Rechnerarchitektur SS 2014

Parallel Random Access Machine (PRAM)

Michael Engel

TU Dortmund, Fakultät für Informatik

Teilweise basierend auf Material von Gernot A. Fink und R. Yahyapour

5. Juni 2014

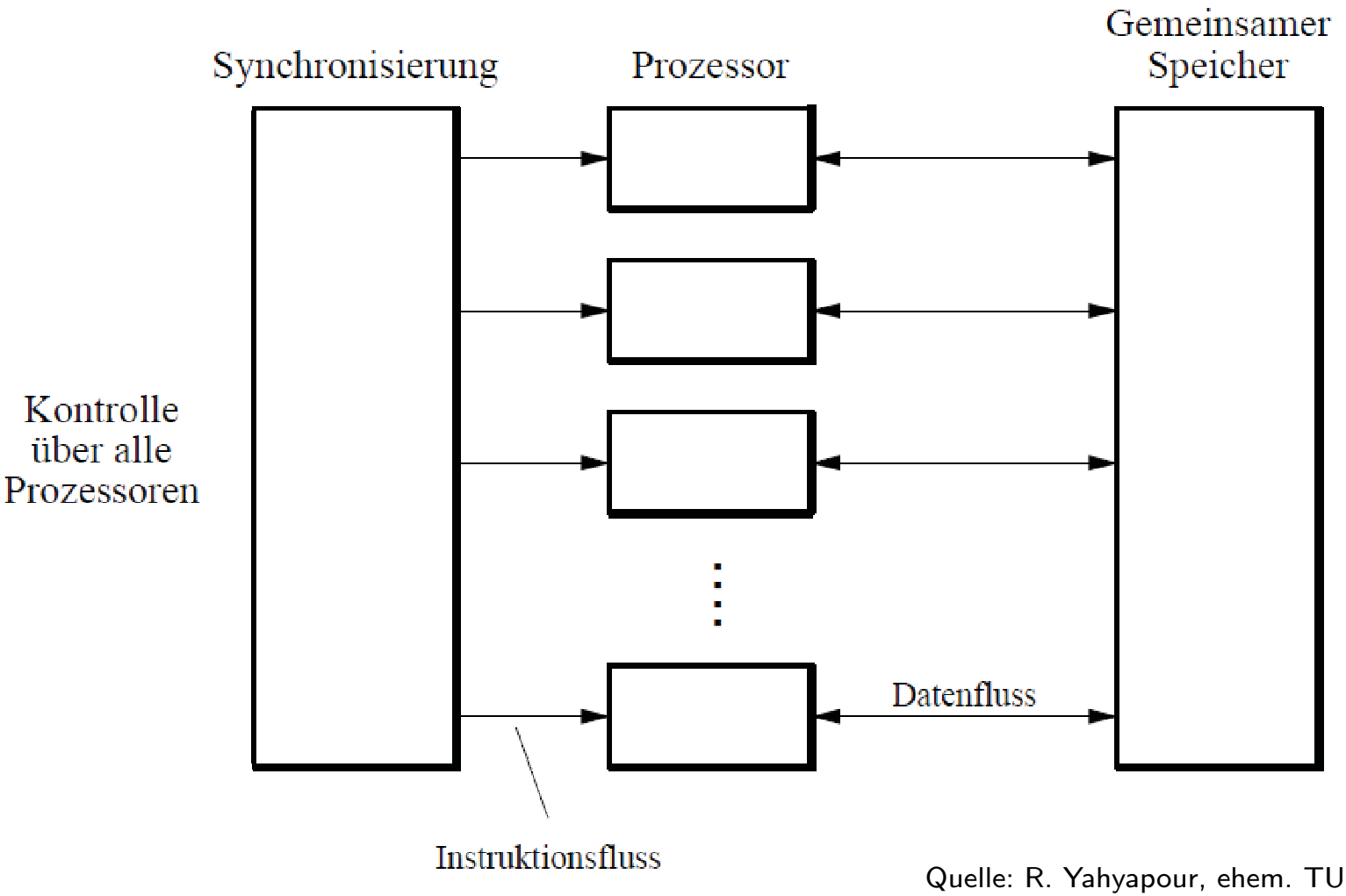
Parallel Random Access Machine (PRAM)

