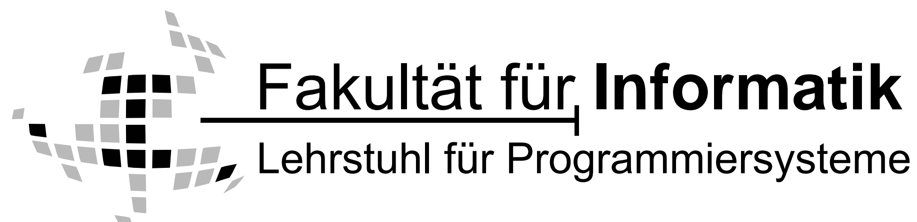
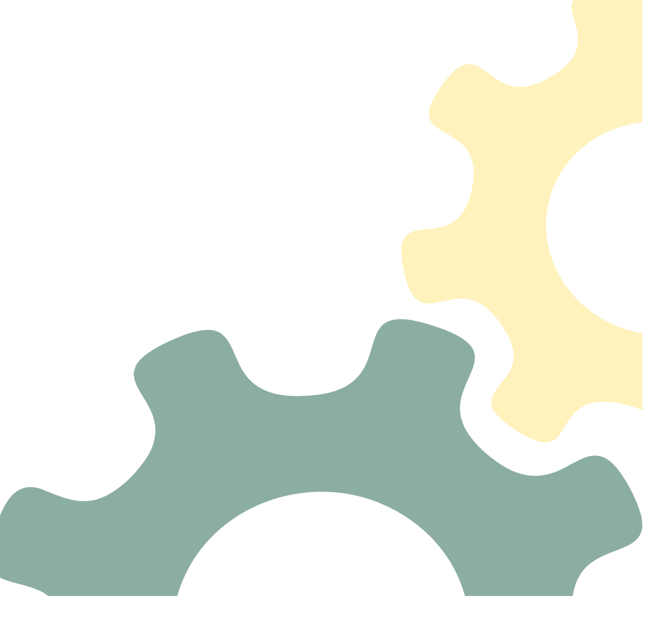
Parallele Algorithmen
Basistechniken

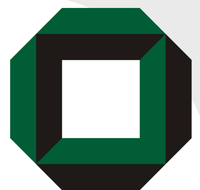
Prof. Dr. Walter F. Tichy
Dr. Victor Pankratius
Ali Jannesari





Parallel Random Access Machine (PRAM)

- Abstraktes Maschinenmodell, in dem sich parallele Algorithmen sehr gut spezifizieren lassen.
- Entspricht einer synchronen MIMD-Maschine mit gemeinsamem Adressraum.
- Speicherzugriffsvarianten
  - EREW: Exclusive Read, Exclusive Write
  - ERCW: Exclusive Read, Concurrent Write
  - CREW: Concurrent Read, Exclusive Write
  - CRCW: Concurrent Read, Concurrent Write



Probleme des PRAM-Modells

- Es gibt keine echte, synchrone MIMD-Maschine mit gemeinsamem Adressraum.
- Alle Speicherzugriffsvarianten außer EREW sind unrealistisch.
- Die implizite Annahme, dass Kommunikationsoperationen eine Zeiteinheit dauern, ist ebenfalls unrealistisch.
- ABER: Ist man sich der Unzulänglichkeiten bewusst, dann ist die PRAM ein sehr elegantes Modell zur Spezifikation paralleler Algorithmen.
- Diese können einfach für Cluster konvertiert werden.

