Datenparallel Listenenden finden

geg.: int-Array link speichert Listen; link[i] enthält den Index des Nachfolgers oder nil

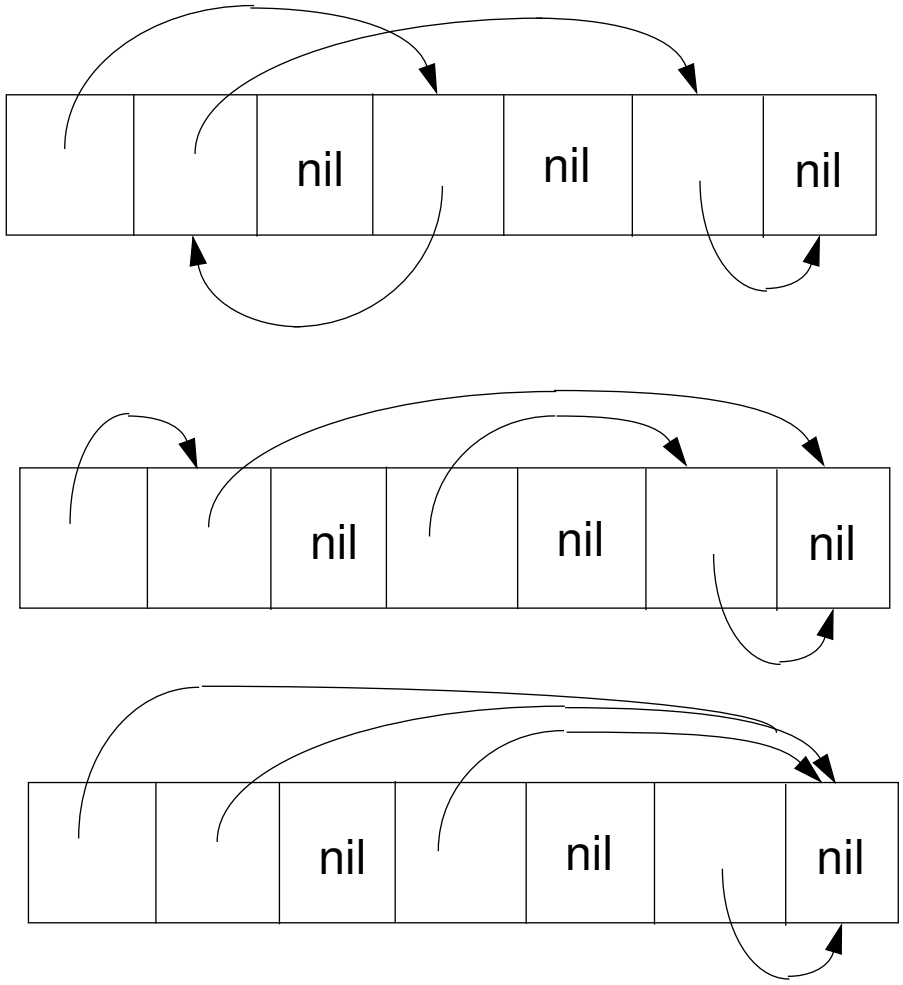
ges.: int-Array last; last[i] enthält den Index des letzten Elementes der Liste link[i]

Methode: Worker Prozess i berechnet last[i] = last[last[i]] in log N **Runden**

```
int d = 1;
last[i] = link[i];
Barriere

while (d < N)
{  int newlast = nil;
   if ( last[i] != nil &&
        last[last[i]] != nil)
      newlast = last[last[i]];
   Barriere
   if (newlast != nil)
      last[i] = newlast;
   Barriere
   d = 2*d;
}
```

last[i] zeigt auf Listenende, wenn es höchstens d Elemente entfernt ist



© 2000 bei Prof. Dr. Uwe Kastens

Schleifenparallelisierung

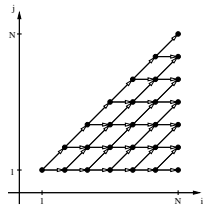
Entwicklungsschritte:

- **geschachtelte Schleifen**,
  die sequentiell **auf Arrays** operieren

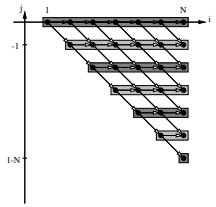
```
DECLARE B[0..N,0..N+1]

FOR I := 1 ..N
    FOR J := 1 .. I
        B[I,J] :=
            B[I-1,J]+B[I-1,J-1]
    END FOR
END FOR
```

