Parallelisierung

Christian Lengauer

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

Dr. Armin Größlinger, Dr. Christoph A. Herrmann

Dr. Jean-François Collard, Prof. Paul Feautrier

D-CON Treffen, Universität Bamberg, 5. März 2010

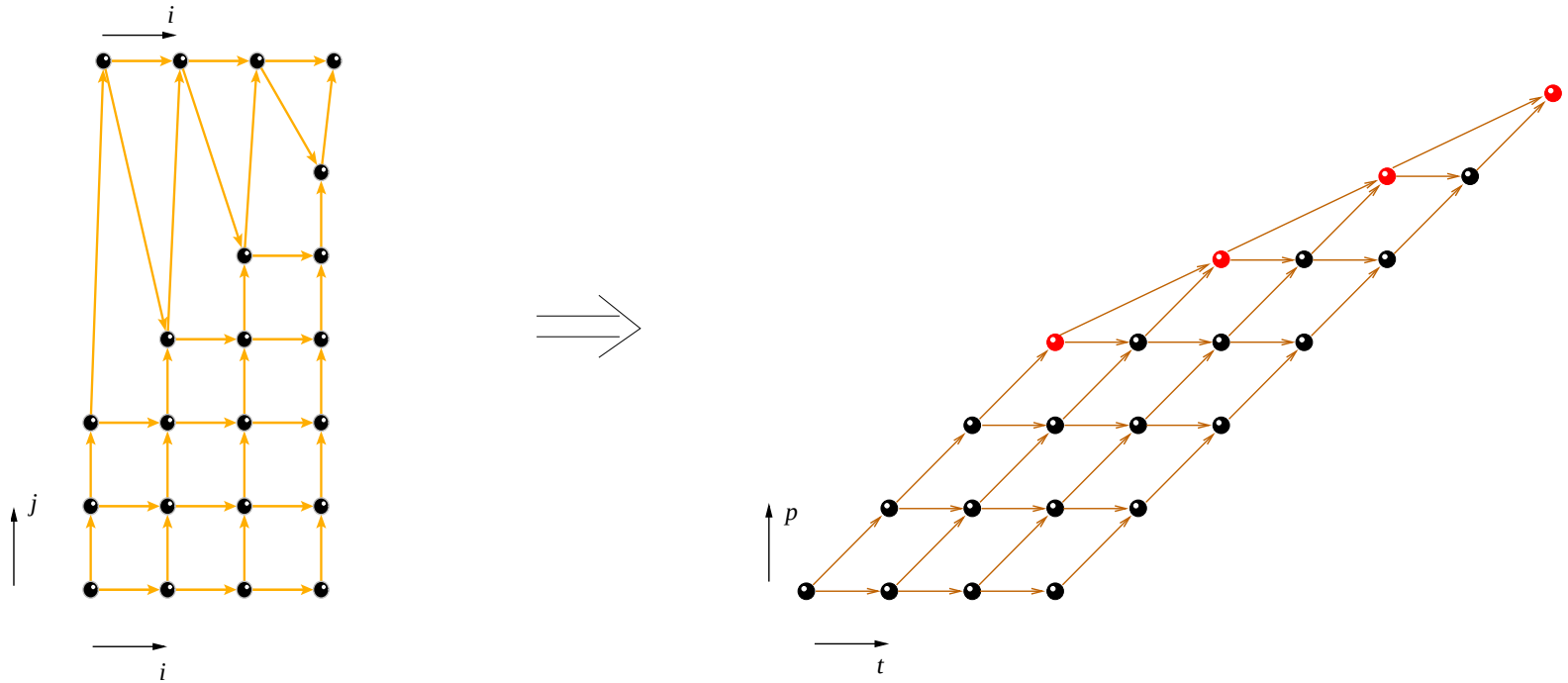
Schleifentransformationen

- Syntaxgerichtet:
  - Spaltung, Fusion, Permutation, Umkehr, Verschiebung, Aufrollen, etc.
  - \+ einfache Anwendung
  - \+ in einfachen Situationen oft ausreichend
  - – keine Ermittlung der besten Lösung
  - – bessere Lösung oft schwerer umzusetzen
- Modellgerichtet:
  - bilde Quellcode auf ein Berechnungsmodell ab
  - finde die optimale parallele Lösung in diesem Modell
  - \+ Qualitätsmetrik: eine vorgegebene Optimierungsfunktion
  - \+ Suche und Transformation vollautomatisch
  - – Analyse und Zielcode möglicherweise komplex
  - – Optimalität im Modell garantiert nicht effizienten Zielcode

Das klassische Polyedermodell

for $i = 1$ to n do
  for $j = 1$ 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

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



Quellabhängigkeitsgraph

Zielabhängigkeitsgraph

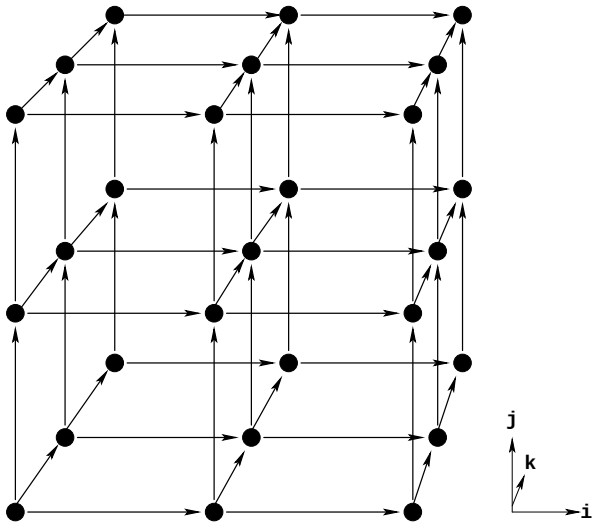
Leistungsfähigkeit des elementaren Modells

