Das Polyedermodell zur automatischen Schleifenparallelisierung

Christian Lengauer

Prof. Peter Faber, Prof. Sergei Gorlatch, Priv.-Doz. Martin Griebl,

Dr. Armin Größlinger, Dr. Christoph A. Herrmann,

Dipl.-Inf. Andreas Simbürger, Tobias Grosser B.Sc.

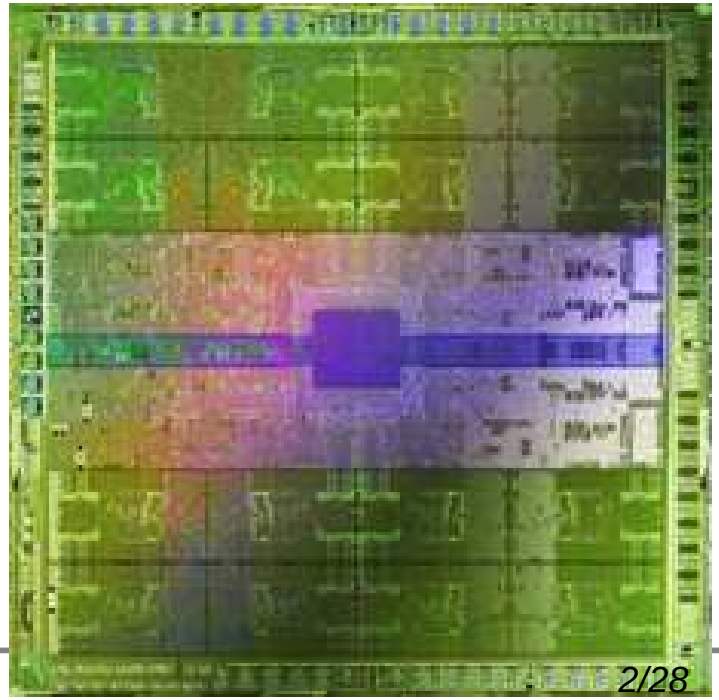
Dr. Jean-François Collard, Prof. Paul Feautrier

InvasIC-Vortrag, Universität Erlangen-Nürnberg, 3. März 2011

Massive Parallelität am Wendepunkt

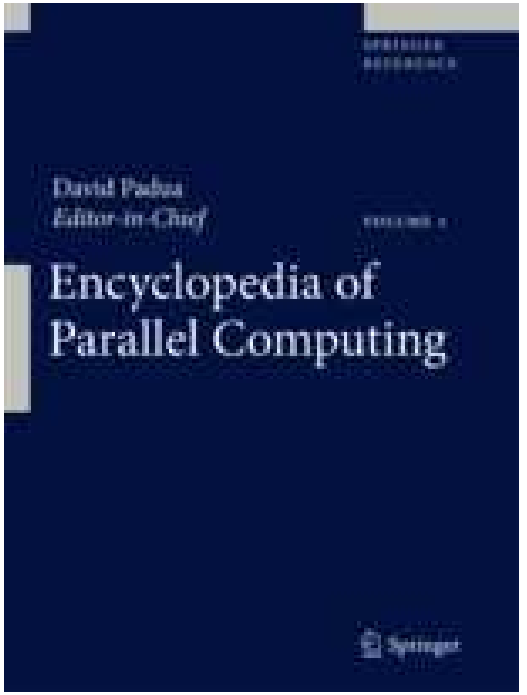
- Die Vergangenheit: Massive Parallelität war...
  - ein Nischenthema
  - teuer und kaum verbreitet in Hardware
  - eingeschränkt in Software
  - von Spezialisten per Hand programmiert und optimiert
  - auf einer Assembler-ähnlichen Abstraktionsebene
  - kaum leistungsportabel

- Die Zukunft: Massive Parallelität wird (muss!) werden...
  - flächendeckend verfügbar
  - billig in Hardware
  - divers in Software
  - Nichtexperten zugänglich und von ihnen zwangsweise genutzt
  - auf diversen Problem-näheren Abstraktionsebenen und mit Werkzeug- und Laufzeitunterstützung
  - leistungsportabler



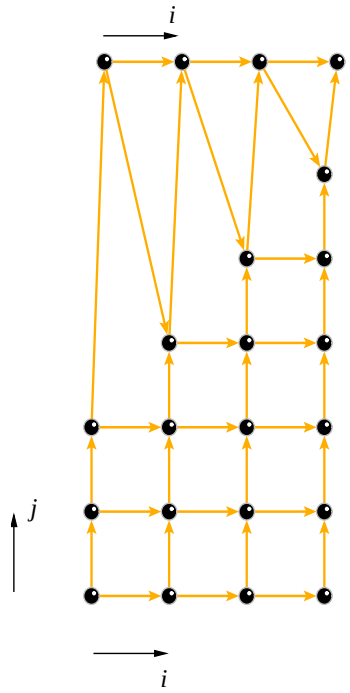
Das Polyedermodell

- Das Polyedermodell ist ca. 25 Jahre alt
- Die Entwicklung begann mit einem sehr eingeschränkten Basismodell
- Sie verlief in zwei Richtungen:
  - Weiterentwicklung der theoretischen Grundlagen des Basismodells
  - Erweiterung des Basismodells
- Abriss des Vortrags:
  - Skizze des Basismodells
  - Skizze von sieben Erweiterungen
  - Ausblick auf weitere Erweiterungen
- Lösungssuche in einer modellgerichteten Parallelisierung:
  - \+ “random-access”: alle Lösungen sind gleich schwer erreichbar
  - \+ optimierend: findet das Optimum bezüglich einer Kostenfunktion
  - \+ vollautomatisch
  - – Analyse und Zielcode möglicherweise komplex
  - – Optimalität im Modell garantiert nicht effizienten Zielcode
- Referenz: Paul Feautrier and Christian Lengauer. The Polyhedron Model. In David Padua et al., editors, Encyclopedia of Parallel Computing. Springer-Verlag, Juni 2011.