PRAM Sprachkonstrukte (1)

- Imperative Programmiersprache mit Erweiterungen, um Parallelität zu spezifizieren.
- Asynchrones FORALL:

```
FORALL i : P IN PARALLEL
  Anweisung_1(i)
  …
  Anweisung_n(i)
END
```

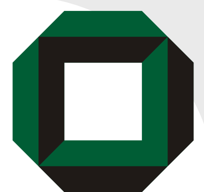
- P ist eine Menge; für jedes Element in P wird ein Prozessor eingesetzt, wobei jeder Prozessor ein unterschiedliches Element i der Menge P bekommt.
- Jeder Prozessor führt die Anweisungsfolge 1 .. n asynchron aus.
- Die FORALL-Anweisung endet, wenn jeder Prozessor die Anweisungsreihenfolge abgearbeitet hat. Zwischendurch erfolgt keine Synchronisation.

PRAM Sprachkonstrukte (2)

- synchrone Form des FORALL:

```
FORALL i : P IN SYNC
  Anweisung1(i)
  …
  Anweisungn(i)
END
```

- Wie vorher, nur dass jetzt alle Prozessoren die Anweisungsfolge synchron („im Gleichschritt“) ausführen.
- Die synchrone Ausführung gleicher Anweisungen vermeidet viele Laufzeitprobleme, da unterschiedliche Ausführungsgeschwindigkeiten der Prozessoren nicht beachtet werden müssen.

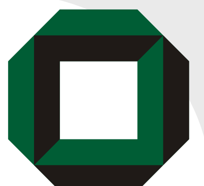


Synchrone Zuweisung

Synchrone Abarbeitung der Zuweisung:

```
L := R
```

1) Alle n Prozessoren werten L synchron aus und erhalten eine Adresse.
2) Alle n Prozessoren werten R synchron aus und erhalten einen Wert.
3) Alle n Prozessoren speichern ihren Wert an ihre Adresse (EW oder CW)
4) (1) & (2) können im Prinzip auch gleichzeitig ausgeführt werden. Seiteneffekte sind möglich, aber die Reihenfolge der Auswertungen ist undefiniert.



Synchrones IF-THEN-ELSE

Synchrone Abarbeitung der bedingten Anweisung:

```
IF B THEN S1 ELSE S2
```

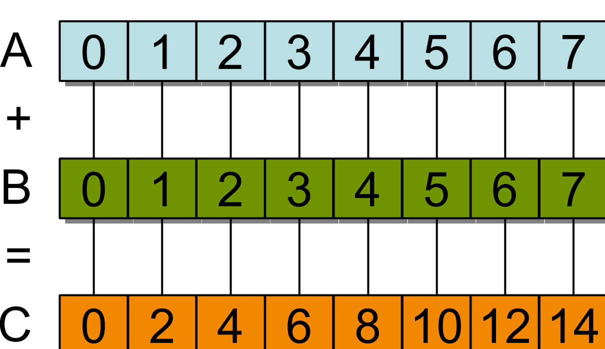
1) Alle n Prozessoren werden B synchron aus.
2) Die Menge der Prozessoren partitioniert sich in die Mengen M_T (für B=T) und M_F (für B=F) in Abhängigkeit des Ergebnisses von B.
3) Die Menge M_T führt S1 synchron aus.
4) Die Menge M_F führt S2 synchron aus.
5) Die Untermengen M_T und M_F können (müssen aber nicht) parallel abgearbeitet werden.
6) Ausführung endet, wenn die Mengen M_T und M_F beide fertig sind.

FORALL (1)

- Addition eines n-elementigen Vektors

```
FORALL i : [0 .. n-1] IN SYNC
  C[i] := A[i] + B[i];
END
```

- Bsp: für n = 8



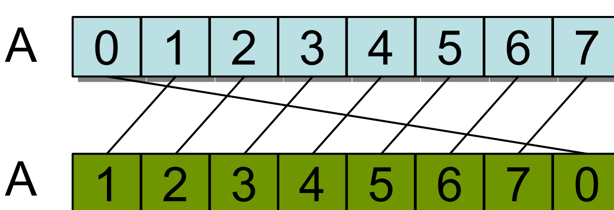

```
FORALL i : [0 .. n-1] IN PARALLEL
```
wäre ebenfalls möglich, da die Anweisungen voneinander unabhängig sind.