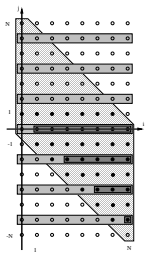
- **Datenabhängigkeiten** analysieren



- **Schleifen transformieren**,
  Datenabhängigkeiten beibehalten
- **innere Schleifen parallel**
  auf Prozessorfeld abbilden



- **Arrays** so auf Prozessorfeld **verteilen**,
  dass **viele Zugriffe lokal** sind



© 2000 bei Prof. Dr. Uwe Kastens

Iterationsraum von Schleifenschachteln

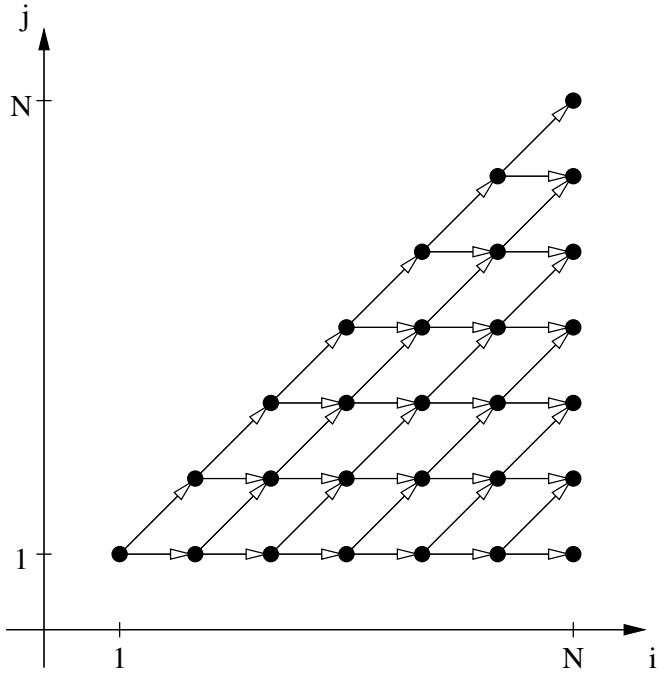
Iterationsraum n-fach geschachtelter Schleifen:

- **n-dimensionaler Raum von ganzzahligen Punkten** (Polytop)
- jeder Punkt (i_1, ..., i_n) steht für eine Ausführung des innersten Schleifenrumpfes
- Schleifengrenzen i. a. erst zur Laufzeit bekannt
- Iterationsraum ist nicht notwendig rechtwinklig begrenzt
- Iterationsraum wird sequentiell abgearbeitet

Beispiel:
Berechnung des Pascalschen Dreiecks

```
DECLARE B[0..N,0..N+1]

FOR I := 1 ..N
    FOR J := 1 .. I
        B[I,J] :=
            B[I-1,J]+B[I-1,J-1]
    END FOR
END FOR
```



© 2000 bei Prof. Dr. Uwe Kastens

Datenabhängigkeiten

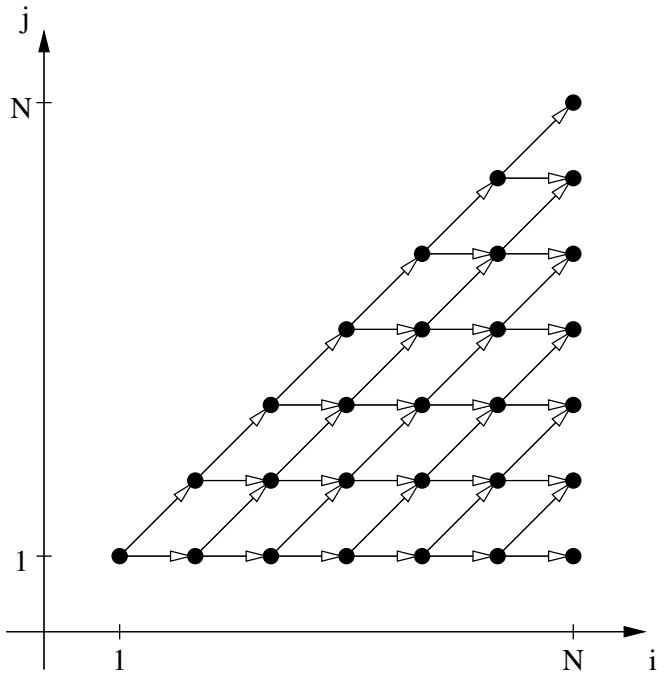
Datenabhängigkeit von Iterationspunkt i zu j:

- Iteration **i** berechnet einen Wert, der in Iteration **j** benutzt wird; (Daten-)Flussabhängigkeit
- relativer **Abhängigkeitsvektor d** = **j - i** = $(j_1 - i_1, ..., j_n - i_n)$
  gilt für alle Iterationspunkte außer am Rand
- Flussabhängigkeiten können **nicht entgegen der Ausführungsreihenfolge** gerichtet sein, nicht rückwärts in der Zeit

Beispiel:
Berechnung des Pascalschen Dreiecks

```
DECLARE B[0..N,0..N+1]

FOR I := 1 ..N
    FOR J := 1 .. I
        B[I,J] :=
            B[I-1,J]+B[I-1,J-1]
    END FOR
END FOR
```



© 2000 bei Prof. Dr. Uwe Kastens

Schleifentransformation

Der **Iterationsraum** geschachtelter Schleifen wird auf **neue Koordinatenachsen** transformiert.

Ziele:

- **innere Schleife(n) parallel** ausführen
- **Lokalität** der Datenzugriffe verbessern; räumlich und zeitlich (Wiederverwendung im Cache)
- **systolisches** Rechen- und Kommunikationsschema erreichen

Datenabhängigkeiten nicht zeitlich rückwärts und nicht gleichzeitig

3 lineare **Grundtransformationen**, kombinierbar:

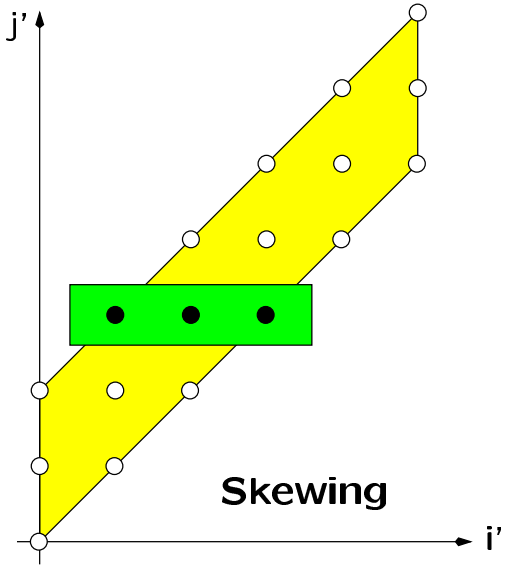
- **Reversal**: eine Dimension des Iterationsraumes spiegeln
- **Permutation**: Schleifen vertauschen
- **Skewing**: Iterationsraum schräg verschieben

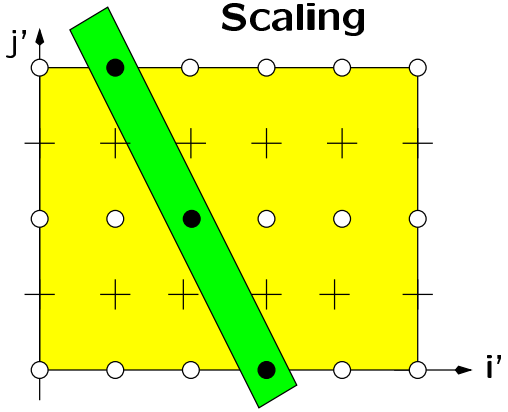
nicht-lineare Transformationen, z. B.:

- **Scaling**: streckt den Iterationsraum in einer Dimension (Lücken)
- **Tiling**: zusätzliche Schleife bearbeitet Kacheln fester Größe