Maschinenmodell zur Analyse paralleler Algorithmen

Annahmen:

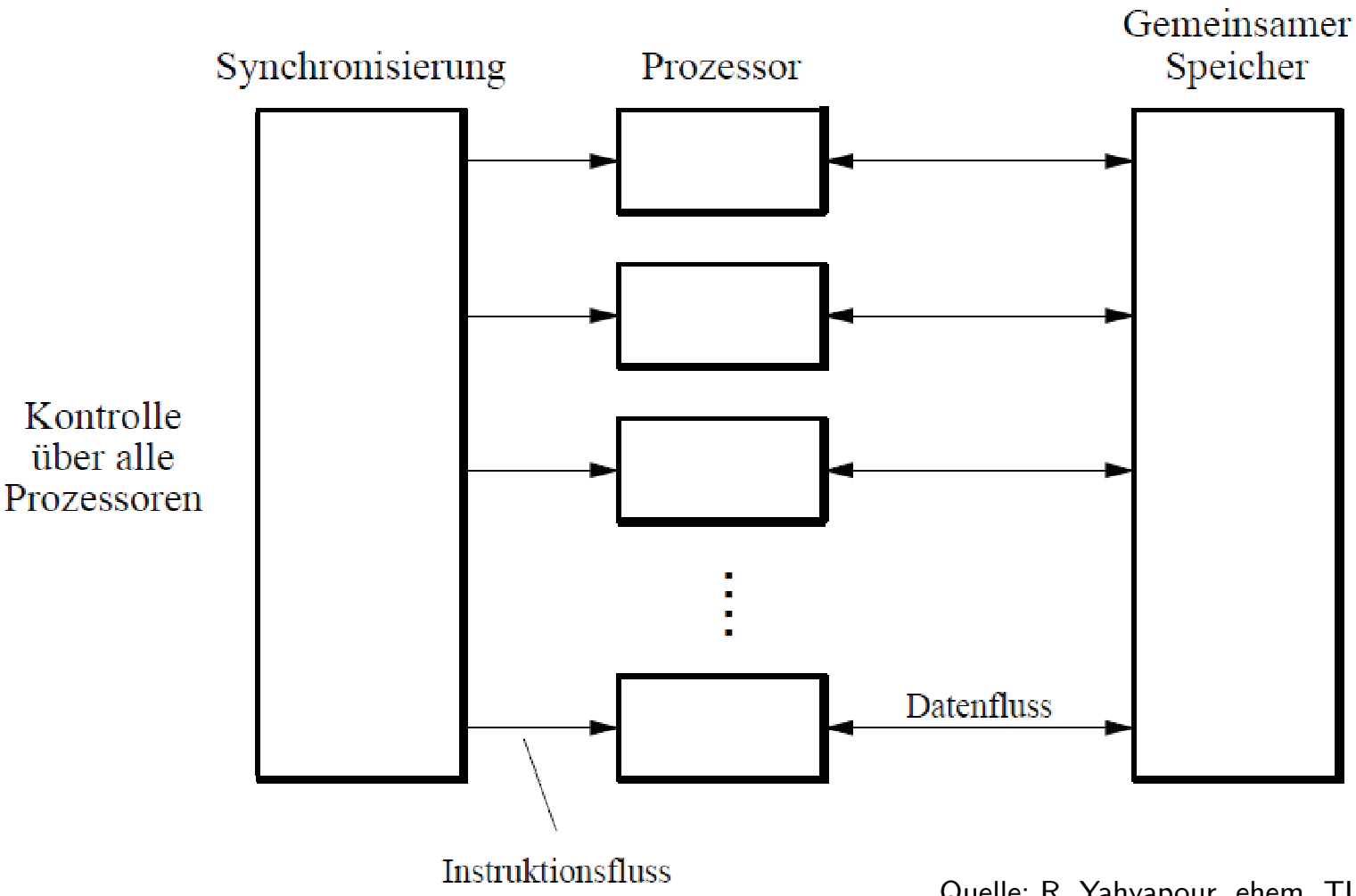
- PRAM verwendet einen gemeinsamen Speicher
- Zugriff auf jede Speicherzelle erfolgt in konstanter Zeit (von jedem Prozessor aus)
- Prozessoren kommunizieren über gemeinsamen Speicher
- Alle Prozessoren sind identisch, haben lokalen Speicher
- Operationen haben identische Ausführungszeiten
- Alle Prozessoren führen synchron *dasselbe* Programm aus
- Effekt der Operationen hängt von lokalen Daten ab

