Geometrie hat man eine solide mathematische Grundlage und kann durch einfache Basistransformationen neue parallele Schleifen gewinnen. Allerdings hat diese Eleganz auch den Preis, dass es sehr starke Einschränkungen (Abschnitt 3) an die Quellprogramme geben muss, damit dieses auch als Polytop beschrieben werden können. Durch diese Einschränkungen ist das Polytop-Modell grundsätzlich nur für numerische Algorithmen bzw. Programme sinnvoll einzusetzen.

Wer sich näher mit dem Polytop-Modell beschäftigen will, dem empfiehlt sich übrigens ein Blick auf *LooPo*([2]) zu werfen.

Literatur

[1] `http://en.wikipedia.org/wiki/Fourier_Motzkin_elimination`.

[2] `http://www.infosun.fim.uni-passau.de/cl/loopo/`.

[3] Martin Griebl, Christian Lengauer, and Sabine Wetzel. Code generation in the polytope model. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'98)*, pages 106–111. IEEE Computer Society Press, 1998.

[4] Sabine Wetzel. Automatic Code Generation in the Polytope Model, 1995.

[5] Christian Wieninger. Automatische Methoden zur Parallelisierung im Polyedermodell, 1995.