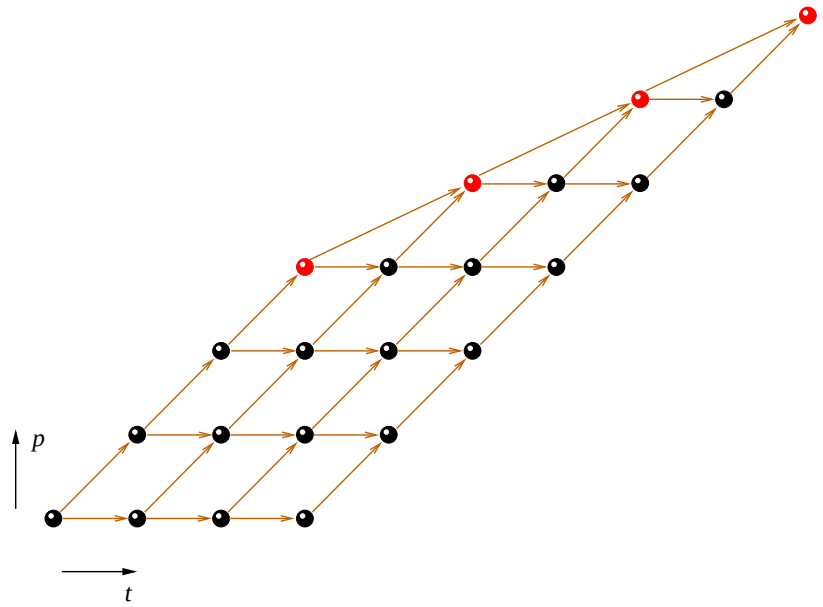
Ein erster Eindruck

for $i = 1$ to n do
  for $j = 0$ to $i + m$ do
    $A(i, j) = A(i-1, j) + A(i, j-1)$
  od;
  $A(i, i+m+1) = A(i-1, i+m) + A(i, i+m)$
od



Quellpolyeder

for $t = 0$ to $m + 2 * n - 1$ do
  parfor $p = \max(0, t-n+1)$ to $\min(t, \lceil (t+m)/2 \rceil)$ do
    if $2 * p = t + m + 1$ then
      $A(p-m, p+1) = A(p-m-1, p) + A(p-m, p)$
    else
      $A(t-p+1, p+1) = A(t-p, p+1) + A(t-p+1, p)$
    fi
  od
od



Zielpolyeder

Das Basismodell

- Anforderungen an den Quellcode:
  - Ein (möglicherweise nicht perfekt) geschachtelter Schleifensatz
  - Schleifenrumpf: Folge von Zuweisungen
  - Variablen: Feldelemente oder Skalare
  - In den Zählvariablen der umgebendenen Schleifen affin-lineare Schleifengrenzen
  - In den Schleifenvariablen affin-lineare Feldindizes
  - Strukturparameter sind an Stellen von Konstanten erlaubt
  - Unterprogrammaufrufe werden als atomar angesehen und nicht parallelisiert
  - Keine verzeigerten Strukturen, nur Felder
  - Keine Objektorientierung
- Leistungen des Modells:
  - Vollautomatische Abhängigkeitsanalyse
  - Optimierende Suche nach einer besten Lösung im Lösungsraum des Modells, bezogen auf eine Optimierungsfunktion
  - Beispiele für Optimierungsfunktionen:
    - minimale Schrittzahl plus minimale Prozessorzahl
    - minimale Schrittzahl plus maximaler Durchsatz
    - minimale Zahl von Kommunikationen
- Herausforderung: effizienter Zielcode

Das Basismodell

- Optionen für Zielschleifensätze:
  - synchron (äußere Schleifen sequenziell)
  - asynchron (äußere Schleifen parallel)
- Nutzung:
  - Parallelisierung
  - Speicheroptimierung
- Referenzen:

Christian Lengauer. Loop parallelization in the polytope model. In Eike Best, editor, *CONCUR'93*, LNCS 715, pages 398–416. Springer-Verlag, 1993.

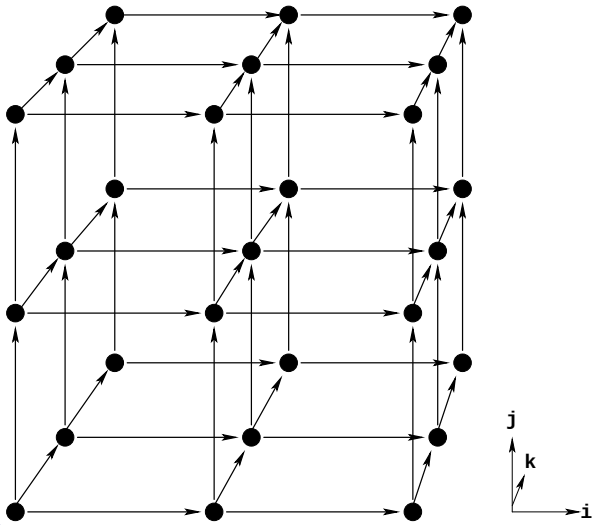
Paul Feautrier. Automatic parallelization in the polytope model. In Guy-René Perrin and Alain Darte, editors, *The Data Parallel Programming Model*, LNCS 1132, pages 79–103. Springer-Verlag, 1996.