Parallel Random Access Machine (PRAM) II



Quelle: R. Yahyapour, ehem. TU Dortmund

Parallel Random Access Machine (PRAM) II



Quelle: R. Yahyapour, ehem. TU Dortmund

Kennen wir so ein ähnliches Modell schon?

PRAM: Speicherzugriff

- Gemeinsamer Speicher zentraler Aspekt der PRAM zur Kommunikation zwischen Prozessoren
- Wie parallele Zugriffe handhaben?
- Verarbeitungsphasen (ähnlich Pipelining):
  - Lesen
  - Verarbeiten
  - Schreiben

  $\Rightarrow$ Operationen der Form $A = A \circ B$ unproblematich!

PRAM: Speicherzugriff II

Parallele Zugriffe (lesend/schreibend) auf die gleiche Speicherzelle?
Mögliche *Varianten* des PRAM-Modells:

CRCW: *Concurrent Read, Concurrent Write*
gleichzeitiger Lese- und Schreibzugriff möglich

CREW: *Concurrent Read, Exclusive Write*
gleichzeitiger Lesezugriff, aber nur exklusiver Schreibzugriff erlaubt

ERCW: *Exclusive Read, Exclusive Write*
nur exklusiver Lese- und Schreibzugriff erlaubt

EREW: *Exclusive Read, Concurrent Write*
exklusiver Lese-, aber gleichzeitiger Schreibzugriff erlaubt

- Relativ unproblematisch: Concurrent Read
- Definitiv problematisch: Concurrent Write

PRAM: Behandlung von Schreibkonflikten

... im *Concurrent Write* (CW) Modell:

Weak CW: Prozessoren können nur 0 parallel in dieselbe Zelle schreiben.

Common CW: Prozessoren können parallel *den gleichen Wert* in dieselbe Zelle schreiben.

Random / Arbitrary CW: Schreibkonflikt wird durch Auswahl eines *zufälligen* Werts aus den konkurrierenden Schreibanforderungen aufgelöst

Priority CW: Bei Schreibkonflikt erhält Prozessor mit höchster Priorität (i.d.R. mit kleinstem Index) exklusiven Schreibzugriff

Bewertung von parallelen Algorithmen

Ausführungszeiten:

- *Sequentielle Ausführungszeit* T_1: Laufzeit des (besten) sequentiellen Algorithmus zur Lösung eines Problems
- *Parallele Ausführungszeit* T_P: Laufzeit bei Verwendung von P Prozessoren

Abgeleitete Bewertungsmaße:

- *Speedup* $S(P) = \frac{T_1}{T_P}$ Es gilt $S(P) \leq P$
- *Effizienz* $E(P) = \frac{T_1}{P \cdot T_P}$ Es gilt $E(P) \leq 1$

Effizienz gibt Anteil des maximal erreichbaren Speedups an.
$E \cdot P$ Prozessoren tragen effektiv zur Problemlösung bei.

Programmierung einer PRAM: Beispiele

1. Vektoraddition, d.h. Addition von N Zahlen
2. Maximum von N Zahlen (= Vektormaximum)
3. Matrixmultiplikation

Mit welcher Zielsetzung?

Programmierung einer PRAM: Beispiele

1. Vektoraddition, d.h. Addition von N Zahlen
2. Maximum von N Zahlen (= Vektormaximum)
3. Matrixmultiplikation

Mit welcher Zielsetzung?

Klar: Reduktion der Laufzeit!

Programmierung einer PRAM: Beispiele

1. Vektoraddition, d.h. Addition von N Zahlen
   ... in $\log(N)$ Zeit

2. Maximum von N Zahlen (= Vektormaximum)

3. Matrixmultiplikation

Mit welcher Zielsetzung?