© 2000 bei Prof. Dr. Uwe Kastens

Transformationen von

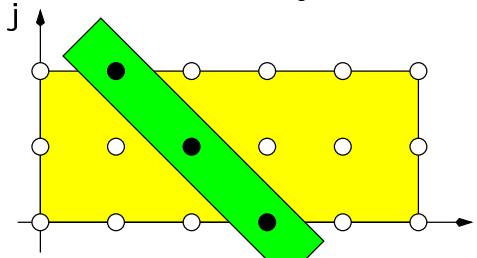




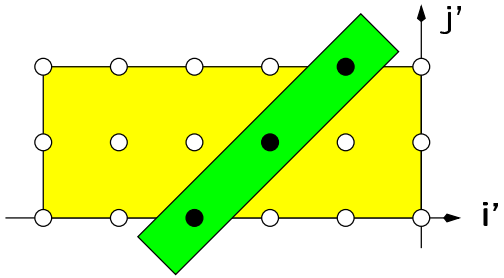
Daten

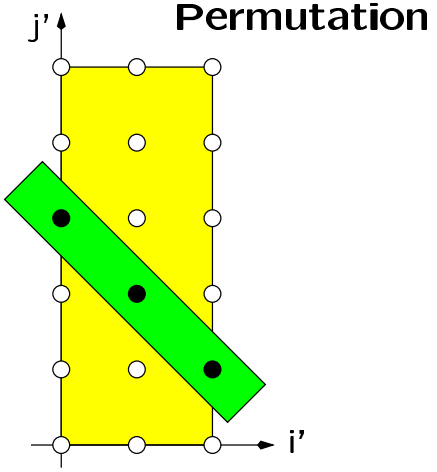
```
REAL B(1:n, 0:m)
```

konvexes Polytop




```
DO i = 0,m-1
   DO j = 0,k-1
      ...
   END
END
```





Schleifenschachteln

Transformationen definiert durch Matrizen

Transformationsmatrizen: systematisches Transformieren, Prüfen von Abhängigkeitsvektoren

Reversal $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} -i \\ j \end{pmatrix} = \begin{pmatrix} i' \\ j' \end{pmatrix}$

Skewing $\begin{pmatrix} 1 & 0 \\ f & 1 \end{pmatrix} * \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i \\ f*i+j \end{pmatrix} = \begin{pmatrix} i' \\ j' \end{pmatrix}$

Permutation $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} j \\ i \end{pmatrix} = \begin{pmatrix} i' \\ j' \end{pmatrix}$

© 2000 bei Prof. Dr. Uwe Kastens

Benutzung von Transformationsmatrizen

- mit Transformationsmatrix **T** neue Iterationspunkte durch alte ausdrücken: **T * i = i´**

z. B. $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} -i \\ j \end{pmatrix} = \begin{pmatrix} i' \\ j' \end{pmatrix}$

- mit Transformationsmatrix T alte Abhängigkeitsvektoren in neue umrechnen: **T * d = d´**

z. B. $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

- mit inverser Transformationsmatrix $\mathbf{T^{-1}}$ i durch i‘ ausdrücken, für Indexausdrücke: $\mathbf{T^{-1} * i = i´}$

z. B. $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{pmatrix} -i' \\ j' \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix}$

- Hintereinanderausführung erst $\mathbf{T^1}$ dann $\mathbf{T^2}$: $\mathbf{T^2 * T^1 = T}$

z. B. $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$

© 2000 bei Prof. Dr. Uwe Kastens

Beispiel: Schleife transformieren und parallelisieren

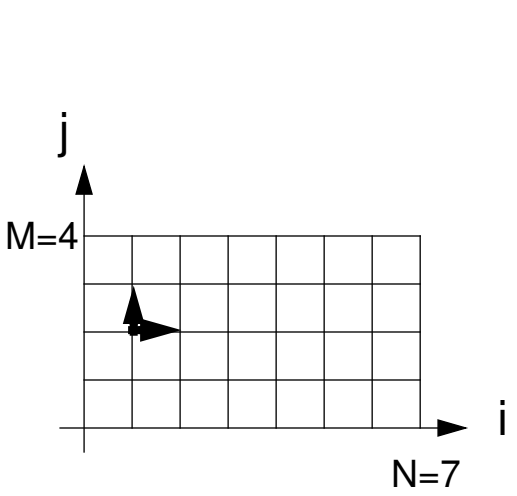
```
for i = 0 to N
   for j = 0 to M
      a[i, j] = (a[i, j-1] + a[i-1, j]) / 2;
```

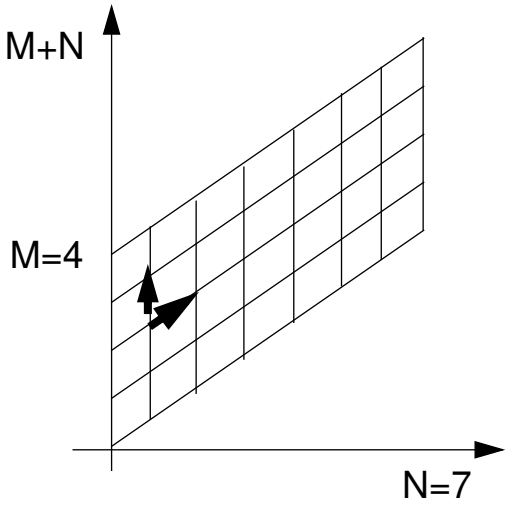
Die obige Schleife soll parallelisiert werden.