FORALL (2)

- Rotieren eines n-elementigen Vektors

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

- Bsp: für n = 8

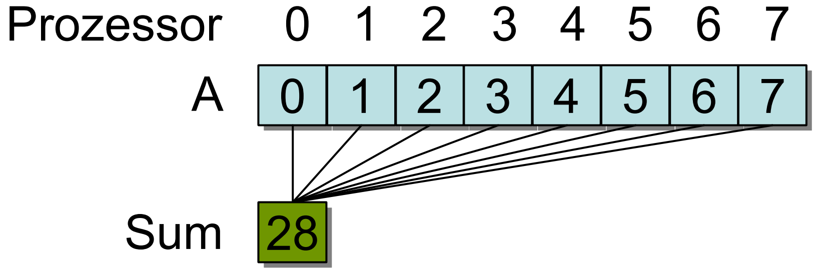



```
FORALL i : [0 .. n-1] IN PARALLEL
```

wäre nicht möglich, da dann unter Umständen falsche Werte weitergegeben werden.

Reduktion (1)

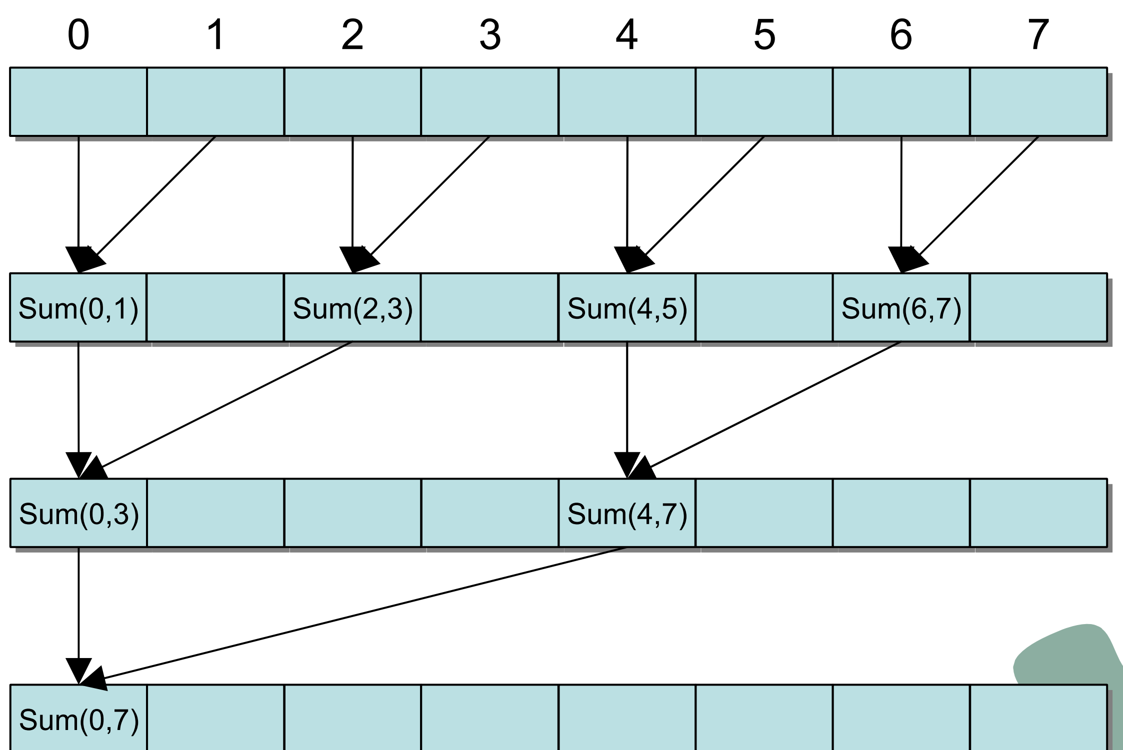
- Eine Reduktion, d.h. das Zusammenfassen mehrerer Datenelemente zu einem Ergebnis, gehört zu den Basistechniken der parallelen Algorithmen. (MPI_Reduce, MPI_Allreduce)
- Bsp: Die parallele Summenbildung

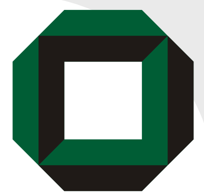


- Summenberechnung sequentiell: O(N)
- Frage: Wie geht's parallel schneller?

