technische universität
dortmund

fakultät für informatik
informatik 12

Synthese Eingebetteter Systeme

Wintersemester 2012/13

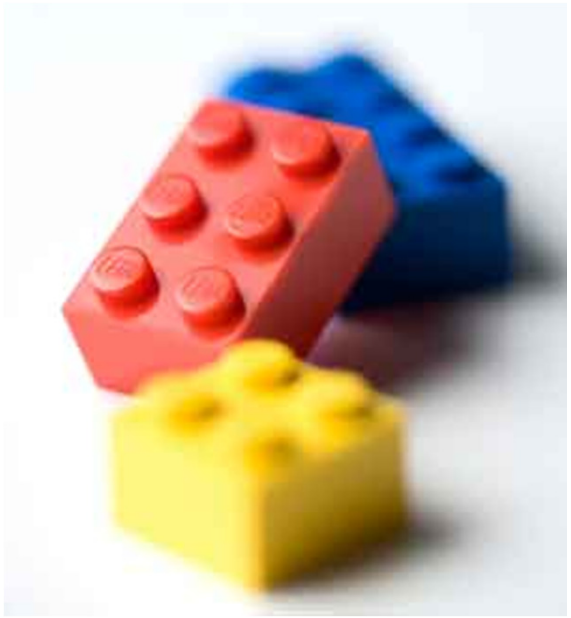
19 – Daedalus: Parallelisierung mit Polytopmodellen

Michael Engel
Informatik 12
TU Dortmund

2013/01/16

Parallelisierung mit Polytopmodellen

- Daedalus: Umwandlung von C zu KPN
- Schleifenprogramme
- Polytopmodell

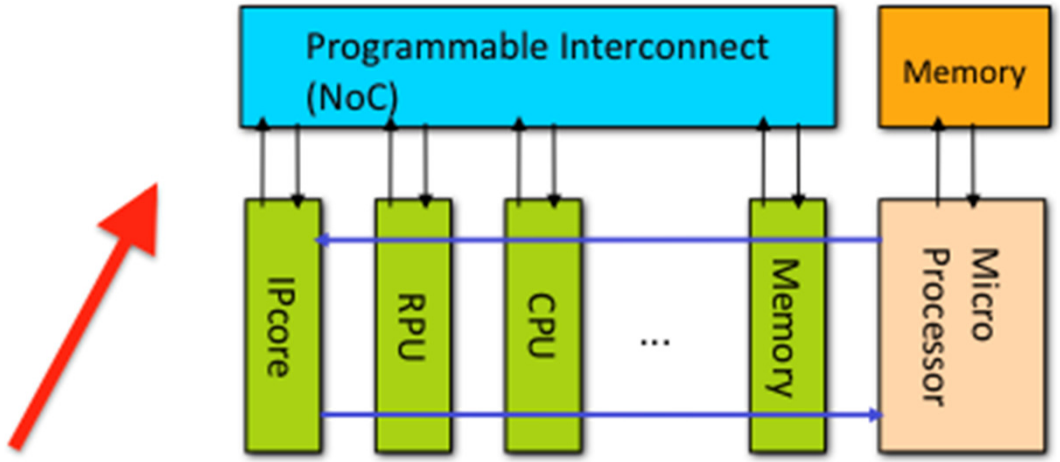


Ziel

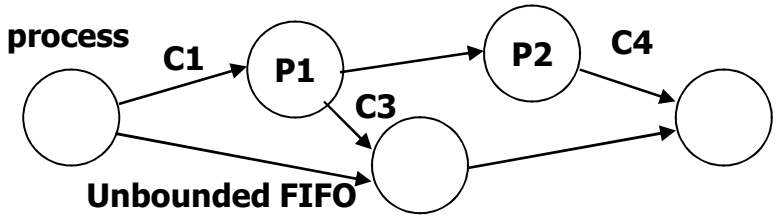
- Erzeugung von Datenflussmodellen, die *korrekt durch Konstruktion* sind
  - Diese sind Ein-/Ausgabe-äquivalent zu gegebenen imperativen sequentiellen verschachtelten Schleifenspezifikationen
- Für viele Signalverarbeitungsanwendungen machbar

```
%parameter N 8 16;
%parameter K 100 1000;

for k = 1:1:K,
 for j = 1:1:N,
   [r(j,j), x(k,j), t ]=F( r(j,j), x(k,j) );
   for i = j+1:1:N,
       [r(j,i), x(k,i), t]=G( r(j,i), x(k,i), t );
 end
 end
end
```