Klar: Reduktion der Laufzeit!

Programmierung einer PRAM: Beispiele

1. Vektoraddition, d.h. Addition von N Zahlen
   ... in $\log(N)$ Zeit

2. Maximum von N Zahlen (= Vektormaximum)
   ... in *konstanter* Zeit

3. Matrixmultiplikation
   ...

Mit welcher Zielsetzung? Klar: Reduktion der Laufzeit!

PRAM Programmierung: Vektoraddition

Geg.: N Zahlen x_i, $i = 1 \ldots N$

Gesucht: $S = \sum_{i=1}^{N} x_i$

Frage: Wie von PRAM-Modell profitieren?

PRAM Programmierung: Vektoraddition

Geg.: N Zahlen x_i, $i = 1 \ldots N$

Gesucht: $S = \sum_{i=1}^{N} x_i$

Frage: Wie von PRAM-Modell profitieren?

Klar: *Beliebige Anzahl von Prozessoren kann benutzt werden!*

PRAM Programmierung: Vektoraddition

Geg.: N Zahlen x_i, $i = 1 \ldots N$

Gesucht: $S = \sum_{i=1}^{N} x_i$

Frage: Wie von PRAM-Modell profitieren?

Klar: *Beliebige Anzahl von Prozessoren kann benutzt werden!*

Aber: Wie Berechnung eines skalaren Ergebnisses auf mehrere Rechenknoten verteilen?

PRAM Programmierung: Vektoraddition

Geg.: N Zahlen x_i, $i = 1 \ldots N$

Gesucht: $S = \sum_{i=1}^{N} x_i$

Frage: Wie von PRAM-Modell profitieren?

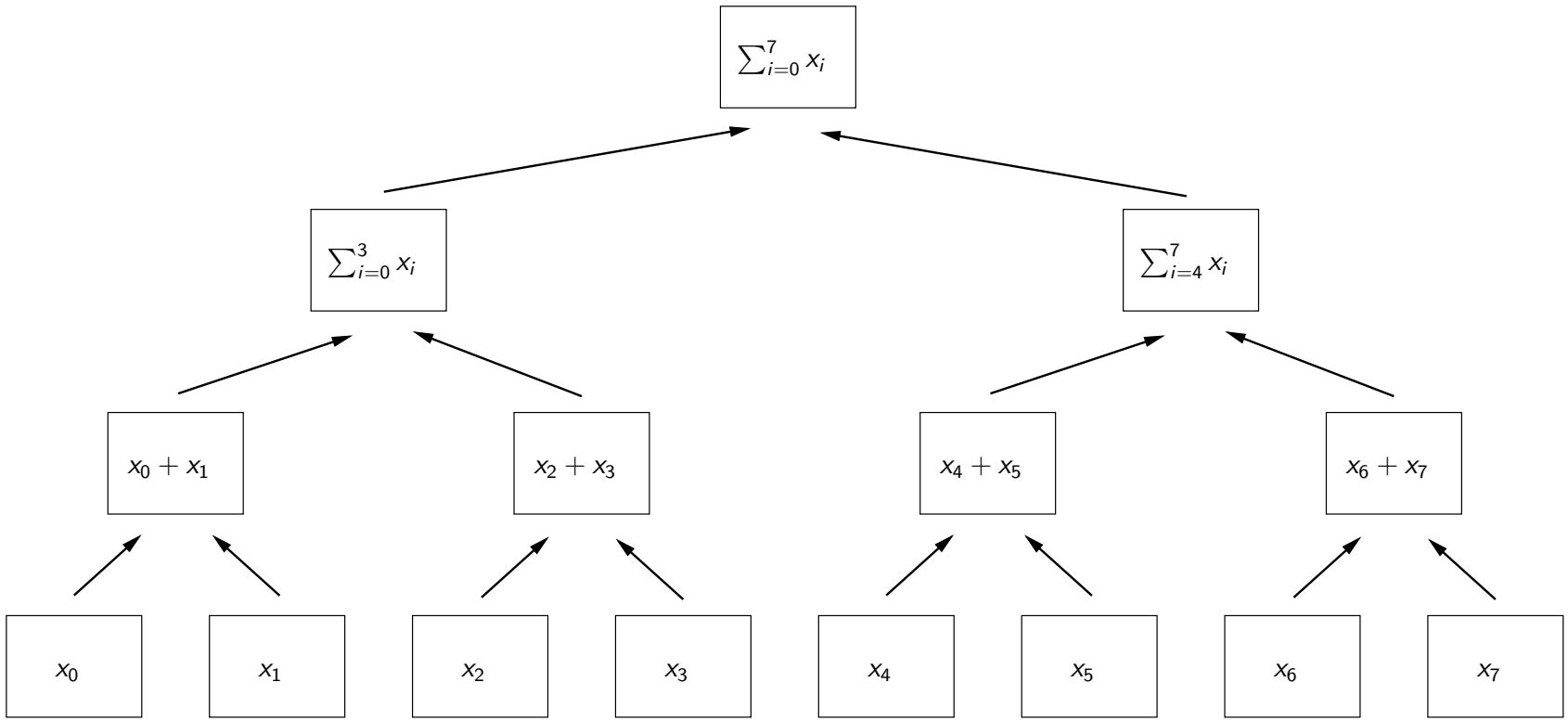
Klar: *Beliebige Anzahl von Prozessoren kann benutzt werden!*