Reduktion (2)

- Baumartiges Reduktionsverfahren:
  Fasse jeweils zwei (im Abstand 2^k, $0 \le k < \log_2(N)$) benachbarte Elemente zusammen:





Reduktion (3)

PRAM Programm einer parallelen Summe

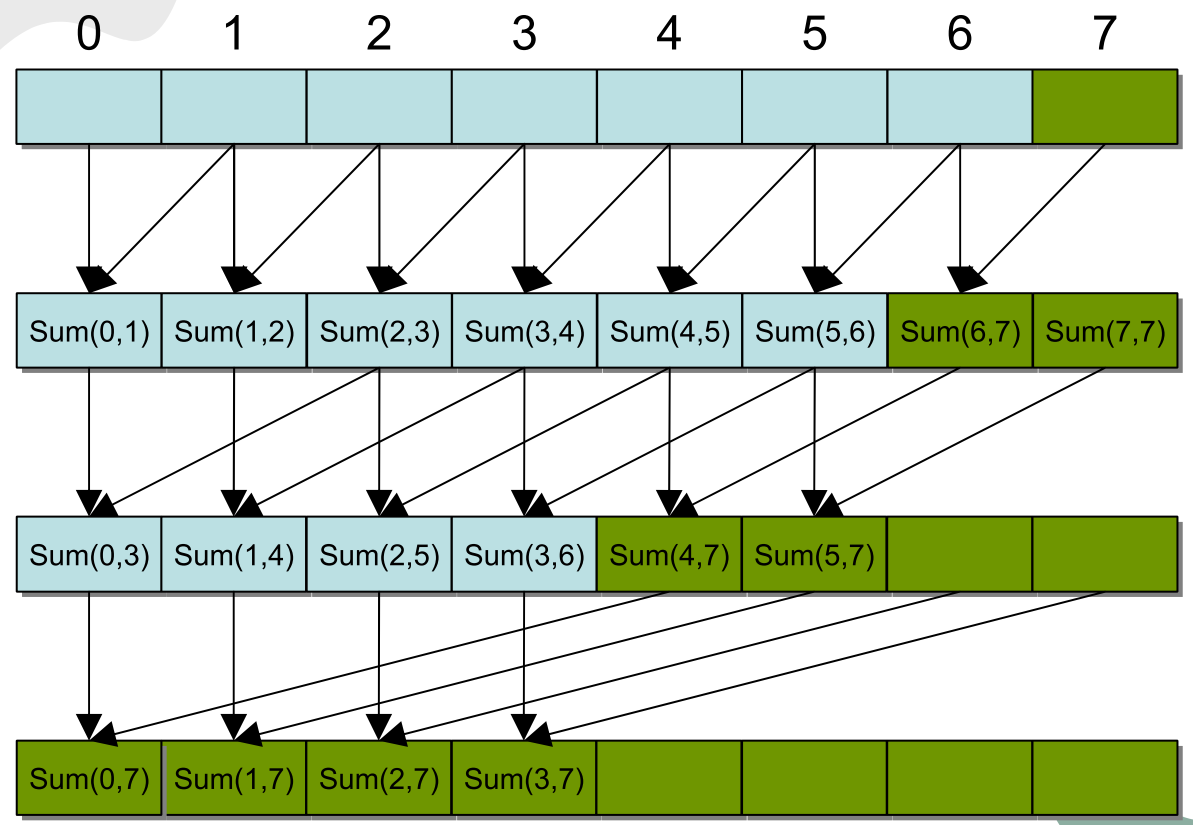
```
CONST N = ...
V : ARRAY [0 .. N-1] OF INTEGER
Spanne :      INTEGER

Spanne := 1;
WHILE (Spanne < N) DO
     FORALL   i : [0 .. N-1] IN SYNC
         IF (i MOD (2*Spanne)) = 0 AND Spanne+i < N
                 V[i] := V[i] + V[i + Spanne];
         END
      END
   Spanne := Spanne * 2;
END
```