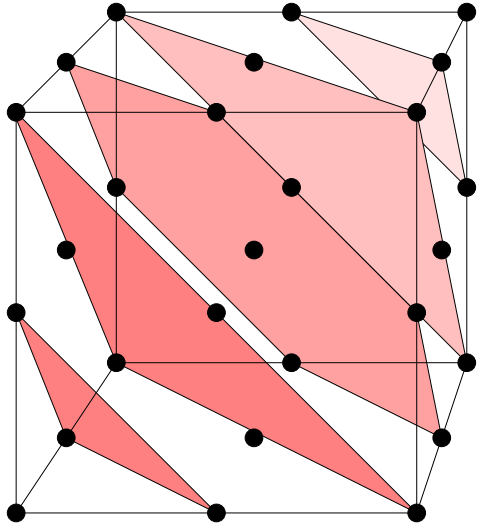
- Vollautomatische Abhängigkeitsanalyse
- Optimierende Suche nach einer besten Lösung im Lösungsraum des Modells, bezogen auf eine Optimierungsfunktion
- Beispiele für Optimierungsfunktionen:
  - minimale Schrittzahl plus minimale Prozessorzahl
  - minimale Schrittzahl plus maximaler Durchsatz
  - minimale Zahl von Kommunikationen
- Herausforderung: effizienter Zielcode
- Standardbeispiel: Produkt quadratischer Matrizen
  - Quellcode:

for $i = \mathbf{0}$ to $n - \mathbf{1}$ do
  for $j = \mathbf{0}$ to $n - \mathbf{1}$ do
    for $k = \mathbf{0}$ to $n - \mathbf{1}$ do
      $C(i,j) = C(i,j) + A(i,k) * B(k,j)$
    od
  od
od

