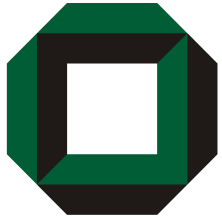
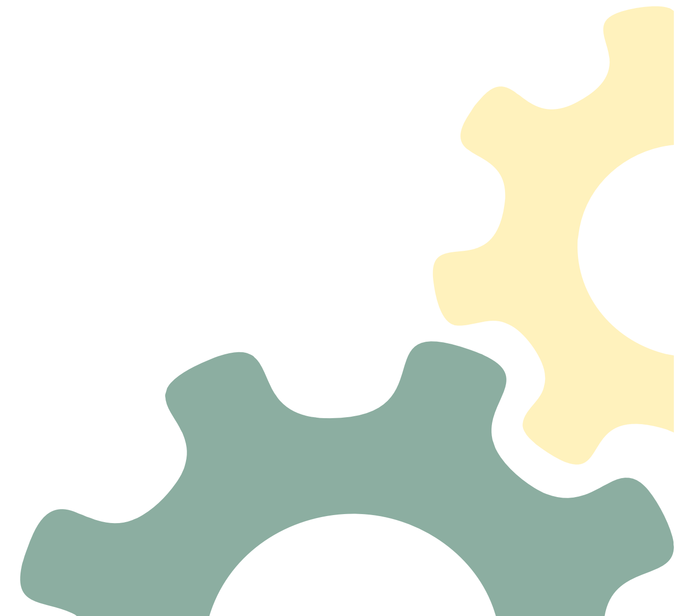


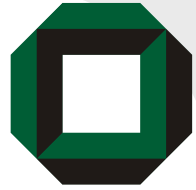
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Programmtransformationen: Vom PRAM Algorithmus zum MPI Programm

Prof. Dr. Walter F. Tichy
Dr. Victor Pankratius
Ali Jannesari

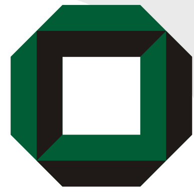
Fakultät für Informatik
Lehrstuhl für Programmiersysteme





Modell und Realität

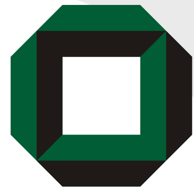
- Das PRAM Modell geht von N logischen Prozessoren aus, das Rechnerbündel hat aber lediglich $P < N$ physikalische Prozessoren → Prozessvirtualisierung notwendig.
- Das PRAM Modell geht von gemeinsamem Adressraum aus, das Rechnerbündel verfügt jedoch über einen verteilten Speicher → Datenverteilung notwendig.



Datenverteilung (1)

- Problem: Wie verteile ich meine (globalen) Daten auf die Speichermodule der einzelnen Prozessoren ?
- PRAM Ansatz: bei einem N-elementigen Vektor V[0 … N-1] hält jeder Prozessor i das Datenelement V[i].

Prozessor	0	1	2	3	4	5	6	7
V	0	1	2	3	4	5	6	7



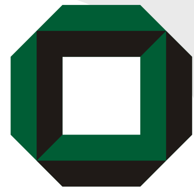
Datenverteilung (2)

- Was tun bei P<N Prozessoren?
  - (1) Zyklische Datenverteilung

Prozessor	0	1	2	3
V	0	1	2	3
	4	5	6	7

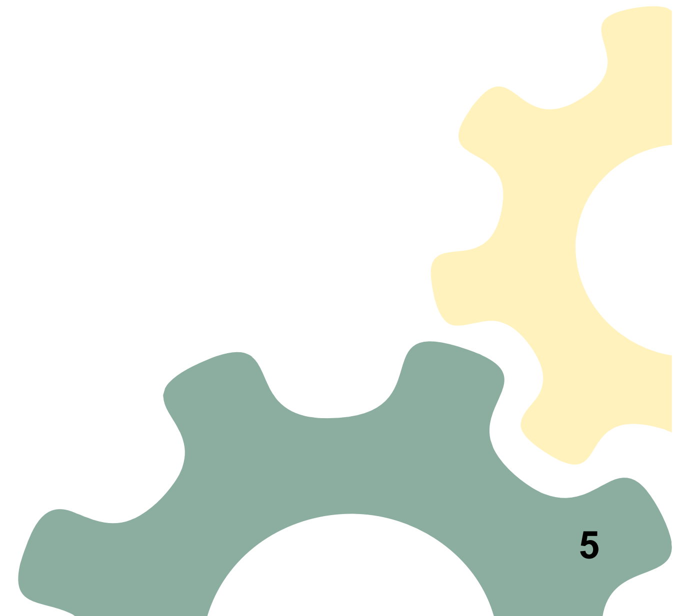
  - (2) Blockweise Datenverteilung

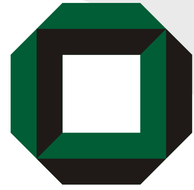
Prozessor	0	1	2	3
V	0	2	4	6
	1	3	5	7



Datenverteilung (3)

- Zyklische Datenverteilung:
  - Jeder Prozessor j bekommt alle Datenelemente i, für die gilt (i mod P) = j.
- Blockweise Datenverteilung:
  - Jeder Prozessor j bekommt die Datenelemente in Segmenten der Länge $\lceil N/P \rceil$ (bis evtl. auf den letzten):
    $V[j*\lceil N/P \rceil \ldots (j+1)*\lceil N/P \rceil -1]$.