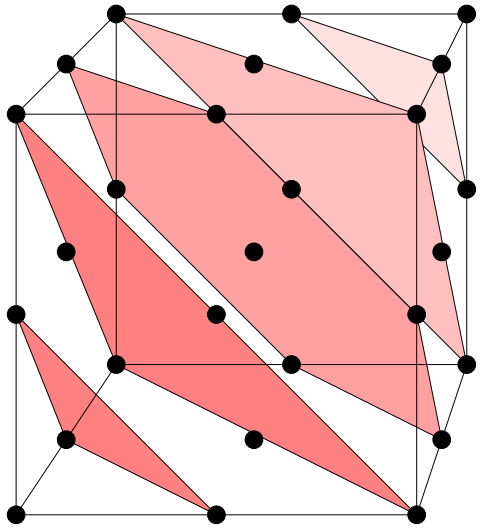
- Standardbeispiel: Produkt quadratischer Matrizen

```
for i := 0 to n − 1 do
    for j := 0 to n − 1 do
        for k := 0 to n − 1 do
            C(i, j) := C(i, j) + A(i, k) ∗ B(k, j)
        od
    od
od
```

Beispiel: Produkt quadratischer Matrizen

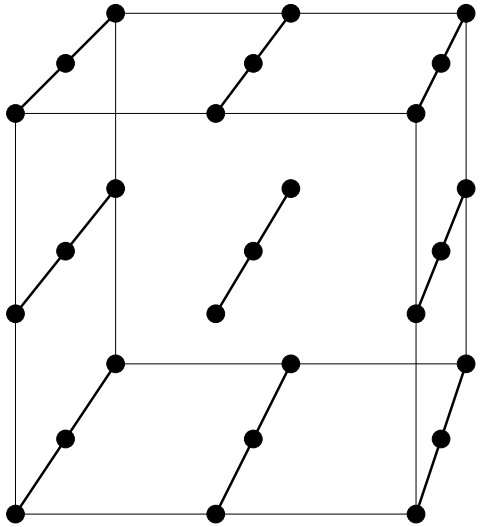
Indexraum,
Abhängigkeiten

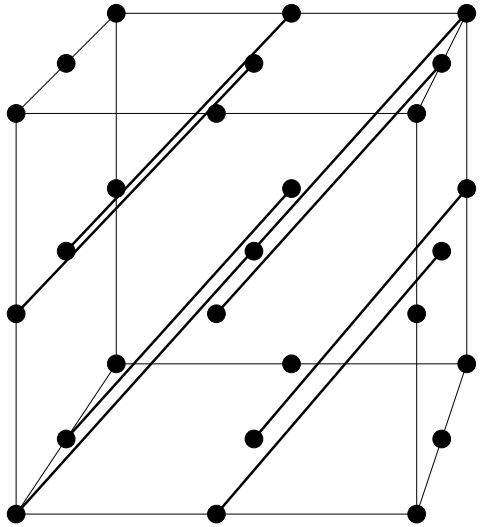




parallele
Schritte

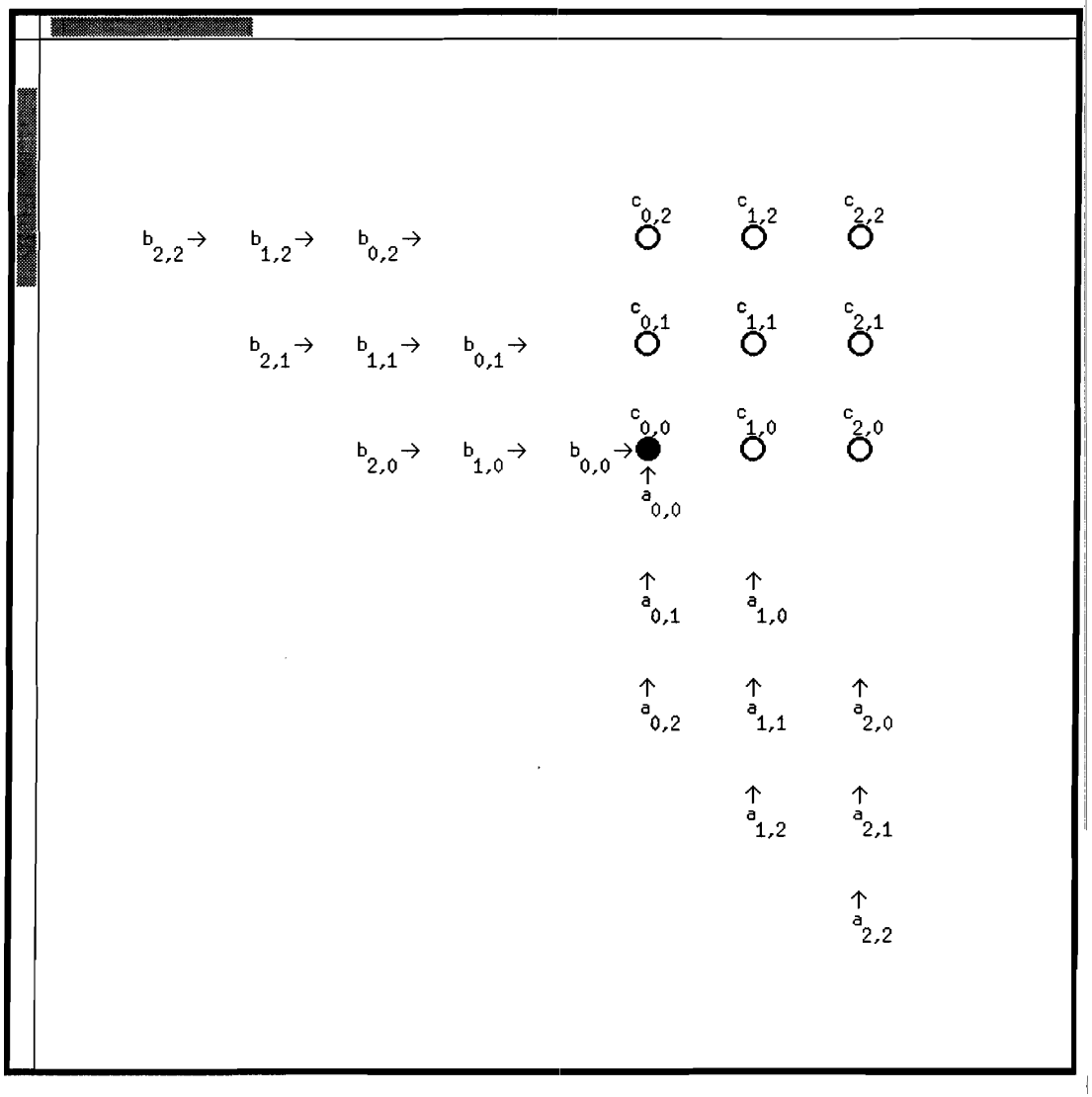
quadratisches
Prozessorfeld



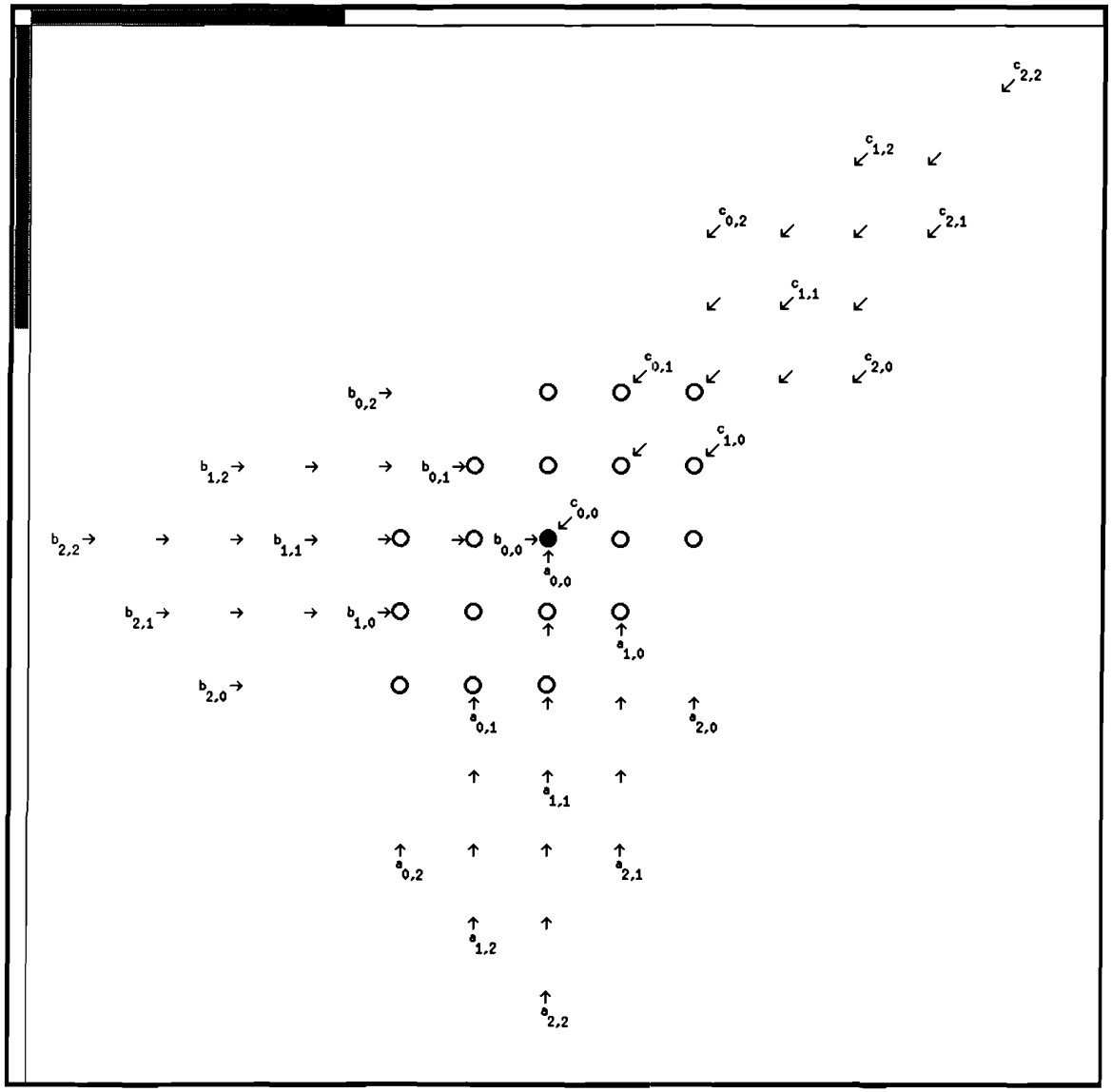


hexagonales
Prozessorfeld

Quadratische Lösung



Hexagonale Lösung



Erweiterung 1: Fallunterscheidungen im Schleifenrumpf

- Konsequenz: [Jean-François Collard, Martin Griebl]
  - Abhängigkeiten können von bedingt variieren.

```
real A[0 .. 2*N + 1]
for i := 0 to N do
    for j := 0 to N do
        A[i + j + 1] := ...
        if cond then
            A[i + j] := ...
        fi;
        ... := A[i + j]
    od
od
```

$cond$ wahr

Erweiterung 1: Fallunterscheidungen im Schleifenrumpf

- Konsequenz: [Jean-François Collard, Martin Griebl]
  - Abhängigkeiten können von bedingt variieren.

```
real A[0 .. 2*N + 1]
for i := 0 to N do
    for j := 0 to N do
        A[i + j + 1] := ...
        if cond then
            A[i + j] := ...
        fi;
        ... := A[i + j]
    od
od
```