Beispiel: Produkt quadratischer Matrizen

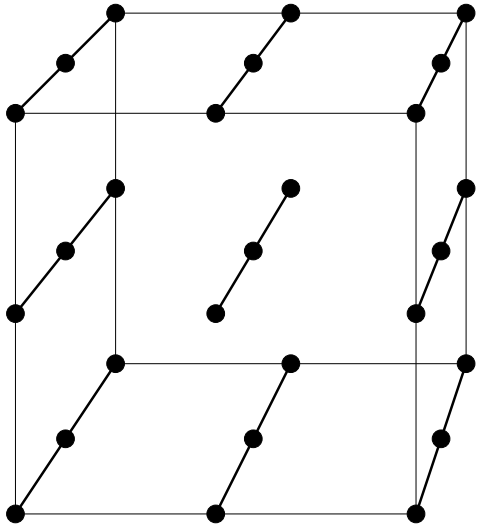
Indexraum,
Abhängigkeiten

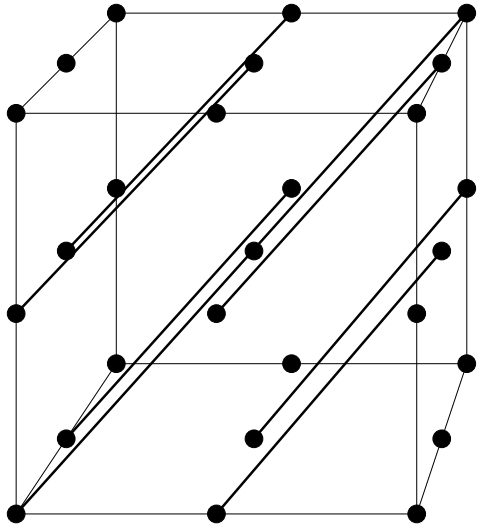




parallele
Schritte

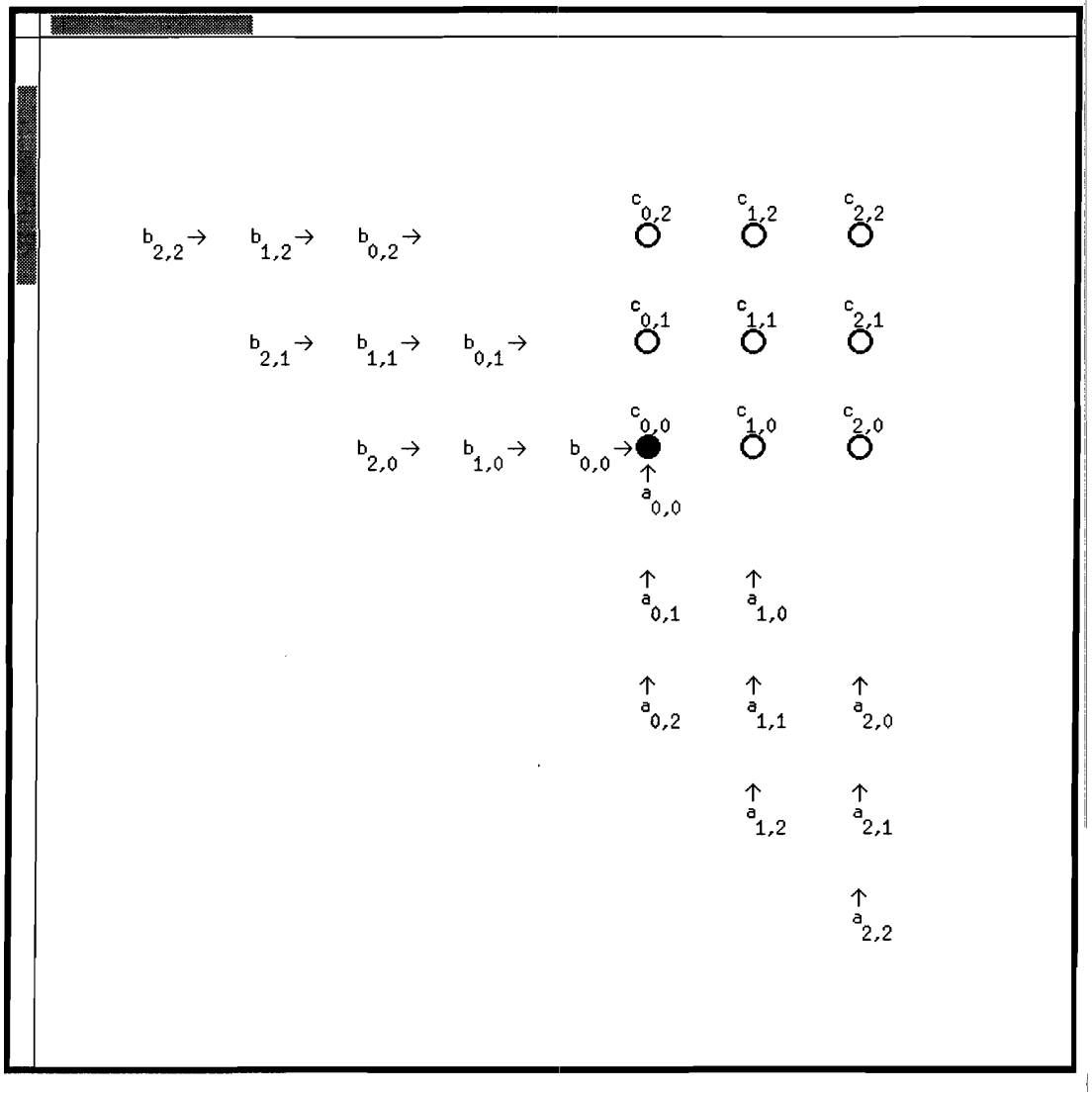
quadratisches
Prozessorfeld



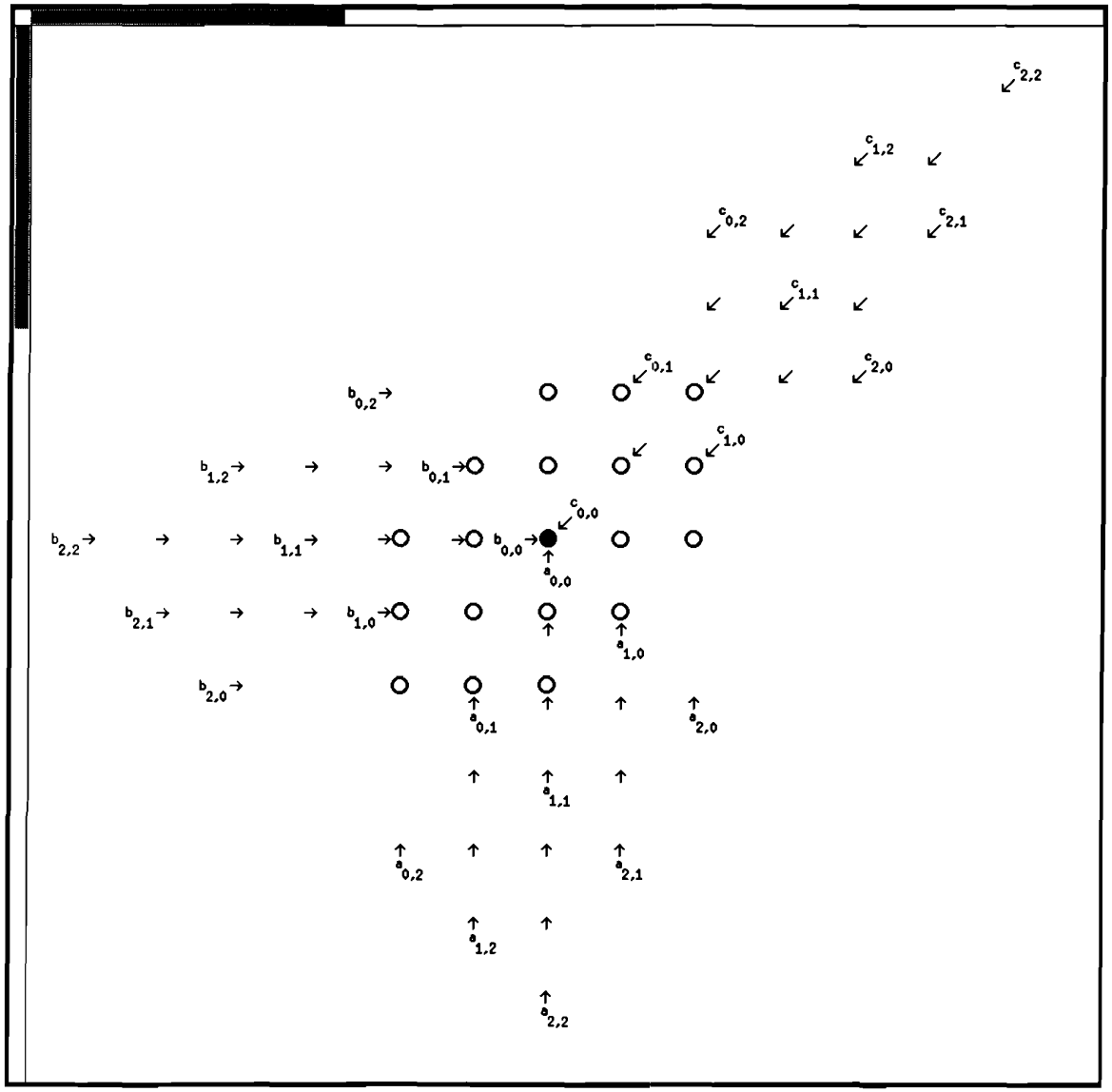


hexagonales
Prozessorfeld

Quadratische Lösung



Hexagonale Lösung



Einschränkungen und Nutzung des Basismodells

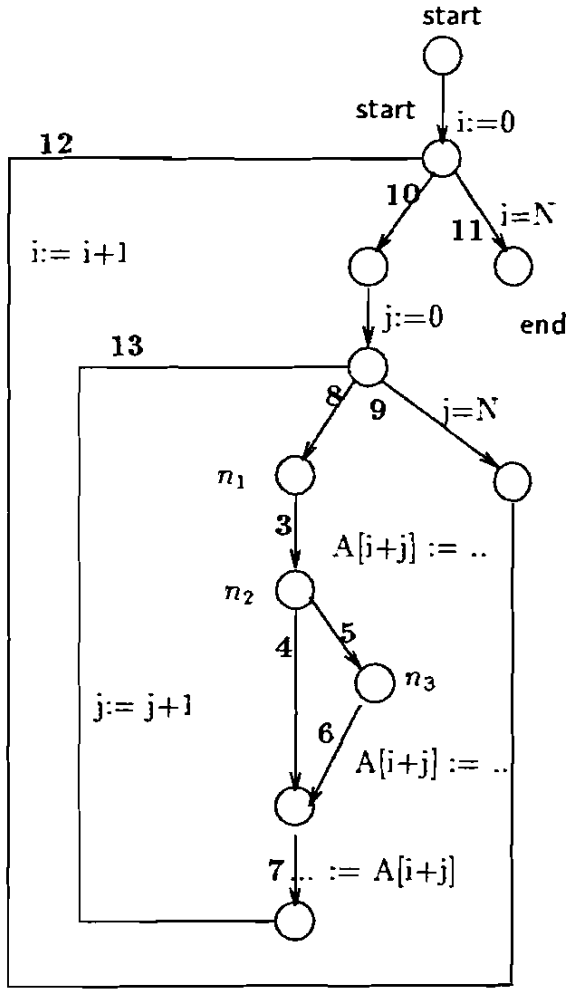
- Einschränkungen:
  - Schleifengrenzen: affine Ausdrücke ($A\,x + b$) in den Indizes der umgebenden Schleifen und in Strukturparametern
  - Feldindizes: affine Ausdrücke in den Indizes der umgebenden Schleifen
  - Zielvariablen von Zuweisungen: Feldelemente oder Skalare
  - Schleifensätze: dürfen imperfekt sein