KPN: Beispiel



Process P1 (‘producer’)

```
While (1){
  Read(C1, token);
  if (token != Token) {
  Write(C2, Execute(token));
  {
  else{
  Write(C3, token);
  }
 }
```

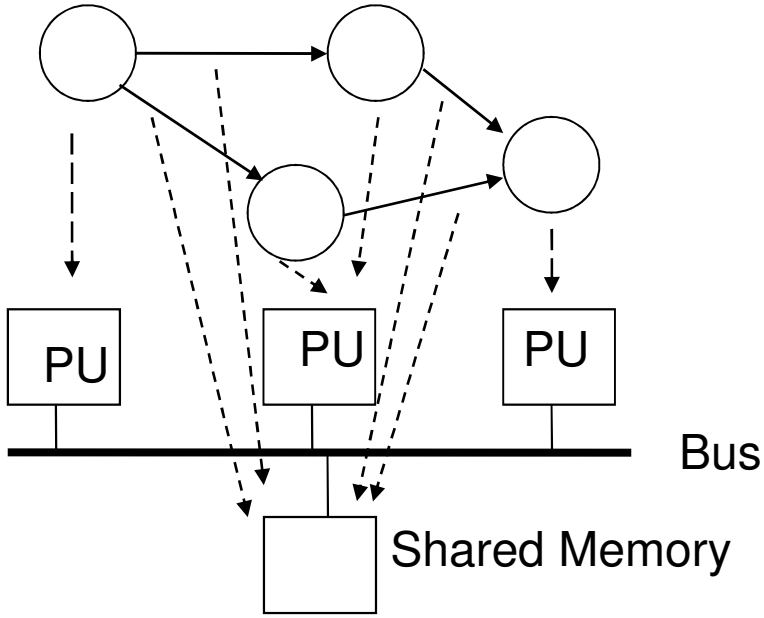
ProcessP2 (‘consumer’)

```
While(1){
 Read(C2, token);
 Write(C4, Execute(token));
}
```

Das charakteristische Tripel von Operationen ist {Read, Execute, Write}. Execute bezeichnet einen abstrakten Rechenoperator, die Kommunikation ist stets point-to-point

Abbildung

Eine als KPN modellierte Anwendung wird für eine parallele Multiprozessor-Architektur *transformiert* (abgebildet oder verteilt)



Umwandlung von C in ein KPN-Modell

Die meisten Anwendungen werden (immer noch) als imperative sequentielle Programme, z.B. in C oder C++ entwickelt.

In Spezialfällen können sie *automatisch* in Ein-/Ausgabe-äquivalente KPNs oder DPNs umgewandelt werden.

Prozessnetzwerke eignen sich besser für die Abbildung auf multiprozessor-basierte Ausführungsplattformen.

Übersetzung und Abbildung

LEICHT angebbar

Sequential Application Specification

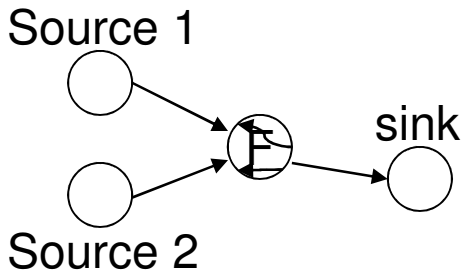
```
for j = 1:1:N,
  [x(j)] = Source1( );
end
for i = 1:1:K,
  [y(i)] = Source2( );
end
for j = 1:1:N,
  for i = 1:1:K,
    [y(i), x(j)] = F( y(i), x(j) );
  end
end
for i = 1:1:K,
  [Out(i)] = Sink( y( I ) );
end
```