$cond$ unwahr

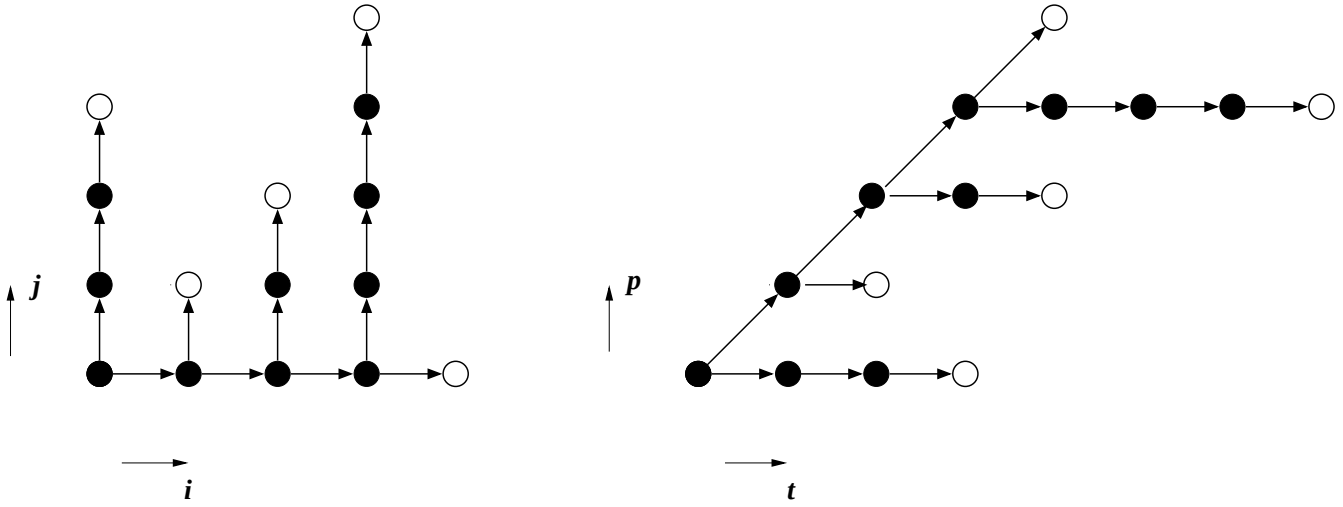
Erweiterung 1: Fallunterscheidungen im Schleifenrumpf

- Methode:
  - Eine präzise Reaching-Definition-Analyse, die folgendes kombiniert:
    - die rückwärtig iterative, klassische Lösung von Datenflussgleichungen
      (erkennt Abhängigkeiten zwischen ganzen Feldern,
      kann Fallunterscheidungen behandeln)
    - lineare Integerprogrammierung à la Polyedermodell
      (erkennt Abhängigkeiten zwischen einzelnen Feldelementen)
  - Versieht Abhängigkeiten mit Bedingungen.
  - Berechnet die Vereinigung aller Abhängigkeiten.
  - Name: Control flow fuzzy array dependence analysis (CfFADA)
  - Referenz: Jean-François Collard and Martin Griebl. A precise fixpoint reaching definition analysis for arrays. In Larry Carter and Jean Ferrante, editors, *Languages and Compilers for Parallel Computing (LCPC'99)*, LNCS 1863, pages 286–302. Springer-Verlag, 1999.

Erweiterung 2: WHILE-Schleifen im Schleifensatz

- Konsequenzen: [Jean-François Collard, Martin Griebl]
  - In WHILE-Dimensionen steht die Anzahl der Schritte erst zur Laufzeit fest.
  - Der statische Indexraum ist kein Polytop, sondern ein Polyeder.
  - Der dynamische Indexraum ist in WHILE-Richtung uneben (ein “Kamm”).

```
for i := 0 while cond1(i) do
    for j := 0 while cond2(i, j) do
        body(i, j)
    od
od
```