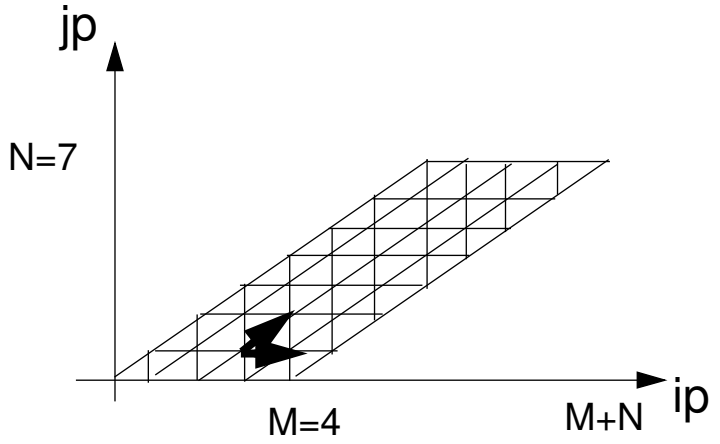
1. Zeichnen Sie den Iterationsraum.
2. Bestimmen Sie die Abhängigkeitsvektoren und zeichnen sie Beispiele davon ein. Warum kann man die innere Schleife nicht parallel ausführen?
3. Wenden Sie eine Skewing-Transformation an und zeichnen den Iterationsraum.
4. Wenden Sie die Permutation-Transformation an und zeichnen Sie den Iterationsraum. Begründen Sie, dass man nun die innere Schleife parallelisieren kann.
5. Geben Sie die zusammengesetzte Transformationsmatrix an und transformieren Sie damit die Abhängigkeitsvektoren.
6. Geben Sie die Inverse der Transformationsmatrix an und transformieren Sie damit die Indexausdrücke.
7. Schreiben Sie die Schleifen vollständig und mit neuen Laufvariablen ip und jp und neuen Schleifengrenzen auf.

© 2000 bei Prof. Dr. Uwe Kastens

Lösung: Schleife transformieren und parallelisieren







$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}$$

Inverse

2. Eine Abhängigkeit in Richtung der parallelen Dimension j ist nicht zulässig.

4. Beide Abhängigkeitsvektoren zeigen vorwärts in ip-Richtung.

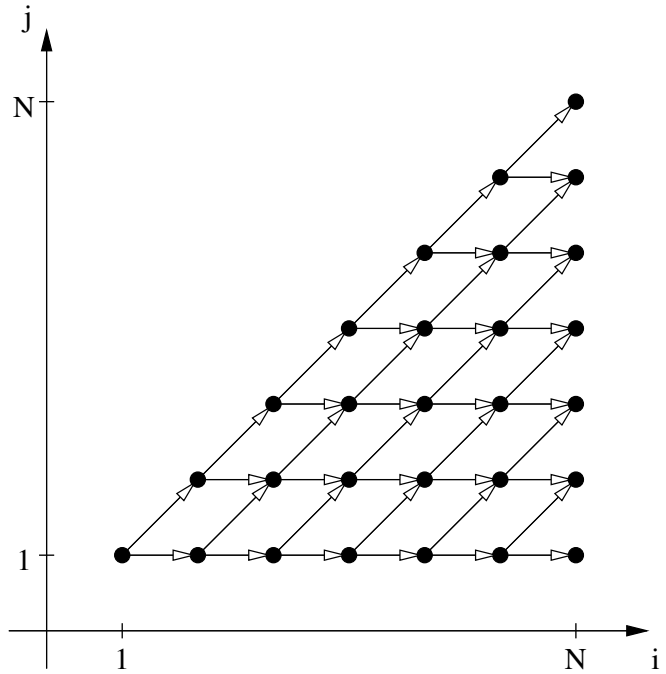
7.
```
for ip = 0 to M+N
    for jp = max (0, ip-M) to min (ip, N)
      a[jp, ip-jp] = (a[jp, ip-jp-1] + a[jp-1, ip-jp]) / 2;
```

