Datenparallel Listenenden finden

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

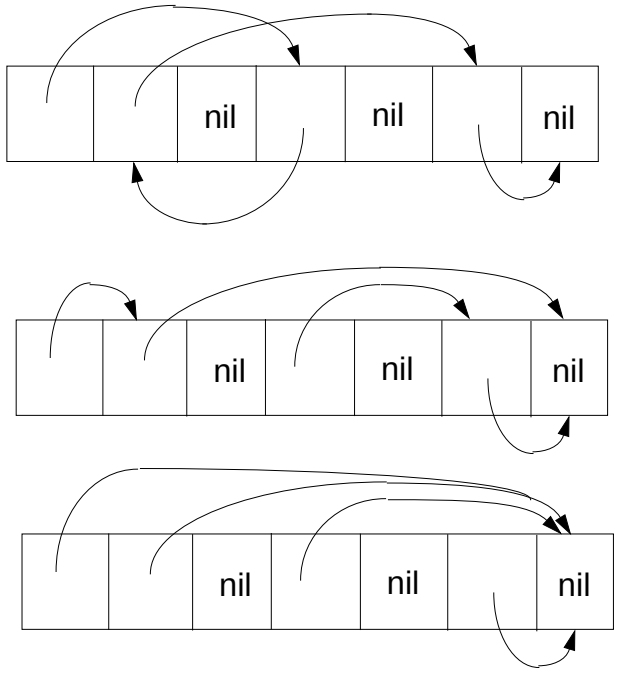
ges.: int-Array last; last[i] enthält den Index des letzten Elementes der Liste link[i]

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



Vorlesung Parallele Programmierung in Java SS 2000 / Folie 49

Ziele:

Datenparallelität nicht nur für Arrays!

in der Vorlesung:

- Paralleles Verfolgen von Listen
- Abstandsverdopplung in Runden für Listen
- last[last[i]]
- Nur nützlich, wenn das Listenende für viele Elemente gesucht wird

Verständnisfragen:

- Welche Rolle spielt der Abstand d hier?

Schleifenparallelisierung

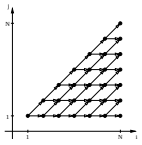
Entwicklungsschritte:

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

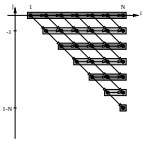
```
DECLARE B[0..N,0..N+1]

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

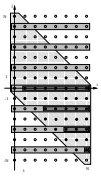
- **Datenabhängigkeiten** analysieren



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



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



Vorlesung Parallele Programmierung in Java SS 2000 / Folie 50

Ziele:

Übersicht zur Schleifenparallelisierung

in der Vorlesung:

Erläuterungen dazu

Verständnisfragen:

Iterationsraum von Schleifenschachteln

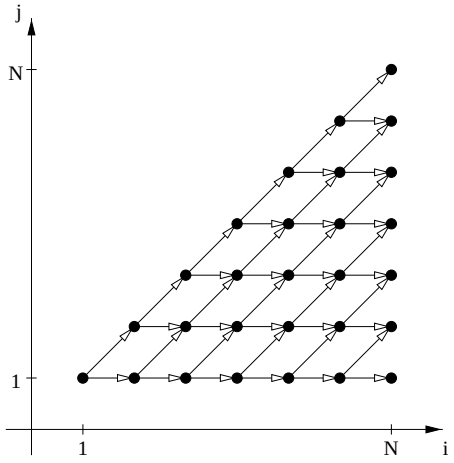
Iterationsraum n-fach geschachtelter Schleifen:

- **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
```



Vorlesung Parallele Programmierung in Java SS 2000 / Folie 51

Ziele:

Begriff des Iterationsraumes verstehen

in der Vorlesung:

- Erläuterungen am Beispiel
- Reihenfolge der Abarbeitung der Iterationspunkte zeigen
- Bei Schrittweiten größer als 1 hat der Iterationsraum Lücken (nicht konvexes Polytop)

Verständnisfragen:

- Zeichnen Sie einen Iterationsraum mit Schrittweite 3 in einer Dimension.