  - Unterprogrammaufrufe: gelten als atomar und werden nicht parallelisiert
  - Verzeigerte Strukturen: nicht erlaubt, nur Felder
  - Optionen für Zielschleifensätze:
    - synchron (äußere Schleifen sequenziell)
    - asynchron (äußere Schleifen parallel)
- Nutzung:
  - Schleifenparallelisierung
  - Cacheoptimierung

Erweiterung 1: Fallunterscheidungen im Schleifenrumpf

[Jean-François Collard, Martin Griebl]

- Konsequenz:
  - Abhängigkeiten können von Fall zu Fall variieren.

```
   real A[0:2*N+1]
   for i = 0 to N
      for j = 0 to N
3        A[i+j+1] := ..
         if (P) then
6           A[i+j] := ..
         end if
7        ..  := A[i+j]
      end for
   end for
```



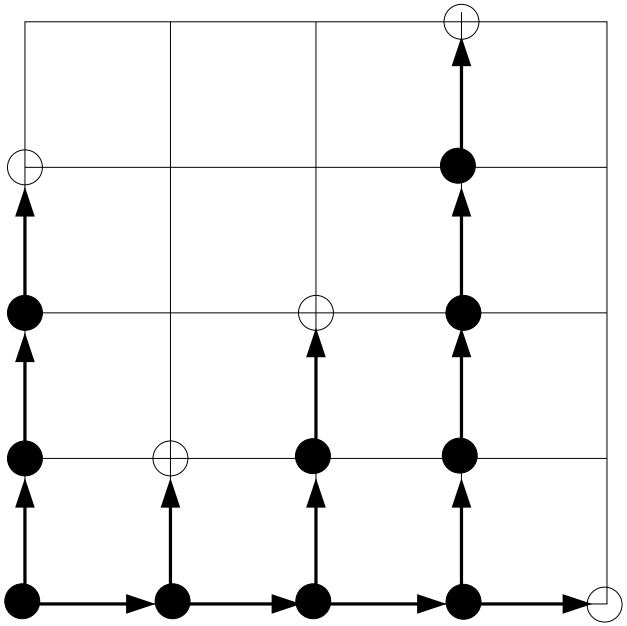
Erweiterung 1: Fallunterscheidungen im Schleifenrumpf