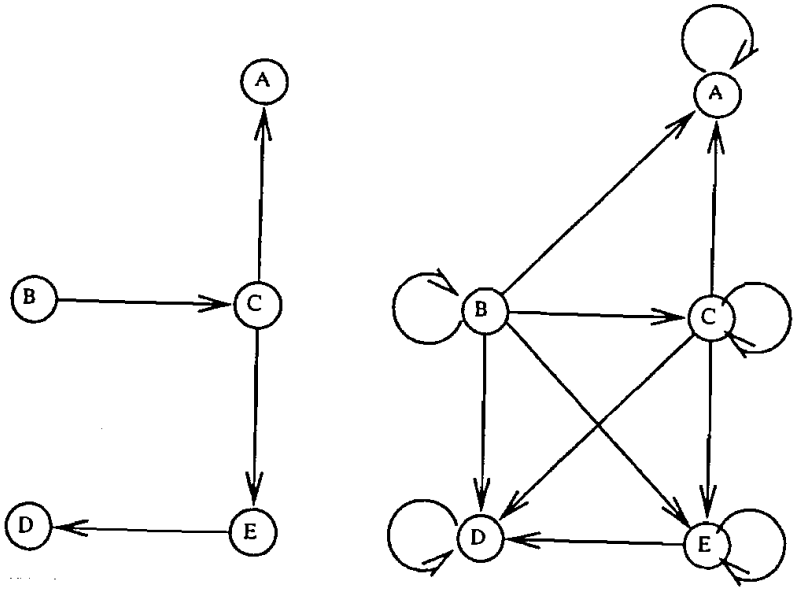


Erweiterung 2: Zwei Ansätze

- Konservativ: [Martin Griebl]
  - Die Kontrollabhängigkeit der WHILE-Schleife wird berücksichtigt.
  - Ein einzelnes WHILE bleibt sequenziell, kann aber verteilt ablaufen.
  - Ein Satz von WHILE-Schleifen kann parallel ablaufen.
  - Herausforderung: Globale Termination (Gelöst für gemeinsamen und verteilten Speicher.)
  - Referenz: Martin Griebl. *The Mechanical Parallelization of Loop Nests Containing* while *Loops*. Dissertation, Universität Passau, 1996. Technical Report MIP-9701
- Spekulativ: [Jean-François Collard]
  - Die Kontrollabhängigkeit einer außen liegenden WHILE-Schleife wird ignoriert.
  - Das WHILE kann parallel ablaufen.
  - Zusätzlicher Speicherbedarf ist möglich.
  - Ein Rollback von Schleifenschritten kann notwendig werden.
  - Herausforderungen:
    - Implementierung von Rollbacks
    - Minimierung von Rollbacks
    - Minimierung des Speicherbedarfs
  - Referenz: Jean-François Collard. Automatic parallelization of while-loops using speculative execution. *Int. J. Parallel Programming*, 23(2):191–219, 1995

Beispiel: Reflexive transitive Hülle; die Datenstruktur

n	*node*	*nrsuc*	*suc*	*rt*
0	A	0		A
1	B	1	C	B, C, A, E, D
2	C	2	A, E	C, A, E, D
3	D	0		D
4	E	1	D	E, D



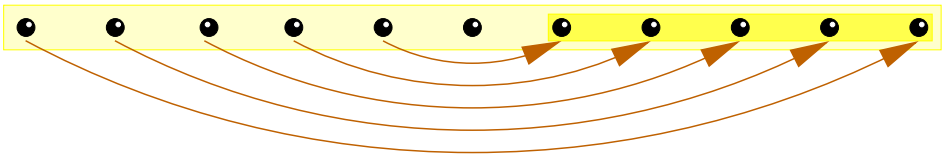
Beispiel: Reflexive transitive Hülle; das Quellprogramm

```
for n := 0 while node[n] ≠ ⊥ do
    rt[n, 0] := n;
    nxt[n] := 1;
    for d := 0 while rt[n, d] ≠ ⊥ do
        if ¬tag[n[rt[n, d]] then
            tag[n, rt[n, d]] := true
            for s := 0 to nrsuc[rt[n, d]] − 1 do
                rt[n, nxt[n] + s] := suc[rt[n, d], s]
            od
            nxt[n] := nxt[n] + nrsuc[rt[n, d]]
        fi
    od
od
```

Parallelisierung: lineare Schrittzahl

Erweiterung 3: Index Set Splitting

- Idee: [Martin Griebl, Paul Feautrier]
  - Partitioniere den Indexraum automatisch mit dem Ziel, ein Abhängigkeitsmuster zu zerlegen und die Parallelität zu erhöhen.



for $i := 0$ to $2 * n - 1$ do
  $A(i) := \ldots A(2*n - i - 1)$
od

$\Longrightarrow$

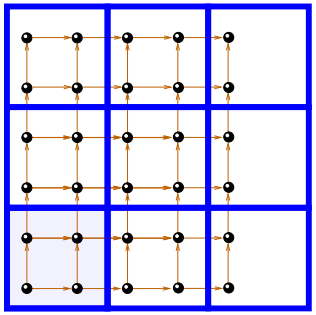
for $i := 0$ to $n - 1$ do
  $A(i) := \ldots A(2*n - i - 1)$
od;
for $i := n$ to $2*n - 1$ do
  $A(i) := \ldots A(2*n - i - 1)$
od

Schrittfunktion: $\lfloor i/2 \rfloor$ (linear)

Schrittfunktion: $\lfloor i/n \rfloor$ (konstant)

- Methode:
  - Trenne die Senken des Graphen vom Rest.
  - Propagiere die Trennungen rückwärts durch den Graphen
  - Herausforderung: Termination bei Zyklen (Schrittgrenze)
  - Referenz: Martin Griebl, Paul Feautrier, and Christian Lengauer. Index set splitting. *Int. J. Parallel Programming*, 28(6):607–631, 2000.

Erweiterung 4: Kacheln (*Tiling*)



- Goal: Bestimmte optimale Granularität der Parallelität [Martin Griebl]
  - Wie? (Form und Größe der Kacheln)
  - Wann? (Vor oder nach der Parallelisierung)
  - Was? (Raum oder Zeit)
- Wann: Nach der Parallelisierung
  - Allgemeiner:
    - Flexible Raumzeitabbildung vor inflexiblem Kacheln.
    - In den Raumdimensionen ist jede Kachelung ist erlaubt.
  - Einfacher: Ein einziger, perfekter Zielschleifensatz.
  - Einheitlicher: eine Kachelform für das gesamte Koordinatensystem.
- Was:
  - Raum: Anpassung an Betriebsmittel (Anzahl der Prozessoren)
  - Zeit: Anpassung an Performanz (Verhältnis Berechnung/Kommunikation)
- Referenzen: zum Kacheln nach der Raumzeitabbildung

Martin Griebl, Peter Faber, and Christian Lengauer. Space-time mapping and tiling: A helpful combination. *Concurrency and Computation: Practice and Experience*, 16(3):221–246, 2004.

U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. PLUTO: A practical and fully automatic polyhedral program optimization system. *Proc. ACM SIGPLAN 2008 Conf. on Programming Language Design and Implementation (PLDI 2008)*, ACM Press, 2008.