© 2000 bei Prof. Dr. Uwe Kastens

Transformation und Parallelisierung

Iterationsraum

original

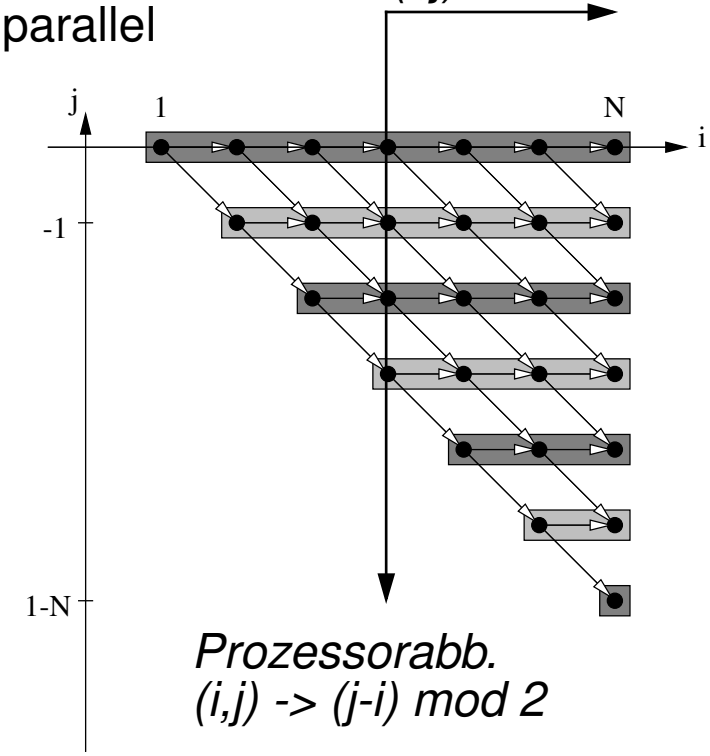



```
DECLARE B[0..N,0..N+1]

FOR I := 1 ..N
    FOR J := 1 .. I
        B[I,J] :=
            B[I-1,J]+B[I-1,J-1]
    END FOR
END FOR
```

transformiert
parallel

Zeitabb.
(i,j) -> i



Prozessorabb.
(i,j) -> (j-i) mod 2

```
DECLARE B[0..N,0..N+1]

FOR I := 1 ..N
    FOR J := 1-I.. 0
        B[I,J+I] :=
            B[I-1,J+I]+B[I-1,J-1+I]
    END FOR
END FOR
```

Datenverteilung

Ziel: Array-Elemente so auf Prozessoren verteilen, dass möglichst viele der **Zugriffe lokal** sind.

Indexraum eines Arrays: n-dimensionaler Raum von ganzzahligen Indexpunkten (Polytop)

- **gleiche Eigenschaften wie Iterationsraum**
- gleiches mathematisches Modell
- gleiche **Transformationen** anwendbar (Skewing, Reversal, Permutation, ...)
- **keine Restriktionen** durch Datenabhängigkeiten