- Methode:
  - Eine präzise Reaching-Definition-Analyse, die folgendes kombiniert:
    - die iterative Lösung von Datenflussgleichungen
      (erkennt Abhängigkeiten zwischen ganzen Feldern,
      kann Fallunterscheidungen behandeln)
    - lineare Integerprogramming
      (erkennt Abhängigkeiten zwischen einzelnen Feldelementen)
  - Versieht Abhängigkeiten mit Bedingungen.
  - Berechnet die unbedingte Vereinigung aller bedingten Abhängigkeiten.
  - Name: Control flow fuzzy array dependence analysis (CfFADA)

Erweiterung 2: WHILE-Schleifen im Schleifensatz

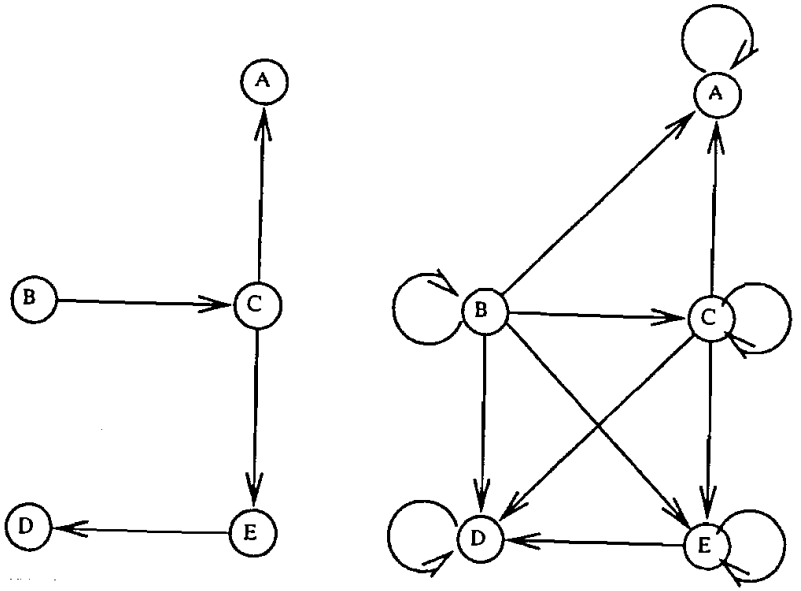
[Jean-François Collard, Martin Griebl]

- Konsequenzen:
  - 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”).



Erweiterung 2: Beispiel (Konvexe Hülle)

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



Erweiterung 2: Beispiel (Konvexe Hülle)

S_1: for $n := 0$ while $node[n] \neq \perp$ do
S_2: $rt[n, 0] := n$
S_3: $nxt[n] := 1$
S_4: for $d := 0$ while $rt[n, d] \neq \perp$ do
S_5: if $\neg tag[n, rt[n, d]]$ then
S_6: $tag[n, rt[n, d]] := tt$
S_7: for $s := 0$ to $nrsuc[rt[n, d]] - 1$ do
S_8: $rt[n, nxt[n]+s] := suc[rt[n, d], s]$
enddo
S_9: $nxt[n] := nxt[n] + nrsuc[rt[n, d]]$
endif
enddo
enddo

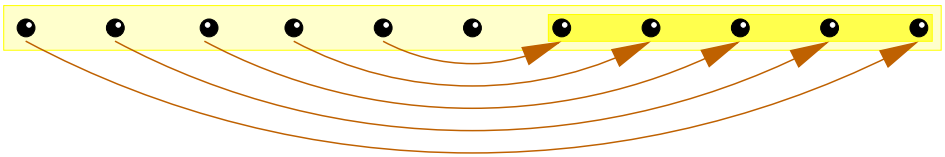
Erweiterung 2: Zwei Ansätze

- 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: Wo enden die “Zähne” des “Kamms”?
    (Gelöst für gemeinsamen und verteilten Speicher.)
- Spekulativ: [Jean-François Collard]
  - Die Kontrollabhängigkeit der WHILE-Schleife wird ignoriert.
  - Ein einzelnes WHILE kann parallel ablaufen.
  - Zusätzlicher Speicherbedarf ist möglich.
  - Ein Backtracking von Schleifenschritten kann notwendig werden.
  - Herausforderungen:
    - Implementierung von Backtracking
    - Vermeidung von Backtracking
    - Minimierung des Speicherbedarfs

Erweiterung 3: Index Set Splitting

[Martin Griebl, Paul Feautrier]

- Idee:
  - 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

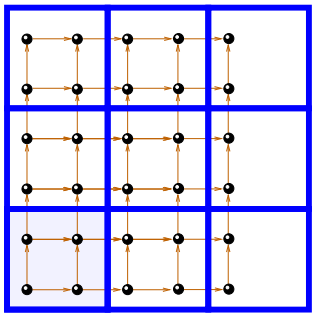
$\Longrightarrow$

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

- Methode:
  - Trenne die Senken des Graphen vom Rest.
  - Propagiere die Trennungen rückwärts durch den Graphen
  - Herausforderung: Termination bei Zyklen (Schrittgrenze)

Erweiterung 4: Kacheln (*Tiling*)

[Martin Griebl]



- Goal: Bestimmte optimale Granularität der Parallelität
  - Wann? (Vor oder nach der Parallelisierung.)
  - Wie? (Form und Größe der Kacheln.)
  - Was? (Raum oder Zeit.)
- Wann: Nach der Parallelisierung
  - Einfacher und allgemeiner: ein einziger, perfekter Zielschleifensatz.
  - Mächtiger: flexible Raumzeitabbildung vor inflexiblem Kacheln.
- Wie: Raum und Zeit getrennt
  - Risiko: Heuristik zahlt sich nur bei bestimmten Allokationen aus.
  - Gewinn: präzise und unabhängige Anpassung an Hardwareparameter.
- Was:
  - Raum: Anpassung an Betriebsmittel (Anzahl der Prozessoren)
  - Zeit: Anpassung an Performanz (Verhältnis Berechnung/Kommunikation)