parallele Laufzeit: $O(\log_2(N))$, Modell: CREW, aber einfach auf EREW zu konvertieren

Präfix & Postfix (1)

Berechnung aller Partialsummen (MPI_Scan)



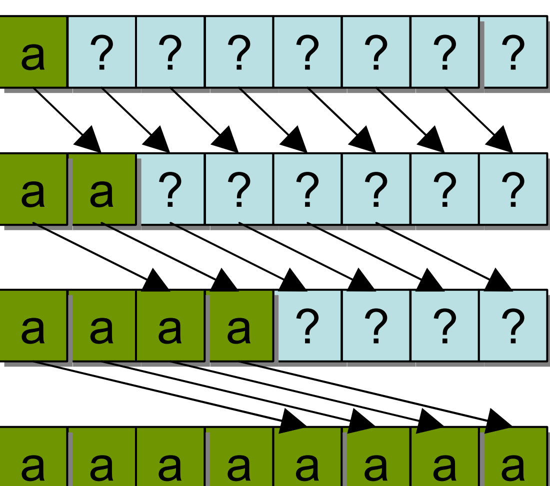
Präfix & Postfix (3)

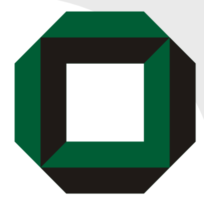
- Präfixoperation(j) berücksichtigt alle Elemente i mit $0 \leq i \leq j$.
- Postfixoperation(j) berücksichtigt alle Elemente i mit $N-1 \geq i \geq j$.
- Beispiel: Prä- und Postfixsummen

Prozessor	0	1	2	3	4	5	6	7
A	1	1	1	1	1	1	1	1
Präfix	1	2	3	4	5	6	7	8
Postfix	8	7	6	5	4	3	2	1

Broadcast (1)

Verteilen von Elementen (MPI_Bcast):





Broadcast (2)

PRAM Programm zur Datenverteilung

```
CONST N = ...
V : ARRAY [0 .. N-1] OF INTEGER
Spanne : ARRAY [0 .. N-1] INTEGER

V[0] := a;
FORALL i : [0 .. N-1] IN SYNC
   Spanne[i] := 1;
   WHILE (Spanne[i] < N) DO
       IF i >= Spanne[i]
          V[i] := V[i - Spanne[i]];
       END
       Spanne[i] := Spanne[i] * 2;
   END
END
```

parallele Laufzeit: O(log2(N)), Modell: EREW

Anwendungen (1)

- Abzählen von Elementen mit einer bestimmten Eigenschaft:

```
FORALL i:[0 .. N-1] IN PARALLEL
  T[i] := test_condition(i);
END
```

```
Präfix(T,+);
```

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

Präfix	0	1	2	3	4	5	6	7
T	0	1	2	2	3	3	4	5

- Ergebnis in T

Anwendungen (2)

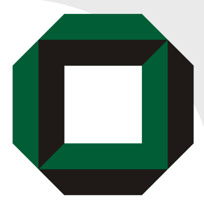
- Lösen von Rekurrenzen (1. Ordnung):

  $X_i = X_{i-1} \circ a_i$, mit $\circ$ binärer Operator

  Beispiel: Präfixsumme als Rekurenz:

  $Sum_i = Sum_{i-1} + a_i$ mit $Sum_0 = a_0$

- Lösen von Rekurrenzen höherer Ordnung auch möglich
- $X_i = a_{i-1} \circ X_{i-1} \circ \ldots \circ a_{i-m} \circ X_{i-m} \circ b_i$

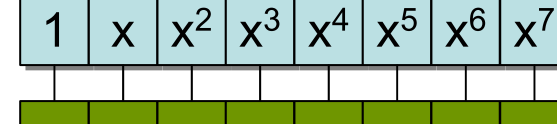


Anwendungen (3)

- Parallele Polynomauswertung:
  $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$

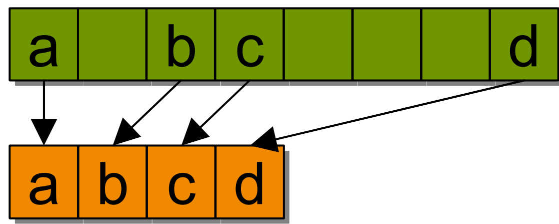
- Laufzeit:
  O(3*log(N))
  = O(log(N))

Prozessor	0	1	2	3	4	5	6	7
X	x							
Broadcast X	1	x	x	x	x	x	x	x
Präfix(X,*)	1	x	x^2	x^3	x^4	x^5	x^6	x^7
A								
T = A * X								
L_Reduce(T,+)	f(x)							



Anwendungen (4)

- Kompaktifizierung einer Liste



- Laufzeit: O(log(N))

L	a		b	c				d

```
if B(L[i]) then T[i] := 1; else T[i] := 0;
```

T	1	0	1	1	0	0	0	1
Präfix(T,+)	1	1	2	3	3	3	3	4

```
if B(L[i]) then L'[T[i]-1] := L[i]
```

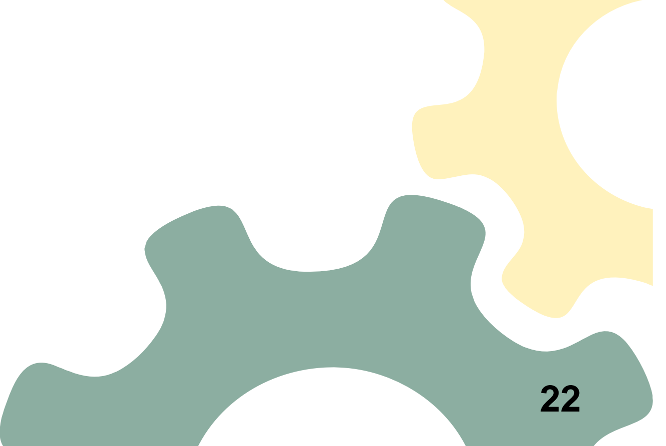
L'	a	b	c	d

Aufgabe (1)

- Gegeben sei ein N-elementiger Zahlenvektor v[0…n-1].
- Schreiben Sie ein paralleles Programm, welches $D = \max\{ v[i] - v[j] \}, 0 \leq i , j < n$ berechnet.
- Was ist die asymptotische Laufzeit?

Aufgabe (2)

- Schreiben Sie ein paralleles Programm, welches zwei Zeichenreihen der Länge N lexikographisch vergleicht.
- Was ist die asymptotische Laufzeit?



Aufgabe (3)

- Gegeben sei ein N-elementiger Zahlenvektor v[0… n-1].
- Schreiben Sie ein paralleles Programm, welches den Wert des maximalen Untervektors bestimmt:

$$U=\max\{ \sum_{i=p}^{q} v[i] \; ; \; 0 \leq p < q < n\}$$

- Was ist die asymptotische Laufzeit?

Aufgabe (4)

- Schreiben Sie ein paralleles Programm, welches die ersten N Fibonacci-Zahlen berechnet.
- $f_i = f_{i-1} + f_{i-2}$, $f_0 = f_1 = 1$