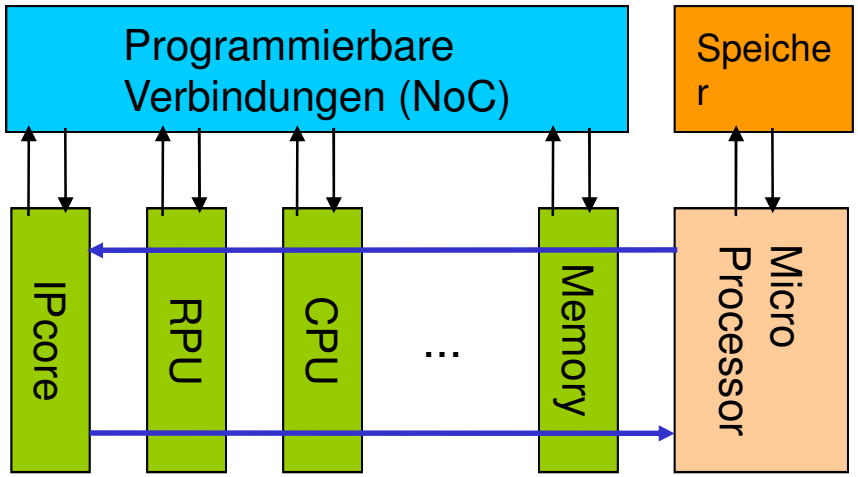
SCHWER abbildbar

Anwendung

SCHWIERIG anzugeben

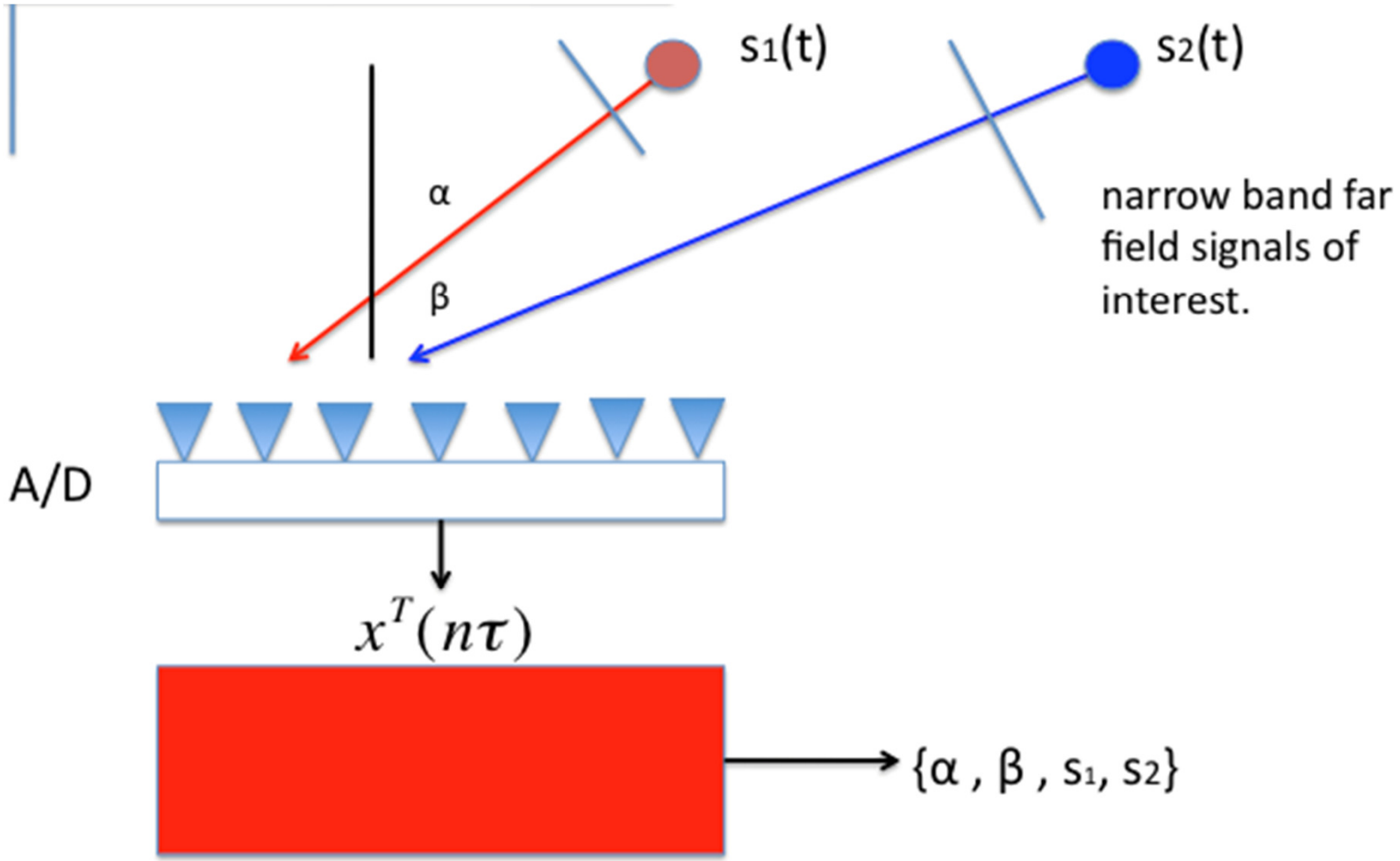
Parallele Anwendungsspezifikation



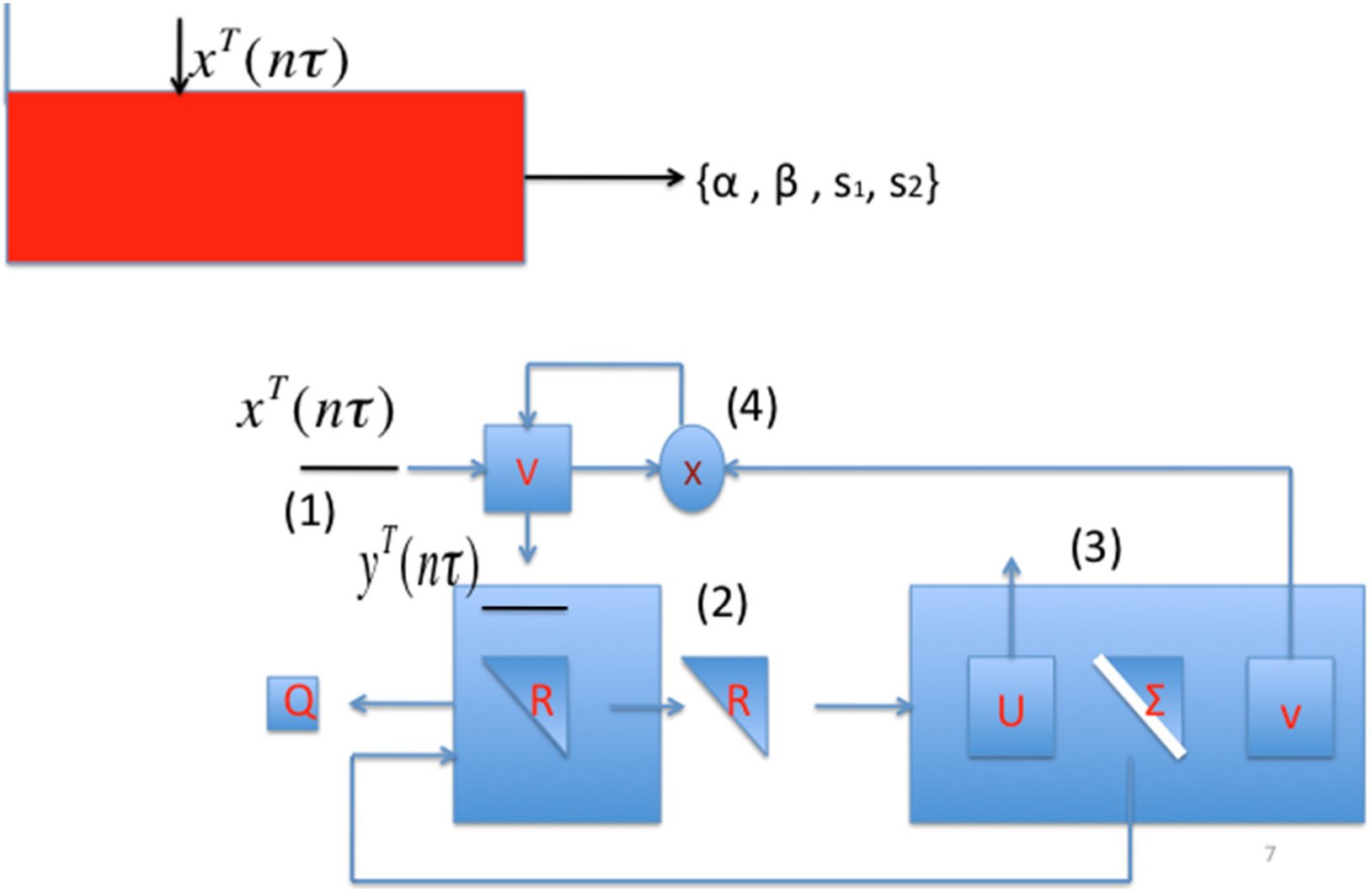


EINFACH abbildbar

Beispiel: Beam Forming



Beispiel: Beam Forming

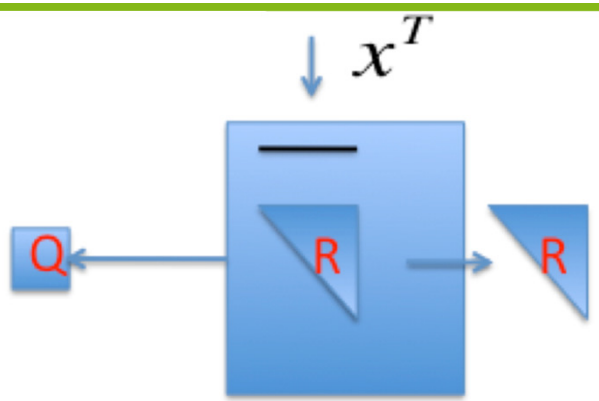


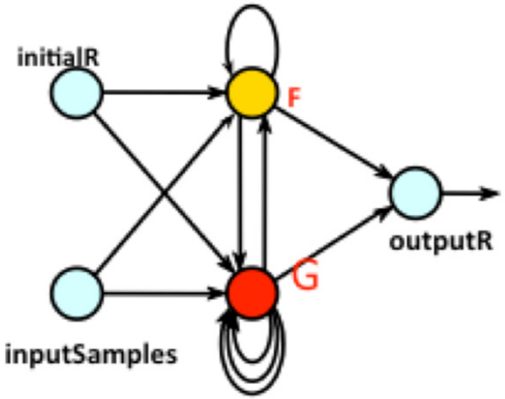
QR-Algorithmus

C-like program:

```
1. for {k = 1;k < K;k++}
2.  for {j = 1;j < N;j++}
3.   [r(j,j) θ(j)]  =  F(r(j,j), x(k,j));
4,     for {i = j+1;i < N;i++}
5.      [r(i,j), x(k,i)] = G(r(i,j), x(k,i), θ(j));
```

Process Network:





8

Polytope

- Ziel: Automatische Parallelisierung von Programmen
- Spezialfall: Parallelisieren von Schleifen
- Grundidee:
  - Vorliegendes Schleifenprogramm als *Polytop* ansehen
- Transformationen auf Polytopen erreichen, dass das resultierende Schleifenprogramm parallelisierbar ist
  - Transformationen beschreiben Basiswechsel der Indexvariablen
  - Transformationen ordnen Programm neu, so dass Datenabhängigkeiten erhalten bleiben und möglichst viele Iterationen der Schleife parallel ausgeführt werden können.

Polytope

- Polytop = verallgemeinertes Polygon in beliebiger Dimension
- konvexe Polytope als Ungleichungssystem darstellbar:

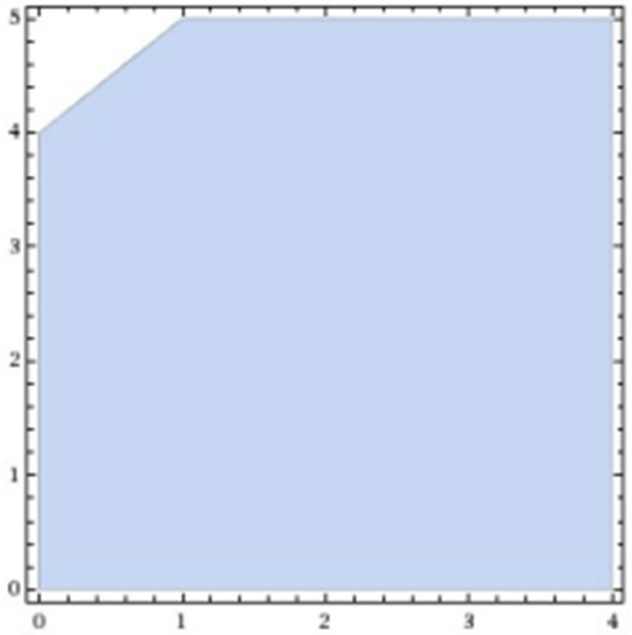
$$A * \vec{x} \geq \vec{b}$$

  - A ist (n x m)-Matrix
- Jede Zeile des Ungleichungssystems beschreibt einen Halbraum
- Polytop besteht also aus n Halbräumen und hat die Dimension m
  - Für m = 2 bzw. m = 3 nennt man Polytope auch Polygone bzw. Polyeder

Polytope: Beispiel

- Im zweidimensionalen lässt sich das Polytop mit fünf Halbräumen definieren:
  - $x \geq 0$
  - $x \leq 4$
  - $y \geq^{\perp} 0$
  - $y \leq 5$
  - $x - y \geq^{\perp} -5$
- Oder in der Form $A * \vec{x} \geq \vec{b}$:

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



Quellprogramm

- Schleifenprogramm, welches mittels des Polytop-Modells parallelisiert werden soll
- Bedingungen, um mit Polytopen beschreibbar zu sein:
  - 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
- untere Schleifengrenzen dürfen durch Max-Funktion, obere durch Min-Funktion beschrieben werden

Affine Funktionen

- Affine Funktion:
  - Allgemeine linerare Funktion
  - Allg. Beschrieben durch f(x) = ax + b
- Affine Funktionen in den Schleifengrenzen sind notwendig, damit Schleifenprogramm als konvexes Polytop beschrieben werden kann
- Affine Funktionen in Array-Indizes sind für die Berechnung von Schedule und Allokation notwendig

Begriffsdefinitionen

Indexraum

- 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*:

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

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

```
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
```

Schleifenprogramm

Begriffsdefinitionen und Beispiel

Operation

- Bezeichnet ein Statement S an der Stelle

  $\vec{i} = (i_1, i_2, \ldots, i_n)$

- Schleifenprogramm, das alle Bedingungen erfüllt:

```
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
```

1. Nur for-Schleifen und Zuweisungen
2. Datentypen nur Arrays und Skalare
3. Schleifengrenzen/Array-Indizes affine Funktionen in den Indizes umgebener Schleifen
4. konstante Parameter in den affinen Funktionen erlaubt
5. untere Schleifengrenzen dürfen durch Max-Funktion, obere durch Min-Funktion beschrieben sein

Quellpolytop

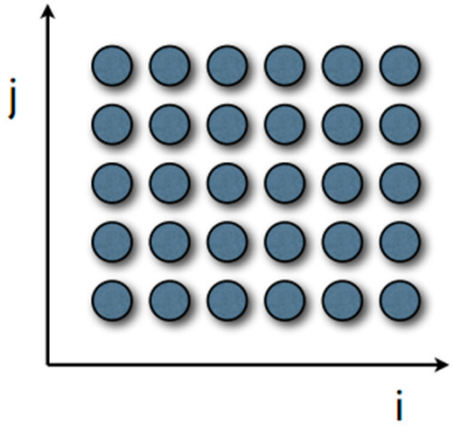
- Aus dem Quellprogramm lasst sich direkt das Quellpolytop ableiten
- Indexraum betrachten und als Polytop der Form

  $A * \vec{x} \geq \vec{b}$ beschreiben
- Im Beispielprogramm:

```
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
```

  - Indexraum

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



Zielpolytop

Aufgabe:

- Quellpolytop so transformieren, dass das resultierende Programm parallelisierbar ist
- Polytop muss durch Matrixmultiplikation verschoben werden, ohne Datenabhängigkeiten zu verletzen
- Transformation benötigt zwei Funktionen:
  - Schedule und
  - Allokation
  - Auch hier: *nur affine Funktionen zulässig*