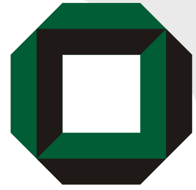




Datenverteilung (4)

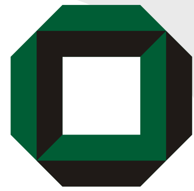
- Gibt es weitere Verteilungsstrategien?
  - Ja, nämlich die zufällige Verteilung (Random) und
  - die blockzyklische Verteilung (Bcycle(k)), bei der jeweils Datenblöcke der Länge k zyklisch über die vorhanden Prozessoren verteilt werden.

Prozessor	0	1	2	3
V	0,1	2,3	4,5	6,7
	8,9	10,11	12,13	14,15



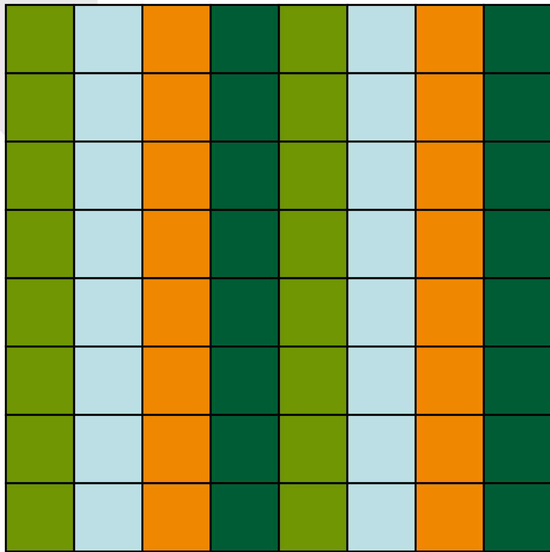
Datenverteilung (5)

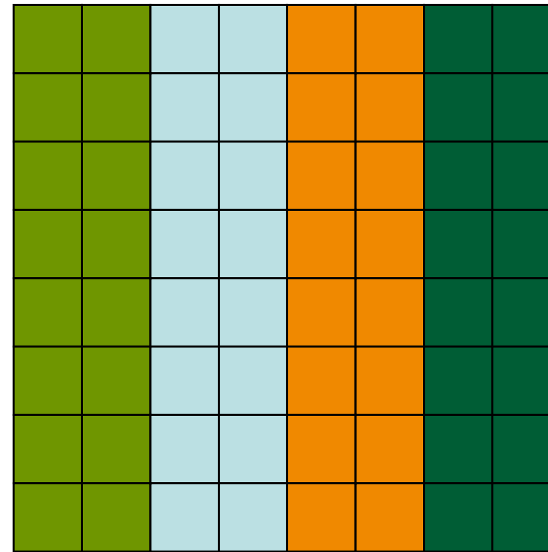
- Bcycle(k) ist die allgemeine Verteilungsstrategie, da
  - Zyklisch = Bcycle(1)
  - Block = Bcycle(N/P).
  - Was macht Bcycle(N)?
- Frage: Was tun bei mehrdimensionalen Feldern?
  - Je eine Bcycle(k) Verteilung pro Dimension!



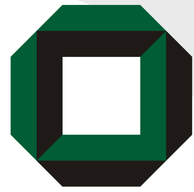
Datenverteilung (6)

- 2D Spaltenweise: Bcycle(N)/Bcycle(1) (64 Elemente auf 4 Prozessoren)