Datenabhängigkeiten

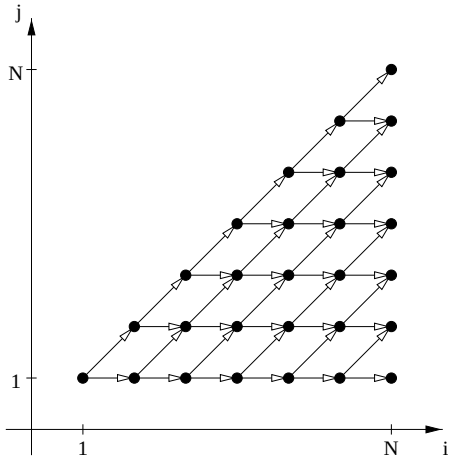
Datenabhängigkeit von Iterationspunkt i zu j:

- 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
```



Vorlesung Parallele Programmierung in Java SS 2000 / Folie 52

Ziele:

Datenabhängigkeiten in Schleifen erkennen

in der Vorlesung:

Erläuterungen dazu

- Vektordarstellung der Abhängigkeiten
- am Beispiel zeigen
- zulässige Richtungen graphisch zeigen

Verständnisfragen:

- Geben Sie einige unterschiedliche Abhängigkeitsvektoren und zugehörige Array-Zugriffe an.

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

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 53

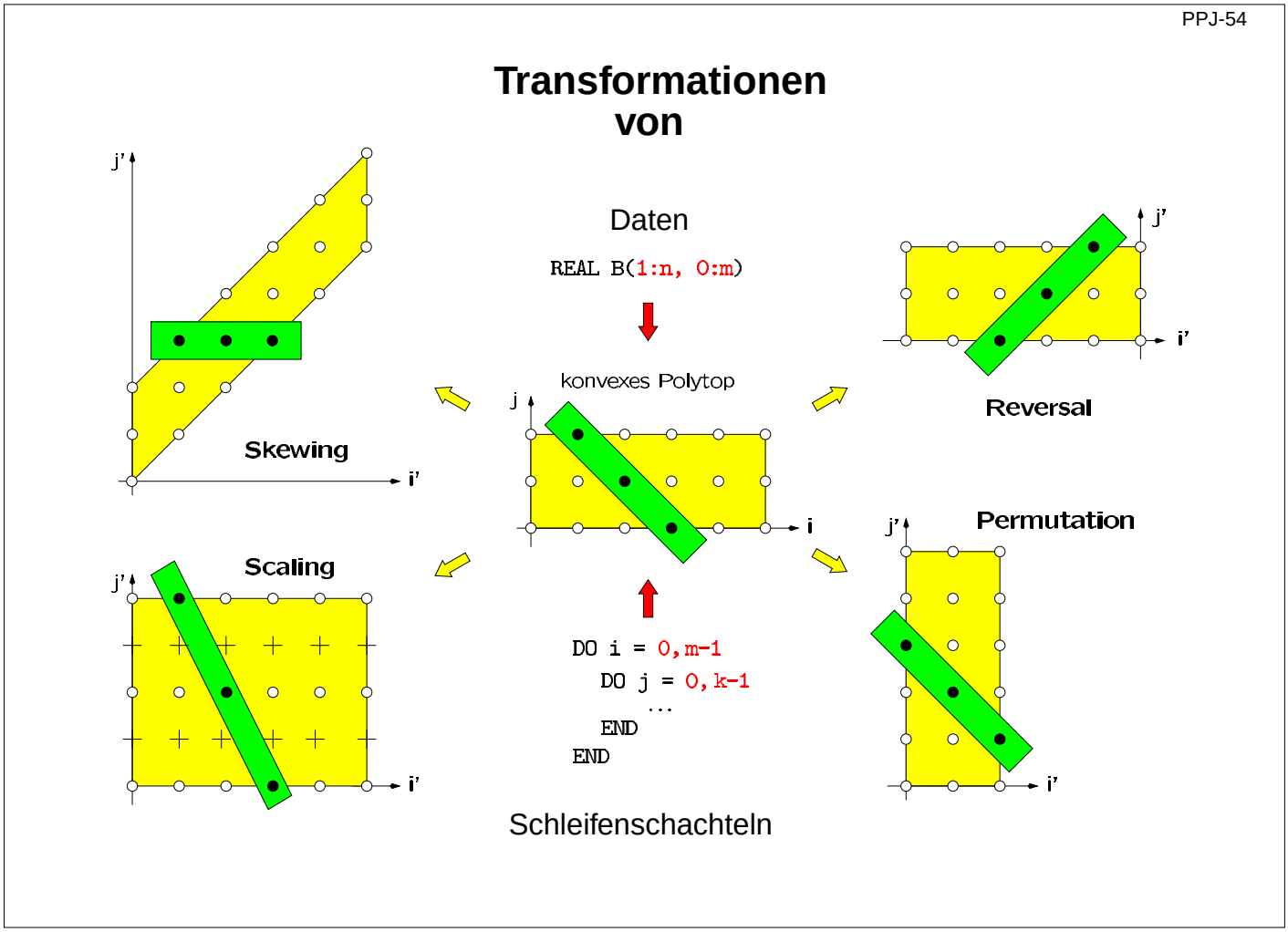
Ziele:

Überblick zu Zielen und Transformationen

in der Vorlesung:

- Erläuterung der Ziele
- zulässige Richtungen der Datenabhängigkeiten
- Transformationen an den Bildern von PPJ-54

Verständnisfragen:



Vorlesung Parallele Programmierung in Java SS 2000 / Folie 54

Ziele:

Veranschaulichung der Transformationen

in der Vorlesung:

- Erläuterungen dazu an konkreten Schleifen
- Transformation von Abhängigkeitsvektoren zeigen
- Skewing in Scaling ändern die Reihenfolge der Abarbeitung der Iterationsraumpunkte nicht; deshalb immer anwendbar.

Verständnisfragen:

- Geben Sie Abhängigkeitsvektoren zu jeder Transformation an, die auch nach der Transformation noch zulässig sind.