Schedule

- Funktion, die jeder Operation Ω einen festen Zeitpunkt zuordnet, wird Schedule genannt:

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

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

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

- Dies stellt sicher, dass eine Operation B nicht vor einer anderen Operation A ausgeführt wird, falls B von A abhängt

Allokation

- Schedule stellt die zeitliche Komponente der Transformation, Allokation die räumliche Komponente
- Allokation weist einer Operation einen bestimmten Prozessor zu:

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

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

Transformation

- Transformation des Polytops $A * \vec{x} \geq \vec{b}$ lässt sich nun mit Hilfe der Koeffizienten der zwei Funktionen t (Schedule) und a (Allokation) berechnen
- Transformationsmatrix:

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

mit den affinen Funktionen

$$t(S, \vec{i}) = \vec{\lambda} * \vec{i} + \vec{c} \quad \text{und} \quad a(S, \vec{i}) = \vec{\sigma} * \vec{i} + \vec{d}$$

- Berechnung der neuen Indizes $\vec{j}$ des Polytops durch $\vec{j} = T * \vec{i}$
- Neue Grenzen des Polytops: $A * (T^{-1} * \vec{j}) \geq \vec{b}$
  - Inverse Matrix T^{-1} existiert nicht immer

Transformation: Beispiel

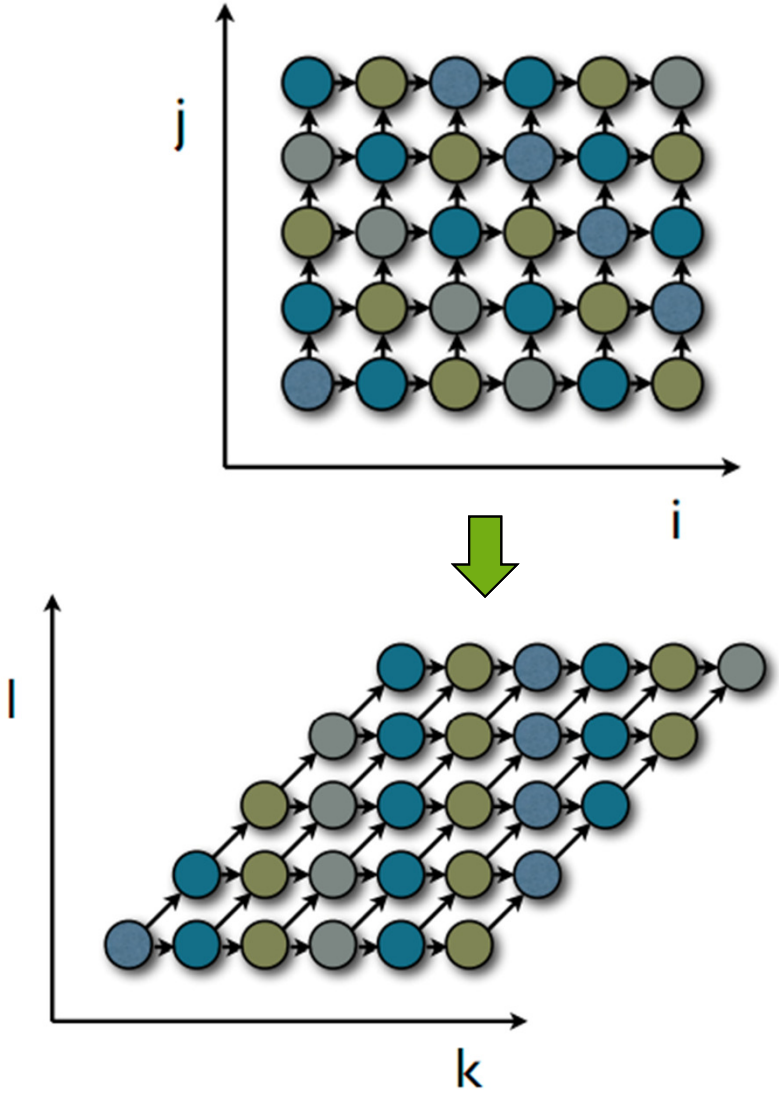
- Für das Beispiel ergibt sich mit
  - Schedule $t(S, \vec{i}) = i + j$
  - Allokation $a(S, \vec{i}) = i$

die Transformationsmatrix

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

und die neuen Indizes

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



Zielprogramm

- Transformiertes Polytop muss wieder zurück in gültiges Schleifenprogramm gewandelt werden
- Polytop für Beispiel:

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

```
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
```

- Umwandlung in Schleifenprogramm:
  - k und l als Schleifenindizes interpretieren
  - Versuchen, passende Schleifengrenzen zu finden
- Hier: *nicht möglich!*
  - Löst man die Ungleichungen nach l bzw. k auf, erhält man für die letzten beiden Zeilen einen Ausdruck in Abhängigkeit von k bzw. l ⇒ äußere Schleifengrenzen nicht bestimmbar

Fourier-Motzkin-Elimination

- Problem mit Fourier-Motzkin-Elimination lösbar
  - eliminiert aus Ungleichungssystem einzelne Variablen
  - Ergibt neues Ungleichungssystem, welches nicht mehr von diesen Variablen abhängig ist
- Auflösung der j Ungleichungen nach zu eliminierender Variable x_j ergibt drei Gruppen von Ungleichungen:

$$x_j \geq \sum_{i=1}^{j-1} a_i * x_i \; (Gruppe\ A)$$

$$x_j \leq \sum_{i=1}^{j-1} a_i * x_i \; (Gruppe\ B)$$

$$x_j \quad spielt\ keine\ Rolle\ (\phi)$$

Fourier-Motzkin-Elimination (2)

- Ursprüngliches Gleichungssystem ist somit äquivalent zu:

$$max(A_1, \ldots, A_{n_A}) \leq \quad x_j \leq min(B_1, \ldots, B_{n_B}) \wedge \phi.$$

  - n_A/n_B: Anzahl Ungleichungen in Gruppe A/B
  - A_i/B_i: rechte Seite einer Ungleichung
- Anwendung der Fourier-Motzkin-Elimination auf Ungleichung im Beispiel und Auflösen nach l ergibt:

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

$$max(0, k-4) \leq l \leq min(k, 5)$$

$$\Longrightarrow 0 \leq k \wedge k \leq 9$$

- Damit lässt sich k nun unabhängig von l beschreiben

Resultierendes Schleifenprogramm

- Das sich ergebende Schleifenprogramm sieht wie folgt aus:

```
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
```

- Die innere Schleife kann nun parallel ausgeführt werden!

Schedule und Allokation

- Bei Berechnen von Schedule und Allokation: Unterscheidung zwischen statementbasiertem und schleifenbasiertem Ansatz:
  - statementbasierter Ansatz berechnet für jedes Statement eine eigene Transformationsmatrix
  - schleifenbasierter Ansatz berechner Matrix für die komplette Schleife

Schleifenbasiert

- Betrachtung einer Schleife als ganzes Element und Berechnung von Schedule und Allokation dafür
- Quellprogramm darf nur aus perfekt verschachtelten Schleifen bestehen:

```
for i_1 := l_1 to u_1 do
    for i_2 := l_2(i_1) to u_2(i_1) do
        for i_3 := l_3(i_1, i_2) to u_3(i_1, i_2) do
            ⋮
            for i_n := l_n(i_1, ..., i_{n-1}) to u_n(i_1, ..., i_{n-1}) do
                {Schleifenkörper B}
            end
            ⋮
        end
    end
end
```

- Die Hyperebenen-Methode von Lamport berechnet Transformationsmatrix zu einer gegebenen Schleife und den Abhängigkeiten der Statements

Statementbasiert

- keine Einschränkungen an das Quellprogramm
- Für jedes Statement lässt sich ein Optimierungsproblem aufstellen
  - Man erhält also auch für jedes Statement eine eigene Transformationsmatrix
- Statementbasierte Methode ist im Gegensatz zu schleifenbasierter optimal bezüglich der Anzahl der parallelisierten Operationen
  - benötigt allerdings deutlich mehr Rechenaufwand

Zusammenfassung

- Daedalus: Umwandlung von C zu KPN
- Schleifenprogramme
- Polytopmodell