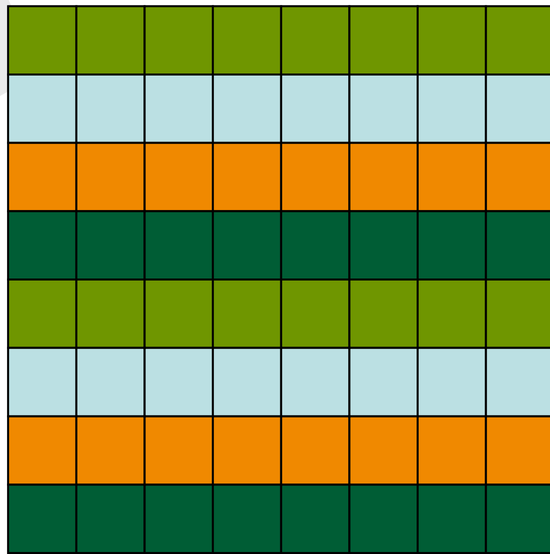


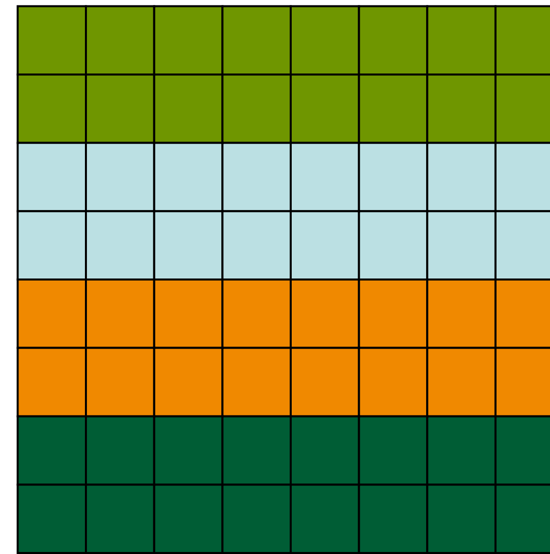
- oder Bcycle(N)/Bcycle(2).



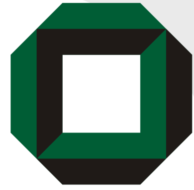
Datenverteilung (7)

- 2D Zeilenweise Bcycle(1)/Bcycle(N)
  (64 Elemente auf 4 Prozessoren)



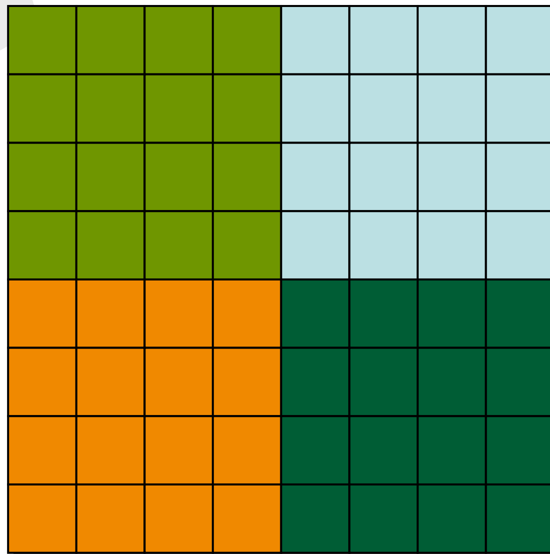


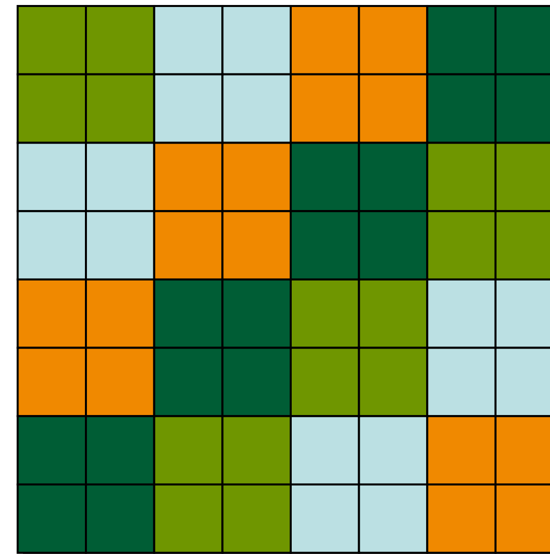
- oder Bcycle(2)/Bcycle(N).



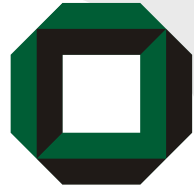
Datenverteilung (8)

- 2D Quadrantenweise (Bcycle(N/SQRT(P))/Bcycle(N/SQRT(P))) (64 Elemente auf 4 Prozessoren)



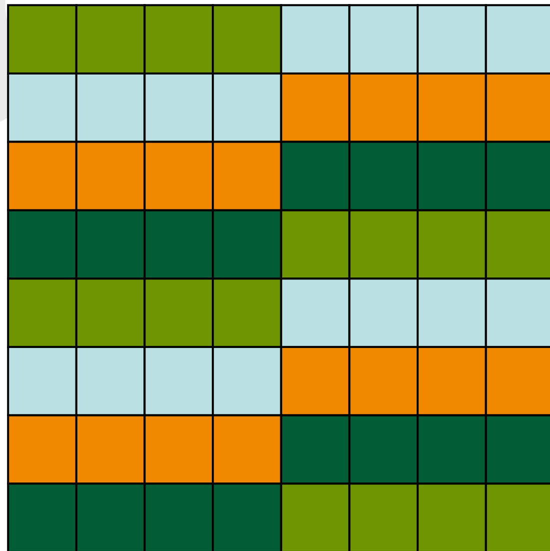


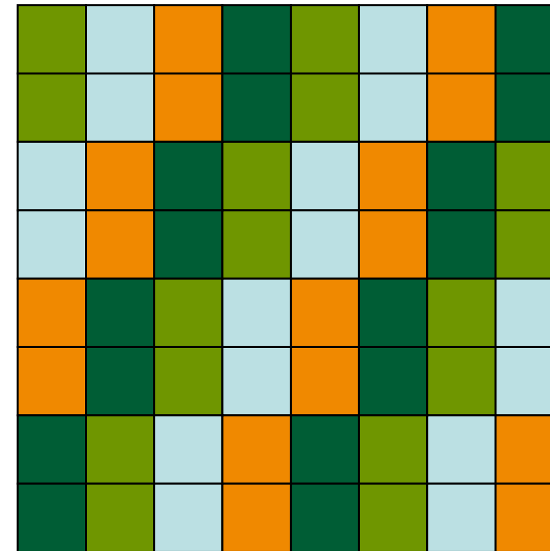
- oder Bcycle(2)/Bcycle(2).