© 2000 bei Prof. Dr. Uwe Kastens

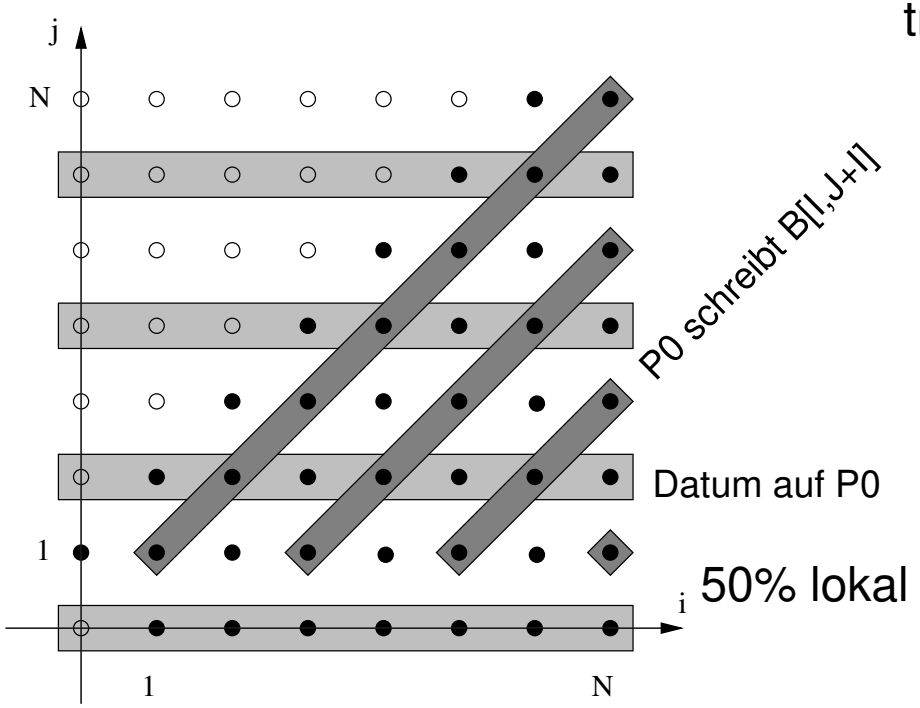
Datenverteilung für parallele Schleifen

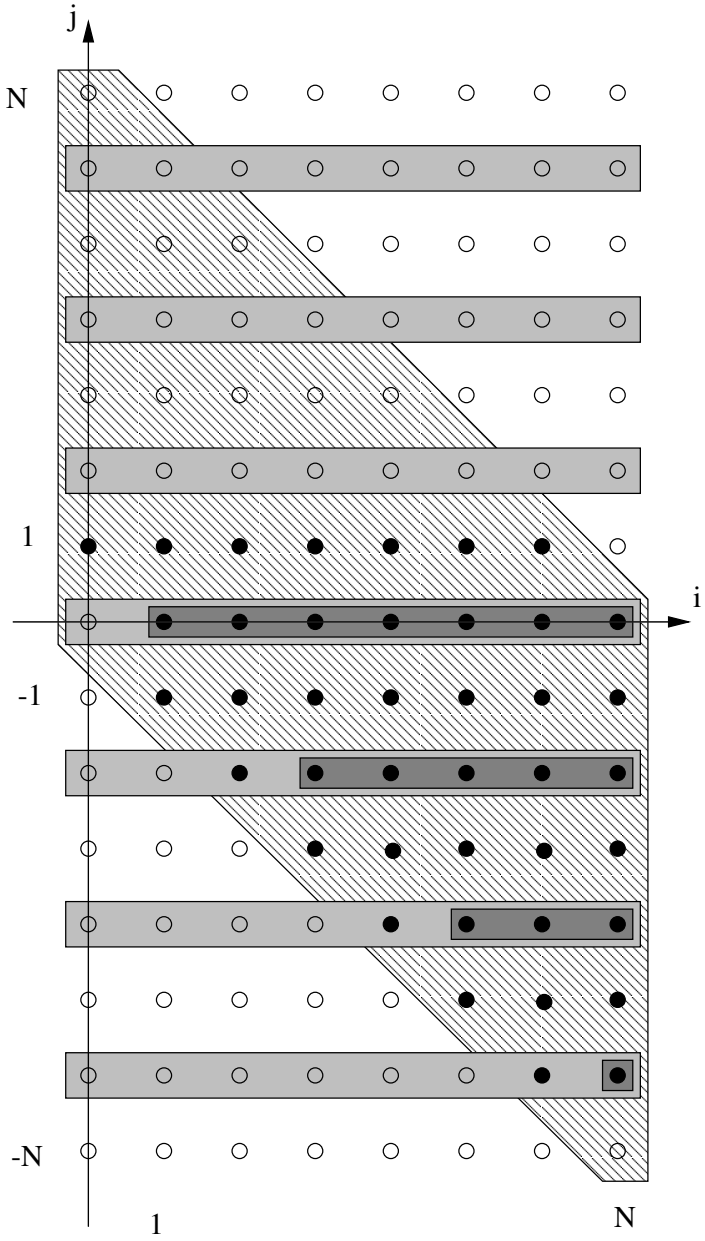
original

Indexraum

transformiert

Skewing
(i,j) -> (i,j-i)






```
DECLARE B[0..N,0..N+1]

FOR I := 1 ..N
    FOR J := 1-I.. 0
        B[I,J+I] :=
            B[I-1,J+I]+B[I-1,J-1+I]
    END FOR
END FOR
```

```
DECLARE B[0..N,-N..N]
...
    B[I,J] :=
        B[I-1,J-1]+B[I-1,J]
```