Erweiterung 5: Ausdrücke

[Peter Faber]

- Ziel: Vermeide wiederholte Berechnungen
- 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)
```

Erweiterung 6: Nicht-affine Feldindexausdrücke

[Armin Größlinger]

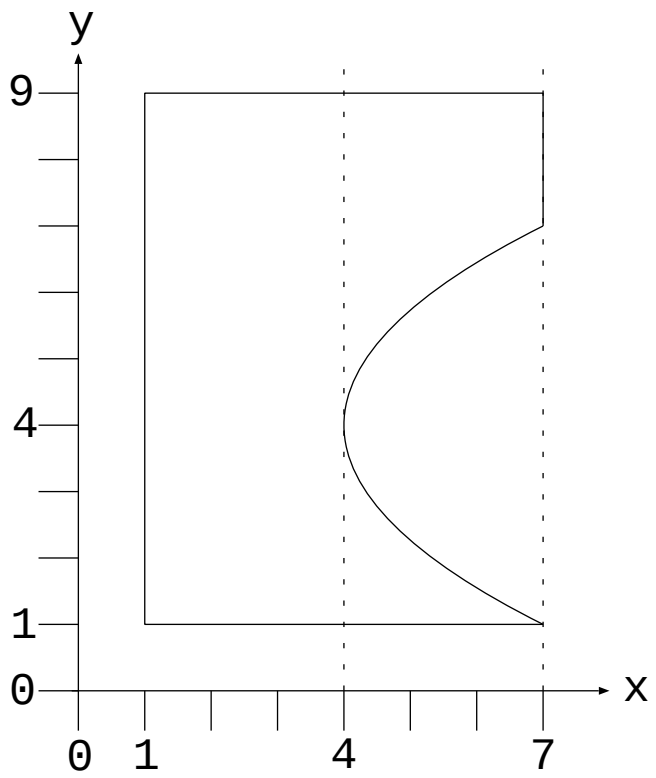
- Ziel: Behandlung von Ausdrücken der Form `A(p*i)`
- “Parameter” p:
  - Hat bekannten, festen Wert.
  - Typischer Fall: Ausdehnung des Polyeders in einer festen Dimension.
- Anwendung: Wähle Zeile oder Spalte einer Matrix als Vektor.
- Methode:
  - Löse die Konfliktgleichungen in $\mathbb{Z}$.
  - Es gibt einen Algorithmus für genau einen Parameter.
  - Mathematik: ganzzahlige Quasipolynome
    (Polynome, deren Koeffizienten periodische Funktionen sind).
- Herausforderung: Abhängigkeitsanalyse
  - Sind die Lösungen innerhalb oder außerhalb des Iterationsraums?
  - In welche Richtung weist die Abhängigkeit?

Erweiterung 7: Nicht-affine Schleifengrenzen

[Armin Größlinger]

- Ziel: Aufzählung von Domänen mit ungeraden Grenzen
  - Grenzen müssen mit Polynomen beschreibbar sein.
  - Domänen sind semi-algebraische Mengen
    (Lösungsmengen von Ungleichungssystemen von Polynomen in $\mathbb{Z}$).
- Anwendungen:
  - Normaler Quellcode: Sieb des Eratosthenes (Grenze `i*i<=n`)
  - Nicht-konstante Schleifenschritte:
    - Beispiel:
      ```
      for (j=0; j<=n; j+=i)
      ```
      $\longrightarrow$
      ```
      for (k=0; k*i<=n; k++)
      ```
      im Schleifenrumpf: `j` $\longrightarrow$ `k*i`
  - Nicht-lineare Schleifentransformationen:
    - Nicht-lineare Schedules können erheblich performanter sein als lineare.
- Herausforderungen:
  - Vermeide Nicht-Affinitäten in der Abhängigkeitsanalyse;
    verschiebe sie in die Codegenerierung.
  - Codevereinfachung

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);
}
```

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.
  - Anwendung: Kacheln und Codegenerierung.
  - Mathematik: Quantorenelimination (in $\mathbb{R}$).
- Auch nicht-lineare Variablen: Z.B. p^2`*i`^{2}, `p*i`^{2}, `i*j`
  - Anwendung: Codegenerierung zur Aufzählung beliebiger semi-algebraischer Mengen.
  - Mathematik: Zylindrische algebraische Dekomposition.

Der Schleifenparallelisierer LooPo

- Eingabe:
  - Schleifencode ohne Parallelität (FORTRAN, C, Rekursionsgleichungen)
  - Spezifikation eines Datenflussgraphen (nächster Schritt unnötig)
- Abhängigkeitsanalyse: Übergang zum Modell
  - Methode: Banerjee (eingeschränkt), Feautrier (vollständig), CfFADA (kann Fallunterscheidungen behandeln)
  - Optional: *index set splitting*, *single-assignment conversion*
- Raumzeitabblidung:
  - Schedule: Lamport (einfach), Feautrier (vollständig), Darte-Vivien (Kompromiss)
  - Allokation: Feautrier (vollständig), Dion-Robert (praktischer), nur Vorwärtskommunikation (bereitet Kachelung vor)
- Codegenerierung:
  - Basiert auf dem französischen Schleifencodegenerierer CLooG
  - Erzeugt Schleifencode und Kommunikationscode
  - Kachelt