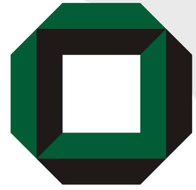
Datenverteilung (9)

- Bcycle(1)/Bcycle(4)
  (64 Elemente auf 4 Prozessoren)



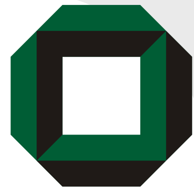


- oder Bcycle(2)/Bcycle(1).



Daten- und Prozessverteilung

- Wir wissen jetzt, welche Daten auf welchen Prozessoren liegen (Datenverteilung).
- Jetzt müssen die N Prozesse auf die P Prozessoren verteilt werden (Prozessverteilung).

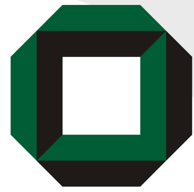


Prozessverteilung (1)

- Rückblick: einfaches PRAM Programm

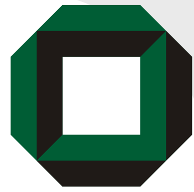
```
FORALL i [0 .. N-1] IN PARALLEL DO
  A[i] := B[i] + C[i]
END
```

- Jeder gedachte Prozess/Prozessor i ist für die Berechnung des i-ten Datenelements zuständig… oder: Datenelement i wird von Prozess / Prozessor i berechnet.



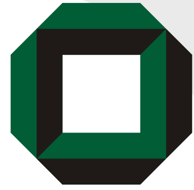
Prozessverteilung (2)

- Aus der Datenverteilung kennen wir den physikalischen Prozessor j, der Datenelement i hält!
- Idee: Lege gedachten Prozess/Prozessor i ebenfalls auf physikalischen Prozessor j. (Denn dort liegen ja die benötigten Daten.)
- → Das ist die sog. *Owner-Computes*-Regel (aus Übersetzerbau für parallele Programmiersprachen).



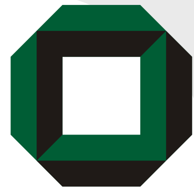
Prozessverteilung (3)

- Bei gegebener Datenverteilung unter Zuhilfenahme der *Owner-Computes*-Regel, kann man die Prozessverteilung analog zur Datenverteilung berechen (bzw. wenn man das eine festlegt, folgt das andere automatisch).



Programmtransformation

- Systematik:
  - Festlegen der Datenverteilung.
  - Emulation von $V = \lceil N/P \rceil$ virtuellen Prozessoren auf je einem physikalischen Prozessor (sog. Virtualisierung).
  - Umwandeln von nichtlokalen Datenzugriffen in Send / Receive Operationen.



Virtualisierung (1)

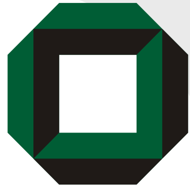
- Emulation von N virtuellen Prozessoren auf P realen Prozessoren.

Beispiel:

```
FORALL i : [0 .. N-1]
       A[i] := B[i] + C[i]
```

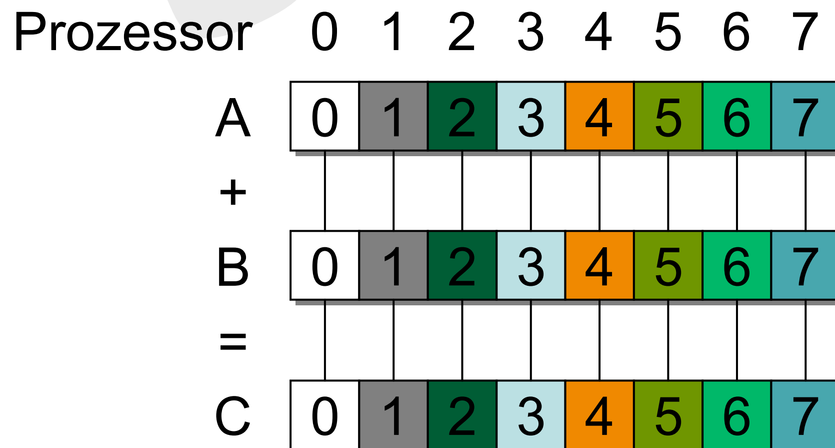
wird zu

```
FORALL i : [0 .. P-1]
  FOR v : [0 .. (N/P)-1]
      A'[v] := B'[v] + C'[v]
```