Aber: Wie Berechnung eines skalaren Ergebnisses auf mehrere Rechenknoten verteilen?

Idee: Geeignete Definition partieller Ergebnisse (parallel berechenbar) und Kombination dieser zu Gesamtergebnis

PRAM Programmierung: Vektoraddition II

Hier: Addition in Baumstruktur ausführen



Annahme: $N = 2^k$, dann $\log_2 N = k$ Zeitschritte erforderlich!

PRAM Programmierung: Vektoraddition II

```
#define N ...

shared int x[N] = {...};
int i, j, stride;
int sum;

for (i = 0, stride = 1; i < ld(N); i++, stride *= 2)
        parallel for (j = 0; j < N; j += 2 * stride)
                x[j] = x[j] + x[j + stride];
sum = x[0];
```

PRAM Programmierung: Vektoraddition III

Laufzeitanalyse

- Sequentielle Laufzeit der Summation:
  - $N-1$ Additionen

$\Rightarrow$ $T_1 = O(N)$

- Parallele Laufzeit auf max. $P = N/2$ Prozessoren:
  - $k = \log_2 N$ Schritte im Baum $\Rightarrow O(\log N)$

$\Rightarrow$ $T_P = O(\log N)$

- Speedup $S(N/2) = \dfrac{T_1}{T_P} = \dfrac{O(N)}{O(\log N)} = O(\dfrac{N}{\log N})$
- Effizienz $E(N/2) = \dfrac{T_1}{N/2 \cdot T_P} = \dfrac{O(N)}{O(N) \cdot O(\log N)} = O(\dfrac{1}{\log N})$

PRAM Programmierung: Vektormaximum

Geg.: N Zahlen x_i, $i = 1 \ldots N$

Gesucht: $M = \max i = 1^N x_i$

Frage: Wie von PRAM-Modell profitieren?

Klar: Beliebige Anzahl von Prozessoren kann benutzt werden!

Problem: Was bringt uns das hier?

Braucht man vielleicht noch weitere PRAM-Eigenschaften?

PRAM Programmierung: Vektormaximum II

```
#define N ...
shared int x[N] = {...}, int ismax[N];
int i, j, ij, max;

parallel for (i = 0; i < N; i++)
        ismax[i] = 1;
parallel for (ij = 0; ij < N * N; ij++) {
        i = ij / N; j = ij % N;
        if (x[i] < x[j])
                ismax[i] = 0;
        }
parallel for (i = 0; i < N; i++)
        if (ismax[i] == 1)
                max = x[i];
```

PRAM Programmierung: Vektormaximum III

Welche PRAM-Eigenschaften werden ausgenützt?

PRAM Programmierung: Vektormaximum III

Welche PRAM-Eigenschaften werden ausgenützt?