Erweiterung 5: Ausdrücke

- Ziel: Vermeide wiederholte Berechnungen [Peter Faber]
- Methode: Schleifengetragene Codeplatzierung (*Loop-carried code placement*)
  - Identifiziert Ausdrücke, die denselben Wert haben.
  - Bestimmt optimalen Zeitpunkt und Platz für die Auswertung.
  - Bestimmt optimalen Platz für das Ergebnis.
- Example: Flachwassersimulation

```
FORALL (j=1:n,i=1:m) H(i,j) =
&   P(i,j) + 0.25 * (U(i+1,j)*U(i+1,j) + U(i,j)*U(i,j)
&                 + V(i,j+1)*V(i,j+1) + V(i,j)*V(i,j))
```

↓

```
FORALL (j=1:n,i=1:m+1) TMP1(i,j) = U(i,j)*U(i,j)
FORALL (j=1:n+1,i=1:m) TMP2(i,j) = V(i,j)*V(i,j)
FORALL (j=1:n,i=1:m) H(i,j) =
&   P(i,j) + 0.25 * (TMP1(i+1,j) + TMP1(i,j)
&                 + TMP2(i,j+1) + TMP2(i,j)
```

- Referenz: Peter Faber. *Code Optimization in the Polyhedron Model – Improving the Efficiency of Parallel Loop Nests*. Dissertation, Universität Passau, lulu.com, 2008.

Erweiterung 6: Nicht-affine Feldindexausdrücke

- Ziel: Behandlung von Ausdrücken der Form `A(p*i)` [Armin Größlinger]
- “Parameter” p:
  - Hat unbekannten, festen Wert.
  - Typischer Fall: Ausdehnung des Polyeders in einer festen Dimension.
- Anwendung: Wähle Zeile oder Spalte einer Matrix als Vektor
- Herausforderung: Abhängigkeitsanalyse
  - Liegen die Lösungen innerhalb oder außerhalb des Iterationsraums?
  - In welche Richtung weist die Abhängigkeit? Parametrisches Vorzeichen möglich!
- Methode:
  - Mathematisches Modell: ganzzahlige Quasipolynome
    (Polynome, deren Koeffizienten periodische Funktionen sind).
  - Löse die Konfliktgleichungen; Koeffizienten rational, Funktionswerte garantiert ganzzahlig.
  - Es gibt einen Algorithmus für genau einen Parameter.
- Referenzen:

Armin Größlinger and Stefan Schuster. On computing solutions of linear diophantine equations with one non-linear parameter. In Proc. 10th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2008), 69–76. IEEE Computer Society, September 2008.

Armin Größlinger. *The Challenges of Non-linear Parameters and Variables in Automatic Loop Parallelization*. Dissertation, Universität Passau, lulu.com, 2009.

Erweiterung 7: Nicht-affine Schleifengrenzen

[Armin Größlinger]

- Ziel: Aufzählung von Domänen mit Grenzen, die keine Geraden sind
  - Grenzen müssen mit Polynomen beschreibbar sein.
  - Domänen sind semi-algebraische Mengen
    (Lösungsmengen von Ungleichungssystemen von Polynomen;
    Algorithmus löst in $\mathbb{R}^n$, dann Schnitt mit $\mathbb{Z}^n$).
- Beispiel: Innere Schleife des Siebs des Eratosthenes

```
for (j = i*i; j <= n; j += i)
```

  - Quadratische Grenze und variable Schrittweite
  - Transformation der variablen Schrittweite:
    - Schleifenkopf:

```
for (j = 0; j <= n; j += i)
```

      $\rightarrow$

```
for (k = 0; k*i <= n; k++)
```

      Schleifenrumpf: `j` $\rightarrow$ `k*i`
  - Nicht-lineare Schleifentransformationen:
    - Nicht-lineare Schedules können erheblich performanter sein als lineare.
- Herausforderung:
  - Codevereinfachung

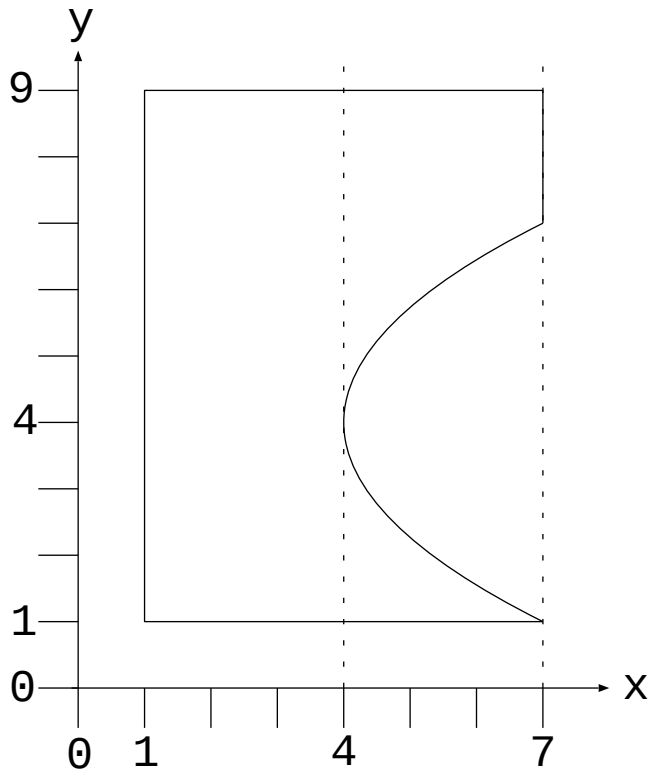
Erweiterung 7: Fälle und Methoden

- Nicht-lineare Parameter: z.B. p^2`*i`, `p*q*i`, `p*i`
  - LP-Lösungsmethoden wie Fourier-Motzkin und Simplex können auf die Behandlung mehrerer nicht-linearer Parameter erweitert werden.
  - Auswahl nach parametrischem Vorzeichen mit Quantorenelimination (in $\mathbb{R}$).
  - Anwendung: parametrisches Kacheln und Codegenerierung.
- Auch nicht-lineare Variablen: z.B. p^2`*i`^{2}, `p*i`^{2}, `i*j`
  - Anwendung: Codegenerierung zur Aufzählung beliebiger semi-algebraischer Mengen.
  - Methode: Zylindrische algebraische Dekomposition.
- Referenzen:

Armin Größlinger, Martin Griebl, and Christian Lengauer. Quantifier elimination in automatic loop parallelization. *Journal of Symbolic Computation*, 41(11):1206–1221, November 2006.