Parallele Programmskelette

- Idee:
  - Gib oft gebrauchte Parallelitätsmuster vor
  - Spezifiziere jedes Muster als Funktion höherer Ordnung
  - Biete Implementierungen für verschiedene Plattformen an
  - Wenn möglich, mache die Skelette mit Metaprogrammierung adaptiv
- Beispiele:
  - Im Kleinen: kollektive Operationen
    - Datentransfer: broadcast, scatter, gather, all-to-all
    - Datentransfer + Berechnung: reduce, scan
  - Im Größeren: algorithmische Muster
    - divide-and-conquer, branch-and-bound
    - dynamische Programmierung, Suche in Suffixbäumen
    - Algorithmen auf markierten Graphen
- Methode:
  - Funktionale Quellsprache: Template Haskell, MetaML, MetaOCaml
  - Imperative Zielsprache: C, C+MPI,...
  - Übersetzung: noch keine Standardtools

Kollektive Operationen

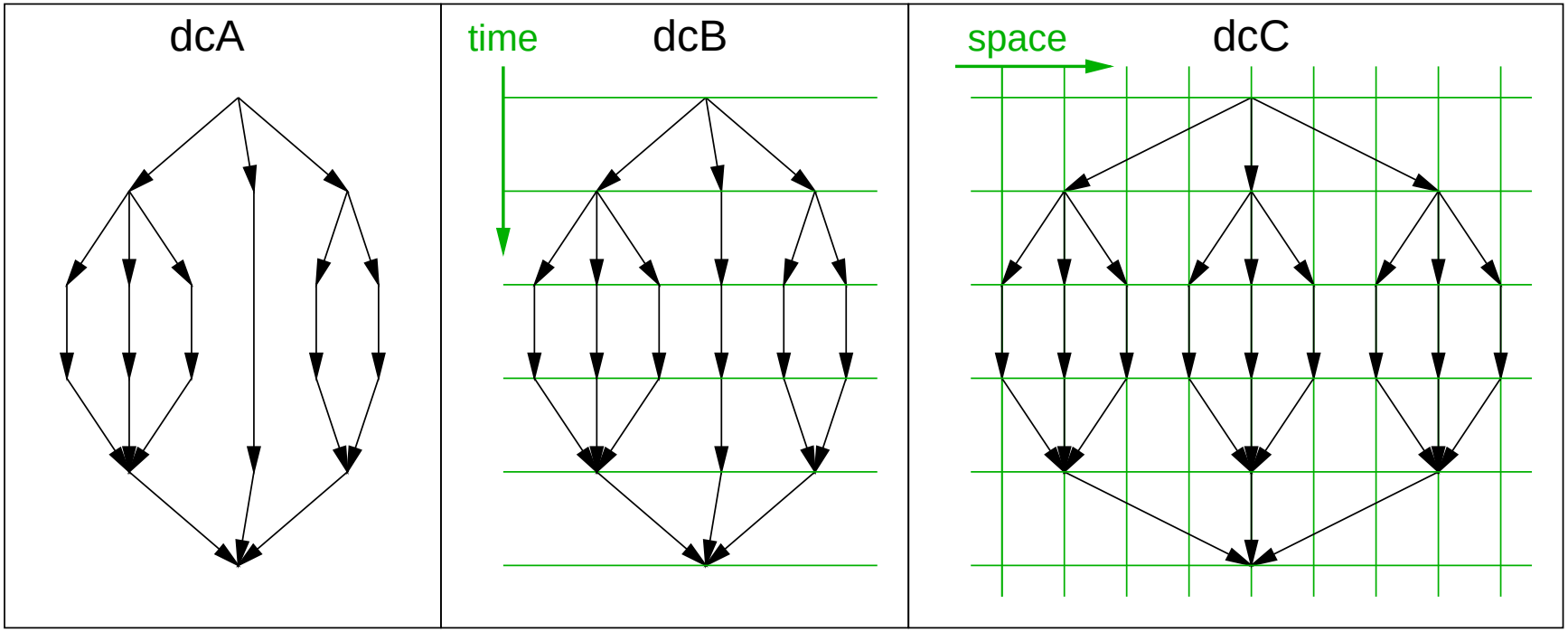
[Sergei Gorlatch]

t_s: *start-up time* t_w: *per-word transfer time* m: Blockgröße

Kompositionsregel	besser wenn
`Scan_1; Reduce_2` $\longrightarrow$ `Reduce`	immer
`Scan; Reduce` $\longrightarrow$ `Reduce`	$t_s > m$
`Scan_1; Scan_2` $\longrightarrow$ `Scan`	$t_s > 2m$
`Scan; Scan` $\longrightarrow$ `Scan`	$t_s > m(t_w + 4)$
`Bcast; Scan` $\longrightarrow$ `Comcast`	immer
`Bcast; Scan_1; Scan_2` $\longrightarrow$ `Comcast`	$t_s > \frac{m}{2}$
`Bcast; Scan; Scan` $\longrightarrow$ `Comcast`	$t_s > m(\frac{1}{2}t_w + 4)$
`Bcast; Reduce` $\longrightarrow$ `Local`	immer
`Bcast; Scan_1; Reduce_2` $\longrightarrow$ `Local`	immer
`Bcast; Scan; Reduce` $\longrightarrow$ `Local`	$t_w + \frac{1}{m}t_s \geq \frac{1}{3}$