- Beliebige Anzahl von Prozessoren (hier N^2)

PRAM Programmierung: Vektormaximum III

Welche PRAM-Eigenschaften werden ausgenützt?

- Beliebige Anzahl von Prozessoren (hier N^2)
- Beliebig grosser Speicher (hier Zwischenergebnisfeld 'ismax[]')

PRAM Programmierung: Vektormaximum III

Welche PRAM-Eigenschaften werden ausgenützt?

- Beliebige Anzahl von Prozessoren (hier N^2)
- Beliebig grosser Speicher (hier Zwischenergebnisfeld 'ismax[]')
- Concurrent Read (in Schritt zwei greifen jeweils N Prozessoren auf jedes 'x[i]' zu)

PRAM Programmierung: Vektormaximum III

Welche PRAM-Eigenschaften werden ausgenützt?

- Beliebige Anzahl von Prozessoren (hier N^2)
- Beliebig grosser Speicher (hier Zwischenergebnisfeld 'ismax[]')
- Concurrent Read (in Schritt zwei greifen jeweils N Prozessoren auf jedes 'x[i]' zu)
- Concurrent Write (in Schritt zwei schreiben i.d.R. mehrere Prozessoren 0 in 'ismax[j]')

Wieviele genau?

Welche CW Betriebsart ist erforderlich?

PRAM Programmierung: Vektormaximum IV

Laufzeitanalyse

- Sequentielle Laufzeit:
  - N Vergleiche erforderlich (je Element)

$\Rightarrow$ $T_1 = O(N)$

- Parallele Laufzeit auf max. $P = N^2$ Prozessoren:
  - 1 Schritt f. Initialisierungvon `ismax[]` (N Proz.)
  - 1 Schritt f. par. Bestimmung von `ismax[]` (N^2 Proz.)
  - 1 Schritt f. par. Prüfung auf Maximum (N Proz.)

$\Rightarrow$ $T_P = O(1)$

- Speedup $S(N^2) = \frac{T_1}{T_P} = \frac{O(N)}{O(1)} = O(N)$
- Effizienz $E(N^2) = \frac{T_1}{N^2 \cdot T_P} = \frac{O(N)}{N^2 \cdot O(1)} = O(\frac{1}{N})$

PRAM Programmierung: Vektormaximum IV

Laufzeitanalyse

- Sequentielle Laufzeit:
  - N Vergleiche erforderlich (je Element)

$\Rightarrow$ $T_1 = O(N)$

- Parallele Laufzeit auf max. $P = N^2$ Prozessoren:
  - 1 Schritt f. Initialisierungvon `ismax[]` (N Proz.)
  - 1 Schritt f. par. Bestimmung von `ismax[]` (N^2 Proz.)
  - 1 Schritt f. par. Prüfung auf Maximum (N Proz.)

$\Rightarrow$ $T_P = O(1)$

- Speedup $S(N^2) = \frac{T_1}{T_P} = \frac{O(N)}{O(1)} = O(N)$
- Effizienz $E(N^2) = \frac{T_1}{N^2 \cdot T_P} = \frac{O(N)}{N^2 \cdot O(1)} = O(\frac{1}{N})$

... wegen “verschwenderischem” Umgang mit Prozessoren!

PRAM Programmierung: Matrixmultiplikation

Geg.: zwei Matrizen $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

Gesucht: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
mit $\mathbf{C} = [c_{ij}]$ und $c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}$ für $0 \leq i, j < n$

Komplexität bei linearer Berechung?

PRAM Programmierung: Matrixmultiplikation

Geg.: zwei Matrizen $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

Gesucht: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
mit $\mathbf{C} = [c_{ij}]$ und $c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}$ für $0 \leq i, j < n$

Komplexität bei linearer Berechung? $O(n^3)$

PRAM Programmierung: Matrixmultiplikation

Geg.: zwei Matrizen $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

Gesucht: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
mit $\mathbf{C} = [c_{ij}]$ und $c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}$ für $0 \leq i, j < n$

Komplexität bei linearer Berechung? $O(n^3)$

Frage: Welche Parallelisierungsmöglichkeiten existieren?