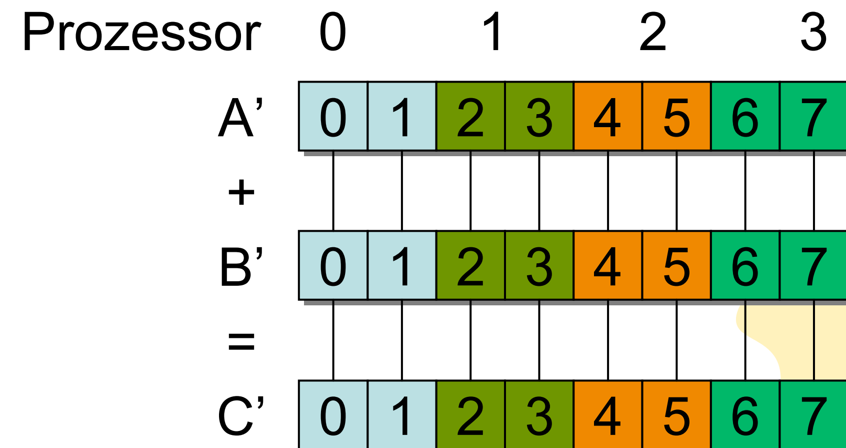


Virtualisierung (2)

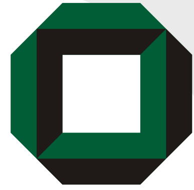
Original PRAM Programm (8 Prozessoren)



Programm nach Virtualisierung (4 Prozessoren)

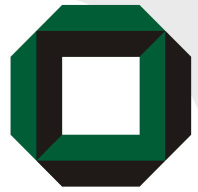


- A', B' und C' sind die verteilten Felder A, B und C



Virtualisierung (3)

- Frage: Ist die Emulation mittels einer einfachen FOR-Schleife zulässig?
- Antwort: Ja bei asynchroner Ausführung!
  Die ursprünglichen Prozessoren der PRAM arbeiten vollkommen unabhängig voneinander.
  → Diese Eigenschaft überträgt sich auf die Virtualisierungsschleife, deren einzelnen Iterationen ebenfalls unabhängig sind.
- Ist das immer der Fall?
- Was passiert bei synchroner Ausführung?

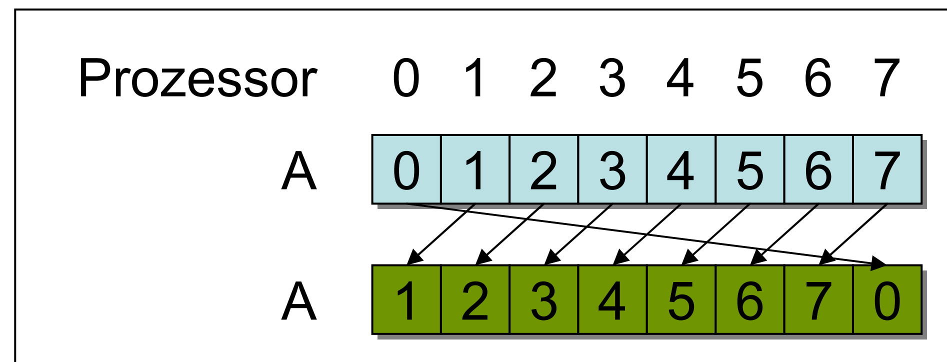


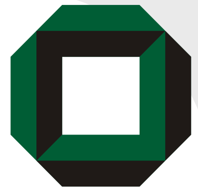
Virtualisierung und Kommunikation (1)

- Bisher haben wir Datenverteilung bei der Virtualisierung nicht betrachtet.

  Was passiert aber bei folgenden Beispiel?

```
FORALL i : [0 .. N-1] IN SYNC
  A[i] := A[(i+1)mod N]
```



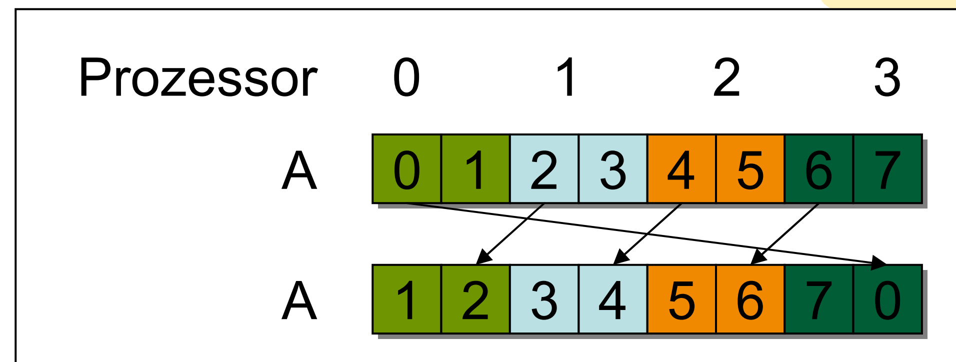


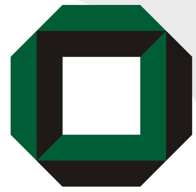
Virtualisierung und Kommunikation (2)