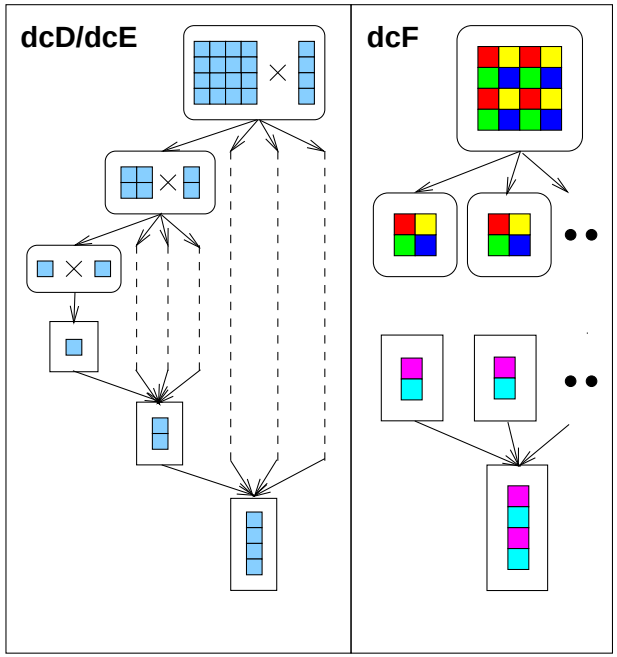
Divide-and-Conquer-Hierarchie: Tasks

[Christoph A. Herrmann]



Skelett	Einschränkung	Anwendung
dcA	unabhängige Teilprobleme	Quicksort, *maximum independent set*
dcB	feste Rekursionstiefe	*n* Königinnen
dcC	fester Teilungsgrad *k*	Karatsuba Integerprodukt (*k*=3)

Divide-and-Conquer-Hierarchie: Daten



`dcD`	Blockrekursion	Dreieicksmatrixinversion (k=2), Batcher Sort (k=2)
`dcE`	elementweise Operationen	Matrix-Vektor-Produkt (k=4)
`dcF`	Kommunication zwischen korrespondierenden Elementen	Karatsubas Polynomprodukt (k=3), *bitonic merge* (k=2), FFT (k=2), Strassens Matrixprodukt (k=7)

Skelettimplementierung

- Prinzip:
  - Spezifikation *recX* = iterative Form *itX*
  - Übergang von der Quellsprache Haskell zur Zielsprache C+MPI
- `dcA (base, divide, combine, input)`
  - dynamische Allokation von Zeit und Raum
  - kein Lastenausgleich
- `dcF (k, indeg, outdeg, basic, divide, combine, n, input)`
  - statische Allokation von Zeit und Raum über zusätzliche Parameter
    - Zahl der Teilprobleme
    - Teilungsgrad der Eingabedaten
    - Kombinationsgrad der Ausgabedaten
    - Rekursionstiefe
  - Abhängigkeiten regulär aber nicht affin (Analyse unnötig)
  - ähnlich wie das Polydermodell aber keine Suche nach einem Schedule
  - symbolische Größeninferenz der Skelettparameter

Was leisten die Methoden?

- Automation erfordert hohe (affine) Regularität.
- Konstant viele Brüche der Regularität sind verkraftbar.
- Nicht-Affinität erfordert erhebliche mathematische Mittel.
- Codegenerierung ist i.A. sehr schwierig; Heuristiken helfen.

- Automatische Schleifenrestrukturierung wird immer praktikabler.
- Skelette eignen sich insb. für Multicores und eingebettete Systeme.
- Es ist schwer, allegmein akzeptiere, abstrakte Skelette zu finden.
- Es gibt immer mehr prototypische Tools.
- Am besten: baue spezialisiertes Tool aus Toolsets.

Referenzen

- Klassisches Polyedermodell

  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.

- Erweiterung 1: Fallunterscheidungen im Schleifenrumpf

  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

  Martin Griebl. *The Mechanical Parallelization of Loop Nests Containing* while *Loops*. Dissertation, Universität Passau, 1996. Technical Report MIP-9701 (Konservativer Ansatz).

  Jean-François Collard. Automatic parallelization of while-loops using speculative execution. *Int. J. Parallel Programming*, 23(2):191–219, 1995 (Spekulativer Ansatz).

- Erweiterung 3: Index Set Splitting

  Martin Griebl, Paul Feautrier, and Christian Lengauer. Index set splitting. *Int. J. Parallel Programming*, 28(6):607–631, 2000.

Referenzen

- Erweiterung 4: Kacheln

  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, March 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

  Peter Faber. *Code Optimization in the Polyhedron Model – Improving the Efficiency of Parallel Loop Nests*. Dissertation, Universität Passau, 2008. 978-1-4092-5550-5.

- Erweiterung 6: Nicht-affine Feldindexausdrücke

  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, 2009. ISBN 978-1-4452-5421-0.

- Erweiterung 7: Nicht-affine Schleifengrenzen

  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, 2009. ISBN 978-1-4452-5421-0.

Referenzen

- Skelette im Kleinen: Kollektive Operationen

  Sergei Gorlatch. Send-receive considered harmful: Myths and realities of message passing. *ACM TOPLAS*, 26(1):47–56, 2004.

- Skelette im Größeren: Divide-and-Conquer

  Christoph Armin Herrmann. *The Skeleton-Based Parallelization of Divide-and-Conquer Recursions*. Dissertation, Universität Passau, 2000. ISBN 3-89722-556-5.

- Metaprogrammierte Skelette

  Christoph Armin Herrmann and Christian Lengauer. Using metaprogramming to parallelize functional specifications. *Parallel Processing Letters*, 12(2):93–210, June 2002.