Armin Größlinger. *The Challenges of Non-linear Parameters and Variables in Automatic Loop Parallelization*. Dissertation, Universität Passau, lulu.com, 2009.

Erweiterung 7: Beispiel




```
for (x=1; x<=4; x++)
  for (y=1; y<=9; y++)
    T1(x,y);
for (x=5; x<=7; x++) {
  for (y=1; y<=⌊4−√(3x − 12)⌋; y++)
    T1(x,y);
  for (y=⌈4+√(3x − 12)⌉; y<=9; y++)
    T1(x,y);
}
```

Die Zukunft des Polyedermodells

Gegenwärtiger Stand und Ausblick

- Prototypen, die das Polyedermodell bereitstellen:
  - GRAPHITE (gcc), Polly
- Prototypen, die das Polyedermodell implementieren:
  - LooPo (Passau), PLUTO (Ohio-State)
- Software zur Abhängigkeitsanalyse:
  - Parametric Integer Programming (PIP), Omega
- Bibliotheken für Polyederoperationen:
  - Polylib, Parma Polyhedral Library, Barvinok Library, Integer Set Library (ISL)
- Software zur Codegenerierung:
  - Chunky Loop Generator (CLooG)
- Gegenwärtig engagieren wir uns in der Verfolgung zweier Ziele:
  - Polly: Akzeptanz von weit mehr Kontrollstrukturen im Quellcode
  - PolyJIT: Umgang mit Nicht-Affinität durch Nutzung von Laufzeitinformation

Polly

- LLVM: [Tobias Grosser]
  - Ein Open-Source Compiler-Framework
  - Zielsprache: LLVM IR (intermediate representation)
  - LLVM IR ist unabhängig von der Quellsprache und der Zielplattform
  - LLVM-Compiler für viele Quellsprachen: FORTRAN, C, C++, Java, Haskell
- Idee:
  - Static Control Part (SCoP): Polyeder-gerechte Kontrollstruktur
  - Extrahiere SCoPs in LLVM IR Code, nicht in Quellcode
- Zusätzlich behandelbar:
  - Programme, die sich wie eine reguläre for-Schleife verhalten
  - Ausdrücke, die sich affin-linear verhalten
  - Funktionen mit bekannten, behandelbaren Nebenwirkungen

Beispiele

```
#define N 64
int A[1024];
int i = 0;
int b, c;
do {
   int b = 2*i;
   int c = b*3 + 5*i;
   A[c] = i; i += 2;
} while (i < N);
```

$\Longrightarrow$

```
#define N 64
int A[1024];
for (int i = 0; i < N; i += 2)
   A[11*i] = i;
```

```
int A[1024];
int *B = A;
while (B < &A[1024]) {
   *B = 1; ++B;
}
```

$\Longrightarrow$

```
int A[1024];
for (int i = 0; i < 1024; i++)
   A[i] = 1;
```

PolyJIT

[Armin Größlinger, Andreas Simbürger]

- Idee:
  - Nutzung freier Kerne zur polyedrischen Analyse und Optimierung
  - Nutzung von Laufzeitinformation zur Behandlung statisch nicht oder schwer behandelbarer Situationen (JIT)
- Ziele:
  - Nutzung der Laufzeitwerte der Strukturparameter:
    - Macht ein parametrisches zu einem nicht-parametrischen Problem
    - Multiversionierung für dominierende Strukturwerte
    - Maschinelles Lernen zur “Interpolation” zwischen Strukturwerten
  - Nachtarieren einer statischen Lösungswahl durch maschinelles Lernen
  - Einsatz von Spekulation? Ein sehr schwieriges Thema...
    - Collards WHILE-Ansatz
    - Intel Itanium
    - Transaktionsspeicher

    Ist das praktikabel...?

Das Potenzial von PolyJIT

[Andreas Simbürger]

LLVM Test-Suite (Top 12 von über 500)

Test	SCoPs	non-affine	side-effect
loop_unroll	324	722	256
loop_invariant	168	303	168
constant_folding	168	283	168
timberwolfmc	152	2601	162
simulator	78	198	8
hexxagon	67	149	2
football	32	334	14
bc	25	681	33
unix-tbl	20	362	36
stepanov_abstraction	18	163	17
bullet	18	2019	32
assembler	16	207	2

Abschluss

- Zwei konkurrierende Ansätze: statisch und dynamisch
- Statisch:
  - Das Ziel: alles mit Analyse zu verstehen
  - Die Probleme:
    - alle Einflüsse zu berücksichtigen
    - die rechnerische Komplexität
- Dynamisch:
  - Das Ziel: alle Einflüsse in der Findung des Optimums zu berücksichtigen
  - Das Problem: aus dem Einzelergebnis ein allgemeineres Verständnis zu gewinnen
- Unser Ansatz: im Polyedermodell
  - statische Analyse, wo sie praktikabel ist
  - dynamische Entscheidungen, wo sie nicht praktikabel ist
- Fazit:
  - Das Polyedermodell ist ein Element der Softwaretechnologie im Manycore-Zeitalter.
  - Viele andere Elemente sind notwendig.
  - Wir brauchen eine breitflächige Entwicklung neuer Softwaretechnologie.