- Blockweise Datenverteilung

```
forall i : [0 .. P-1] in parallel
    v := (N/P);
    ziel := (i-1) mod P;
    quelle:=(i+1) mod P;
    tmp := A[0]; // lokale Indizierung
    for j : [0 .. v-2]
         A[j] = A[j+1];  // lokale Indizes
    end
    SEND(tmp, ziel);
    A[v-1] = RECEIVE(quelle);
 end

// Synchronisation durch
// tmp und RECEIVE
```

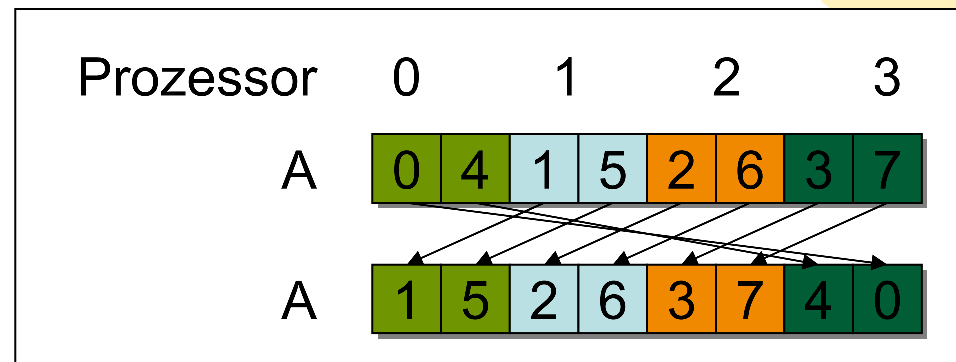


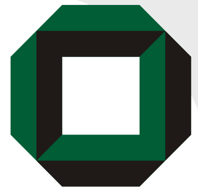


Virtualisierung und Kommunikation (3)

- Zyklische Datenverteilung

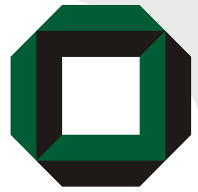
```
forall i : [0 .. P-1] in parallel
   v := (N/P);
   ziel := (i-1) mod P;
   quelle := (i+1) mod P;
   for j : [0 .. v-1]
      if i = 0 then SEND(A[(j+1) mod v], ziel);
                 else SEND(A[j], ziel);
      T[j] := RECEIVE(quelle);
    end
    for j : [0 .. v]
      A[j] := T[j];
  end
end
```





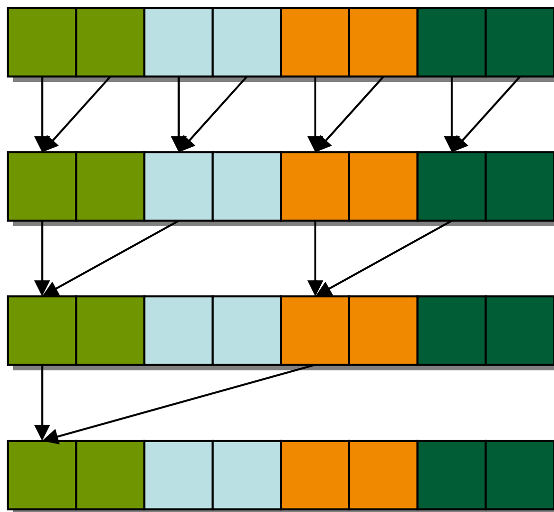
Virtualisierung und Kommunikation (4)

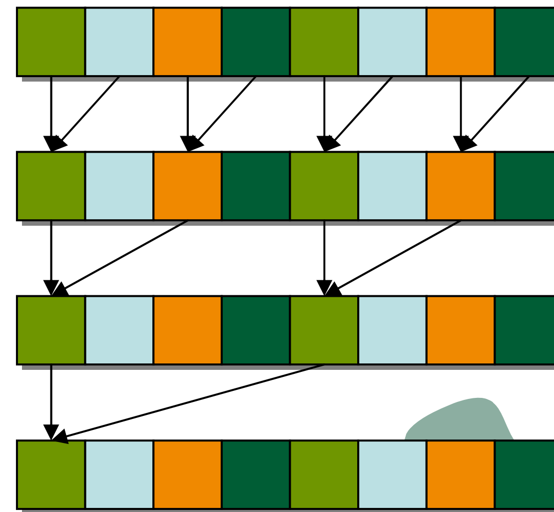
- Komplexität des Programms sowie Anzahl und Art der Kommunikationsoperationen hängt direkt von der Datenverteilung ab!
- → eine „gute“ Datenverteilung ist ein entscheidendes Kriterium für die Verständlichkeit des resultierenden Programms. Deswegen: VORHER über die Datenverteilung nachdenken ☺
- Insbesondere wichtig: Indextransformationen, Unterscheidung zw. lokalen und nichtlokalen Zugriffen, Umwandlung von synchroner Ausführung in asynchrone mit Hilfsvariablen und Synchronisation durch Kommunikation.