PRAM Programmierung: Matrixmultiplikation

Geg.: zwei Matrizen $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

Gesucht: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
mit $\mathbf{C} = [c_{ij}]$ und $c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}$ für $0 \leq i, j < n$

Komplexität bei linearer Berechung? $O(n^3)$

Frage: Welche Parallelisierungsmöglichkeiten existieren?

Naheliegend: Parallele Berechnung der c_{ij} auf n^2 Prozessoren

PRAM Programmierung: Matrixmultiplikation II

```
#define N ...

shared int A[N][N], B[N][N], C[N][N];
int i, j, k, ij;

parallel for (ij = 0; ij < N * N; ij++) {
        i = ij / N; j = ij % N;
        C[i][j] = 0;
        for (k = 0; k < N; k++)
                C[i][j] += A[i][k] * B[k][j];
        }
```

PRAM Programmierung: Matrixmultiplikation III

Laufzeitanalyse (Variante A)

- Sequentielle Laufzeit:
  - N^2 Elemente in Ergebnismatrix
  - je N Multiplikationen und $N-1$ Additionen

$\Rightarrow$ $T_1 = N^2 * (N + (N-1)) = O(N^3)$

- Parallele Laufzeit auf $P = N^2$ Prozessoren:
  - je 1 Skalarprodukt (N Multiplikationen und $N-1$ Additionen)

$\Rightarrow$ $T_P = O(N)$

- Speedup $S(N^2) = \frac{T_1}{T_P} = \frac{O(N^3)}{O(N)} = O(N^2)$
- Effizienz $E(N^2) = \frac{T_1}{N^2 \cdot T_P} = \frac{O(N^3)}{N^2 \cdot O(N)} = O(1)$

PRAM Programmierung: Matrixmultiplikation III

Laufzeitanalyse (Variante A)

- Sequentielle Laufzeit:
  - N^2 Elemente in Ergebnismatrix
  - je N Multiplikationen und $N-1$ Additionen

$\Rightarrow$ $T_1 = N^2 * (N + (N-1)) = O(N^3)$

- Parallele Laufzeit auf $P = N^2$ Prozessoren:
  - je 1 Skalarprodukt (N Multiplikationen und $N-1$ Additionen)

$\Rightarrow$ $T_P = O(N)$

- Speedup $S(N^2) = \frac{T_1}{T_P} = \frac{O(N^3)}{O(N)} = O(N^2)$
- Effizienz $E(N^2) = \frac{T_1}{N^2 \cdot T_P} = \frac{O(N^3)}{N^2 \cdot O(N)} = O(1)$

... also "optimal" parallelisiert (bei geg. Anz. Prozessoren)

PRAM Programmierung: Matrixmultiplikation IV

Geg.: zwei Matrizen $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

Gesucht: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
mit $\mathbf{C} = [c_{ij}]$ und $c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}$ für $0 \leq i, j < n$

Variante A: Parallele Berechnung der c_{ij} auf n^2 Prozessoren
Speedup $O(N^2)$!

Frage: Weitere Reduktion der Laufzeit möglich?

... also noch mehr Parallelität ausnutzen?

PRAM Programmierung: Matrixmultiplikation IV

Geg.: zwei Matrizen $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

Gesucht: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
mit $\mathbf{C} = [c_{ij}]$ und $c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}$ für $0 \leq i, j < n$

Variante A: Parallele Berechnung der c_{ij} auf n^2 Prozessoren
Speedup $O(N^2)$!

Frage: Weitere Reduktion der Laufzeit möglich?

... also noch mehr Parallelität ausnutzen?

Lösungsmöglichkeit: alle Produkte individuell, dann Summation

PRAM Programmierung: Matrixmultiplikation V

```
#define N ...
shared int A[N][N], B[N][N], AxB[N][N][N], C[N][N];
int i, j, k, ijk, ij, s, stride;

parallel for (ijk = 0; ijk < (N * N * N); ijk++) {
    i = ijk / (N * N); j = (ijk % (N * N)) / N; k = ijk % N
    AxB[i][j][k] = A[i][k] * B[k][j];
    }
parallel for (ij = 0; ij < N * N; ij++) {
    i = ij / N; j = ij % N;
    for (s = 0, stride = 1; s < ld(N); s++, stride *= 2)
        parallel for (k = 0; k < N; k += 2 * stride)
            AxB[i][j][k] = AxB[i][j][k] +
                           AxB[i][j][k + stride];
    C[i][j] = AxB[i][j][0];
    }
```