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

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 55

Ziele:

Matrixrepäsentation verstehen

in der Vorlesung:

- Prinzip erläutern
- konkrete Iterationspunkte abbilden
- Abhängigkeitsvektoren abbilden
- Hintereinanderausführung zeigen

Verständnisfragen:

- Geben Sie weitere Beispiele für Skewingtransformationen an

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

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 56

Ziele:

Anwendung der Matrixdarstellung verstehen

in der Vorlesung:

- Erläuterung der 4 Anwendungen an Beispielen
- Vollständige Transformation von Schleifen

Verständnisfragen:

- Wieso werden die Abhängigkeitsvektoren verändert, obwohl die Abhängigkeiten zwischen den Array-Elementen unverändert bleiben?

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.

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 56a

Ziele:

Methoden am praktischen Beispiel erproben

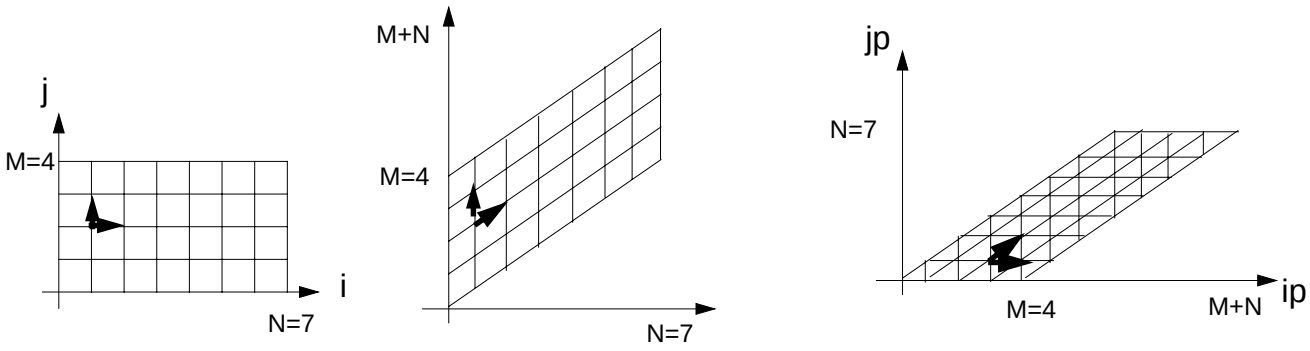
in der Vorlesung:

- Erläuterungen zu den Transformationsschritten
- Lösung auf Folie PPJ-56b

Verständnisfragen:

- Gibt es mehrere Transformationen, die zu einer parallelen inneren Schleife führen?

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;
```