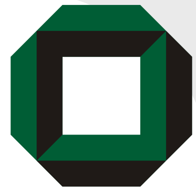


Virtualisierung und Kommunikation (5)

- Beispiel: Reduktions-, Präfix- und Postfixoperationen
- Frage: Was ist hier die geschickteste Datenverteilung? Block oder Zyklisch?





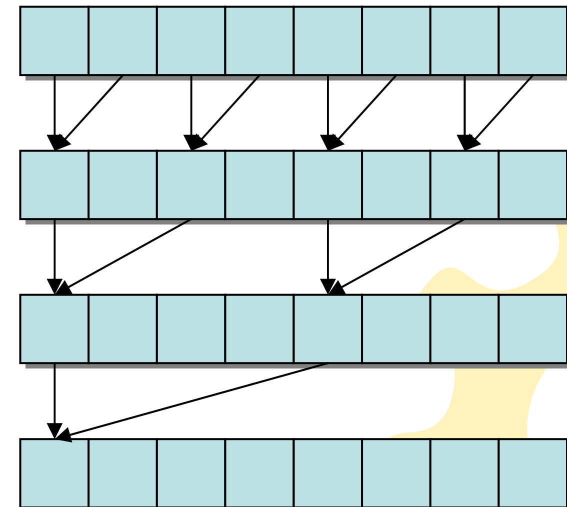


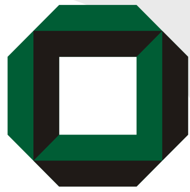
Virtualisierung und Kommunikation (6)

- Reduktionsoperation (Summe)

```
forall i: [0 .. P-1] in parallel
  int summe := 0;  // lokale Deklaration!
  for j : [0 .. (N/P)-1]
    summe := summe + V[j];
    // lokale Summe
  end
  int spanne := 1;
  while (spanne < P) DO
    if (i MOD (2*Spanne) = 0)
      // Empfänger
      summe := summe + RECEIVE(i+spanne);
    if (i MOD (2*Spanne) = Spanne)
      // Sender
      SEND(summe, i-spanne));
    Spanne := Spanne * 2;
  end
end
// Synchronisation implizit durch Kommunikation
```

Anm: Jedes Rechteck ist selbst ein Vektor



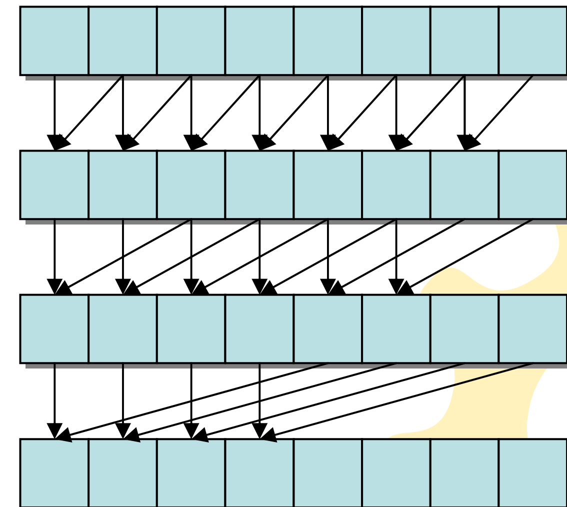


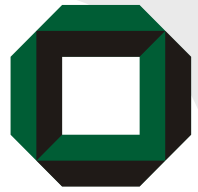
Virtualisierung und Kommunikation (7)

- Postfixoperation (Summe)

```
forall i: [0 .. P-1] in parallel
  spanne := 1;
  sum := 0; lokaleSum :=0;
  for j : [0 .. (N/P)-1]
      lokaleSum := lokaleSum + V[j];
  end
  sum := lokaleSum;
  while (spanne < N) DO
      if (i >= spanne) // Sender;
         SEND(sum, i-spanne);
      if (i + spanne < P) // Empfänger;
         sum := sum + RECEIVE(i+spanne);
      spanne := spanne * 2;
   end  //while
   V[(N/P)-1] := V[(N/P)-1] + sum-lokaleSum;
   for j : [(N/P)-2 .. 0] Step -1
        V[j] := V[j] + V[j+1];
  end //for
end //forall
```

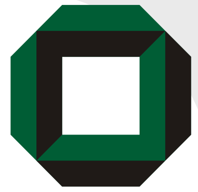
Anm: Jedes Rechteck ist selbst ein Vektor





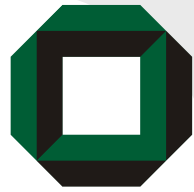
Mehrdimensionale Kommunikation (1)

- Bei mehrdimensionalen Feldern verkompliziert sich die Kommunikationsberechnung, da zusätzliche eine Abbildung der 2D-Datentopologie auf die eindimensionale Nummerierung der Prozessoren stattfinden muss.
- Vereinfachung: Verwende virtuelle Topologien in MPI. Dann ist Kommunikation „rechts, links, unten, oben“ möglich.