PRAM Programmierung: Matrixmultiplikation VI

Laufzeitanalyse (Variante B)

- Sequentielle Laufzeit: $T_1 = N^2 * (N + (N - 1)) = O(N^3)$
- Parallele Laufzeit auf max. $P = N^3$ Prozessoren:
  - je 1 Multiplikation pro Prozessor (`A[i,j,k]`)
  - Addition der `A[i,j,_]` in $O(\log_2 N)$ Schritten
  - Extraktion des Ergebnisses C_{ij} in 1 Schritt

$\Rightarrow$ $T_P = O(1 + \log N + 1) = O(\log N)$

- Speedup $S(N^3) = \frac{T_1}{T_P} = \frac{O(N^3)}{O(\log N)} > O(N^2)$
- Effizienz $E(N^3) = \frac{T_1}{N^3 \cdot T_P} = \frac{O(N^3)}{N^3 \cdot O(\log N)} = O(\frac{1}{\log N})$

PRAM Programmierung: Matrixmultiplikation VI

Laufzeitanalyse (Variante B)

- Sequentielle Laufzeit: $T_1 = N^2 * (N + (N - 1)) = O(N^3)$
- Parallele Laufzeit auf max. $P = N^3$ Prozessoren:
  - je 1 Multiplikation pro Prozessor (`A[i,j,k]`)
  - Addition der `A[i,j,_]` in $O(\log_2 N)$ Schritten
  - Extraktion des Ergebnisses C_{ij} in 1 Schritt

$\Rightarrow$ $T_P = O(1 + \log N + 1) = O(\log N)$

- Speedup $S(N^3) = \frac{T_1}{T_P} = \frac{O(N^3)}{O(\log N)} > O(N^2)$
- Effizienz $E(N^3) = \frac{T_1}{N^3 \cdot T_P} = \frac{O(N^3)}{N^3 \cdot O(\log N)} = O(\frac{1}{\log N})$

... also “prozessorintensiv” parallelisiert

PRAM: Diskussion

- PRAM definiert *theoretisches* Modell paralleler Berechnungen
- Abstrahiert von Details der Kommunikation zwischen Prozessoren sowie der Synchronisation von Berechnungen
- Erlaubt Aussagen über Effizienz paralleler Impoementierungen
- Macht aber *kritische* vereinfachende Annahmen!

PRAM: Diskussion

- PRAM definiert *theoretisches* Modell paralleler Berechnungen
- Abstrahiert von Details der Kommunikation zwischen Prozessoren sowie der Synchronisation von Berechnungen
- Erlaubt Aussagen über Effizienz paralleler Impoementierungen
- Macht aber *kritische* vereinfachende Annahmen!

Welche sind aus Ihrer Sicht am kritischsten?

PRAM: Diskussion

- PRAM definiert *theoretisches* Modell paralleler Berechnungen
- Abstrahiert von Details der Kommunikation zwischen Prozessoren sowie der Synchronisation von Berechnungen
- Erlaubt Aussagen über Effizienz paralleler Impoementierungen
- Macht aber *kritische* vereinfachende Annahmen!

Welche sind aus Ihrer Sicht am kritischsten?

- Welche aktuelle Rechnerarchitektur kommt (mit Einschränkungen) dem PRAM-Modell am nächsten?

PRAM: Diskussion

- PRAM definiert *theoretisches* Modell paralleler Berechnungen
- Abstrahiert von Details der Kommunikation zwischen Prozessoren sowie der Synchronisation von Berechnungen
- Erlaubt Aussagen über Effizienz paralleler Impoementierungen
- Macht aber *kritische* vereinfachende Annahmen!

Welche sind aus Ihrer Sicht am kritischsten?

- Welche aktuelle Rechnerarchitektur kommt (mit Einschränkungen) dem PRAM-Modell am nächsten?

GPGPU! ... mit welchen Gemeinsamkeiten/Unterschieden?