Vorlesung Parallele Programmierung in Java SS 2000 / Folie 56b

Ziele:

Lösung zu PPJ-56b

in der Vorlesung:

Erläuterungen zu

- Den Grenzen der Iterationsräume,
- den Abhängigkeitsvektoren,
- der Transformationsmatrix und ihrer Inversen,
- den Anforderungen an die Parallelisierbarkeit,
- der Umrechnung der Indizierungsausdrücke.

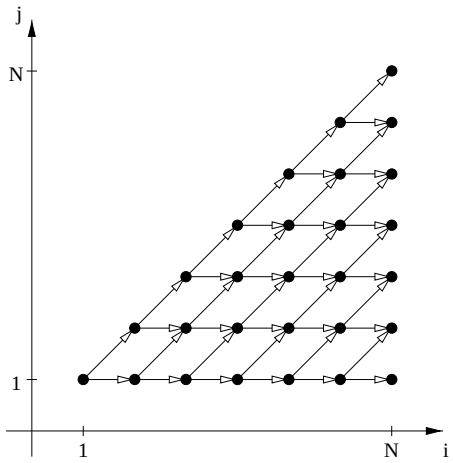
Verständnisfragen:

- Erläutern Sie das Vorgehen.

Transformation und Parallelisierung

Iterationsraum

original

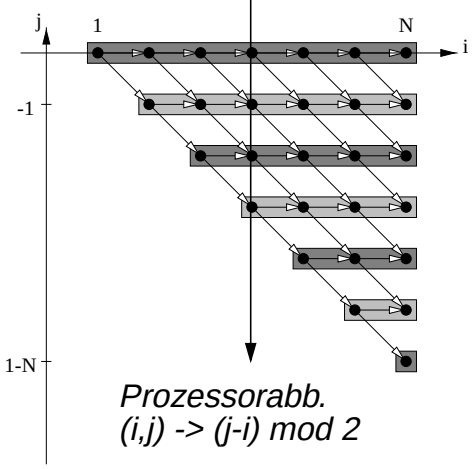



```
DECLARE B[0..N,0..N+1]

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
```

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 57

Ziele:

Parallelisierung am Beispiel

in der Vorlesung:

- Skewing-Transformation erklären
- innere Schleife parallel
- Erläuterungen zur Zeit- und Prozessorabbildung
- mod 2 faltet auf feste Anzahl von Prozessoren

Verständnisfragen:

- Wie lautet die Transformationsmatrix?
- Errechnen Sie damit die Abbildung der Indexausdrücke, der Abhängigkeitsvektoren und der Schleifengrenzen.

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

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 58

Ziele:

Gleiches mathematisches Modell für Indexräume wie für Iterationsräume

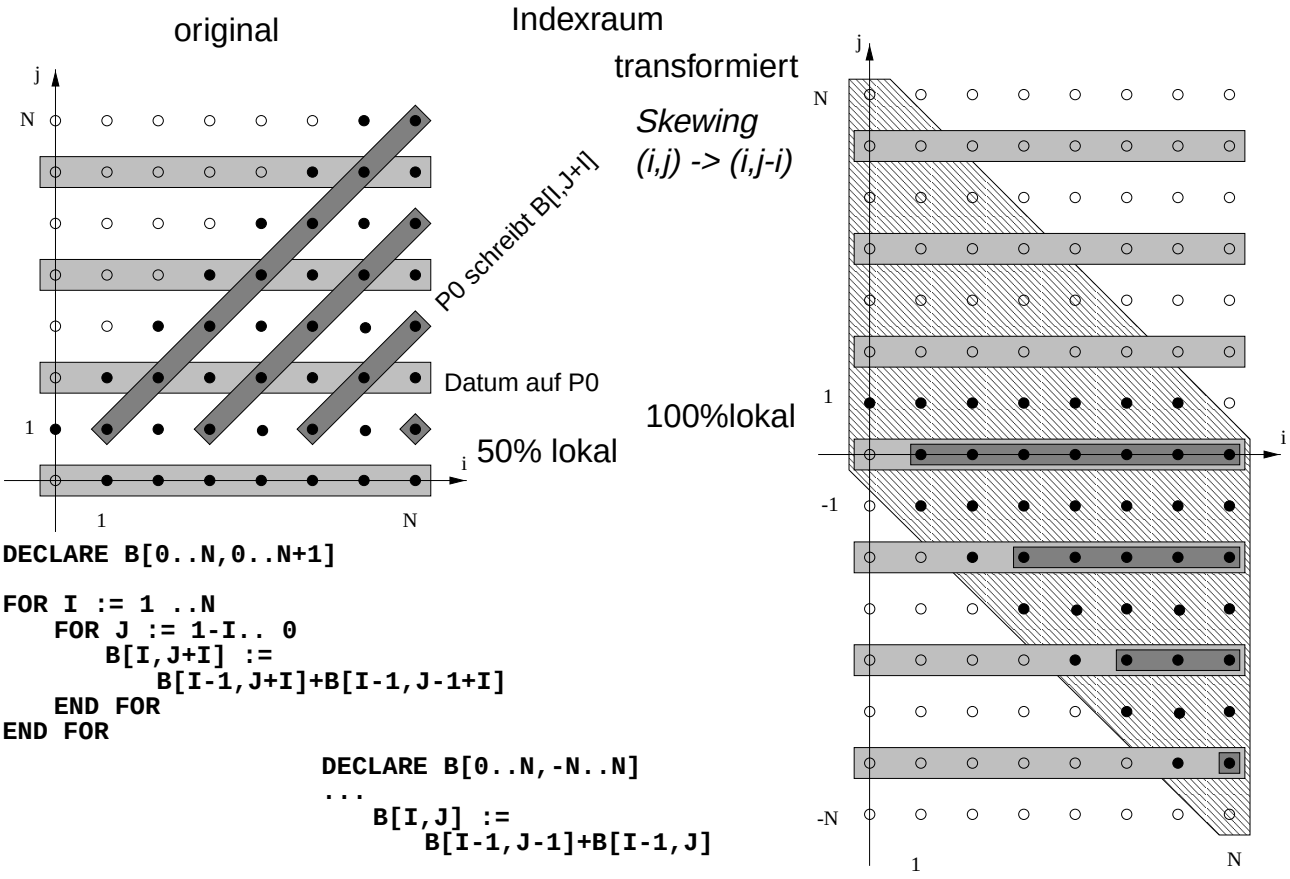
in der Vorlesung:

Erläuterungen und Beispiele zu Transformationen von Indexräumen

Verständnisfragen:

Zeichnen Sie für jede der 3 Transformationen einen Indexraum.

Datenverteilung für parallele Schleifen




```
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]
```

Vorlesung Parallele Programmierung in Java SS 2000 / Folie 59

Ziele:

Nutzen der Indextransformation erkennen

in der Vorlesung:

- lokale und nicht-lokale Zugriffe erläutern
- Indextransformation erklären
- Lokalitätsgewinn zeigen
- Speicherverschnitt durch Skewing

Verständnisfragen:

- Wie berechnen Sie die Indexabbildung mit der Transformationsmatrix?