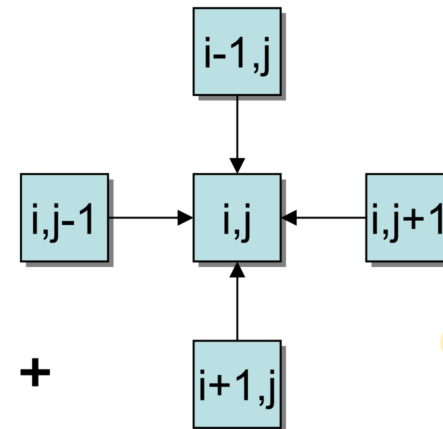
Mehrdimensionale Kommunikation (2)

- Datenverteilung ist im mehrdimensionalen Fall extrem wichtig, da sich in der Regel sog. Oberflächeneffekte ausnutzen lassen.
- Beispiel: 2D Block Verteilung (Kantenlänge k)
  → k^2 Berechungen, aber bei geeigneter
  Datenverteilung lediglich
  → 4*k Kommunikationen !!
- → Aufwandsverhältnis O(N) / 1 anstatt 1 / 1.

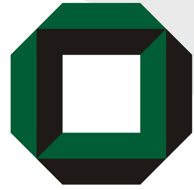


Beispiel Jacobi-Iteration (1)

- In einem N*N Feld wird folgende Iteration gerechnet, bis die Änderungen kleiner als eine vorgegebene Schranke sind:

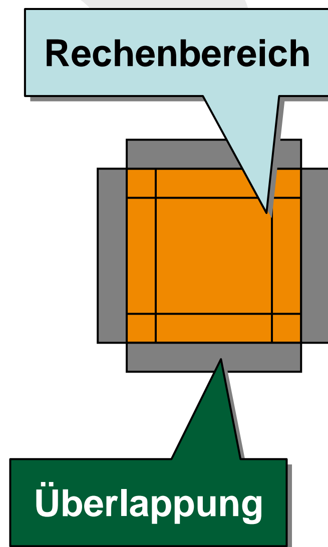


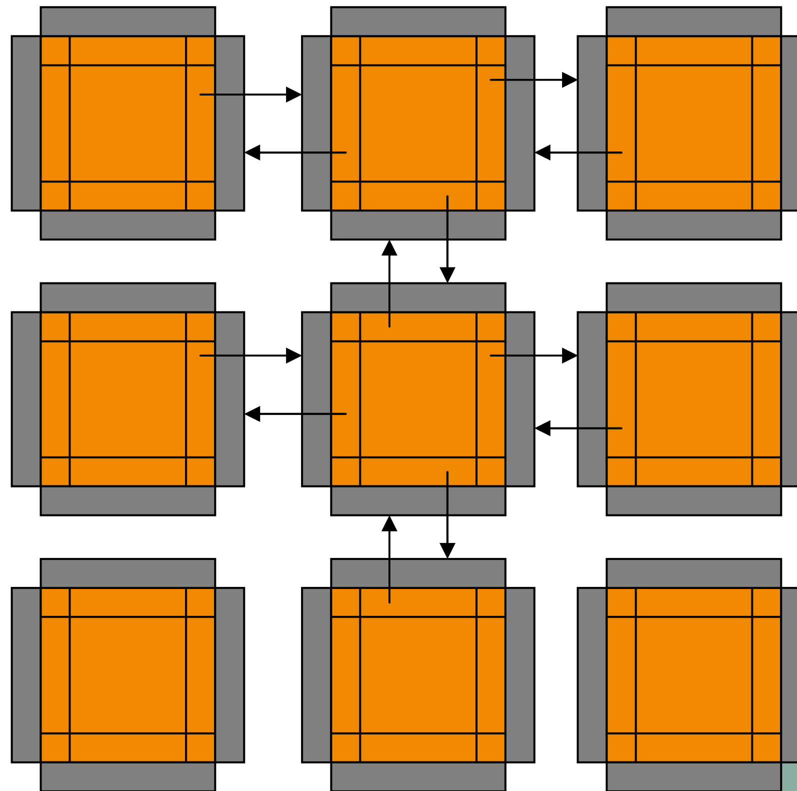

```
forall i: [1.. N-2]
  forall j: [1 .. N-2]
    A[i, j] += theta/4 *
      (A[i, j+1] + A[i, j-1] +
       A[i-1, j] + A[i+1, j]);
  end
end
```

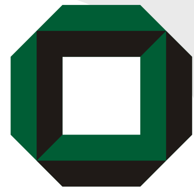


Beispiel Jacobi-Iteration (2)

- Idee: Feld in Blöcke aufteilen, mit Rändern für die Nachbarschaft (Überlappungsbereich)







Beispiel Jacobi-Iteration (3)

- Damit besteht eine Iteration aus:
  1. Ränder austauschen
  2. Iteration auf dem lokalen Block
  3. größte Änderung pro Block bestimmen
  4. Maximum-Reduktion, um Konvergenz zu bestimmen (oder Vergleich mit Schranke mit folgender Oder-Reduktion des bool'schen Ergebnisses).
- Optimierung: nicht Einzelelemente senden, sondern jeden Rand als Block.
- Kommunikationen:4k pro k x k Block.
- Rechenoperationen: k^2 pro Block.