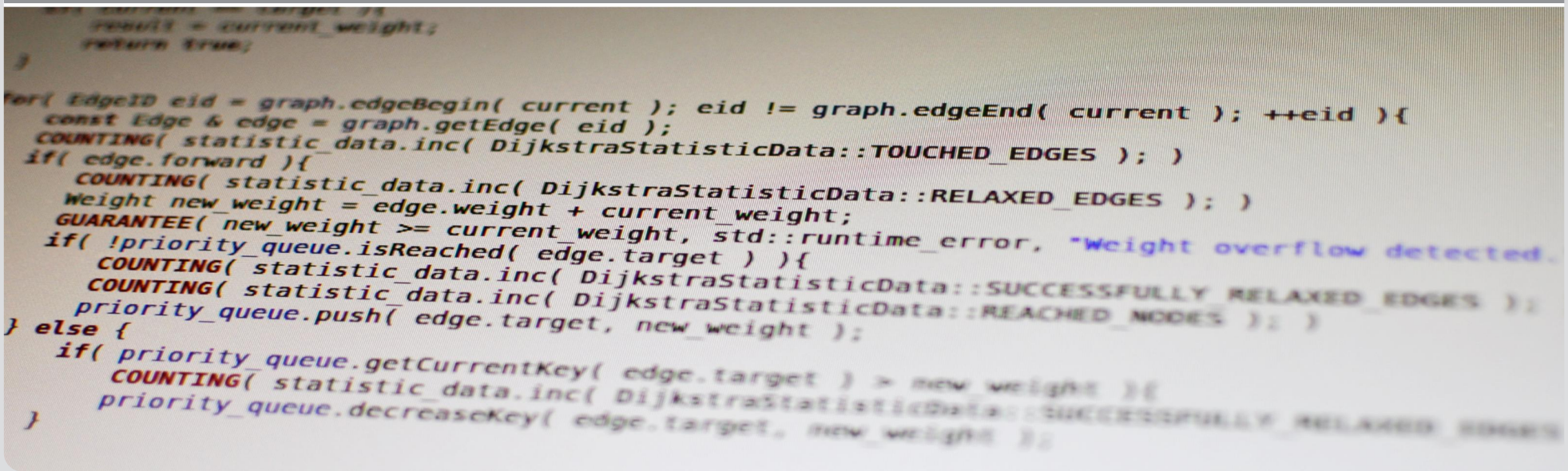
Karlsruher Institut für Technologie

Übung 7 – Algorithmen II

Yaroslav Akhremtsev, Demian Hespe – *yaroslav.akhremtsev@kit.edu, hespe@kit.edu*
Mit Folien von Michael Axtmann (teilweise)

`http://algo2.iti.kit.edu/AlgorithmenII_WS17.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

Online Algorithmen

- Expertenauswahl

Parallelverarbeitung

- Modelle
- Verbindungsstrukturen
- Anwendungen
  - Präfixsumme
  - Paralleles Sortieren

Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

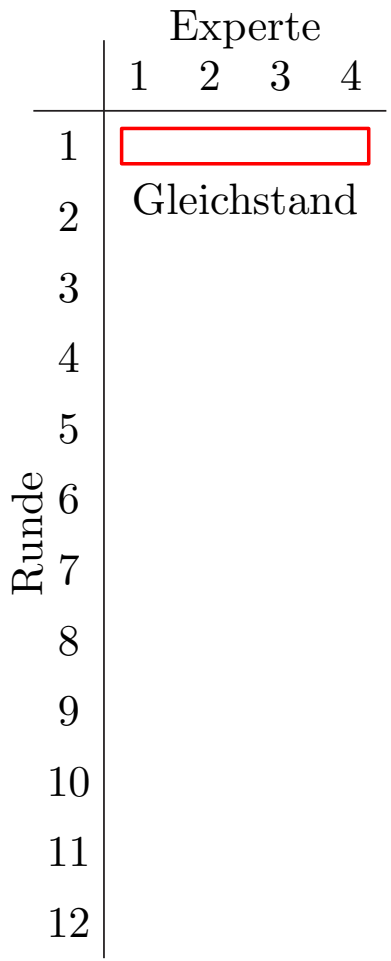
$\Rightarrow$ k mal schlechter als der beste Experte.

Runde	Experte 1	2	3	4
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

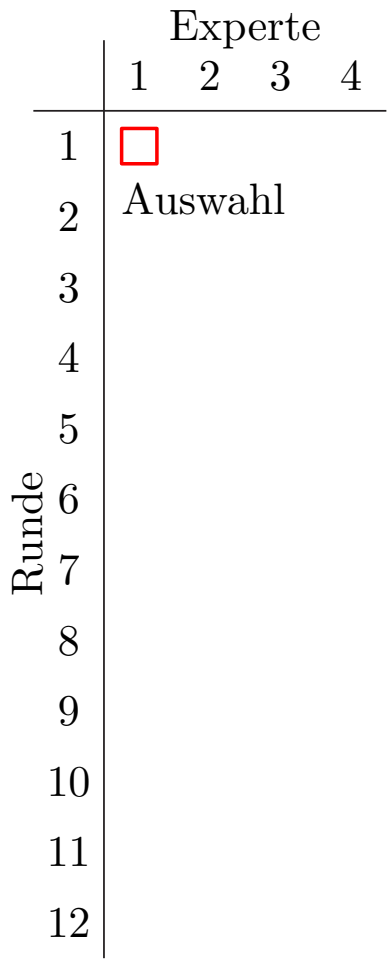
$\Rightarrow$ k mal schlechter als der beste Experte.



Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

$\Rightarrow$ k mal schlechter als der beste Experte.



Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

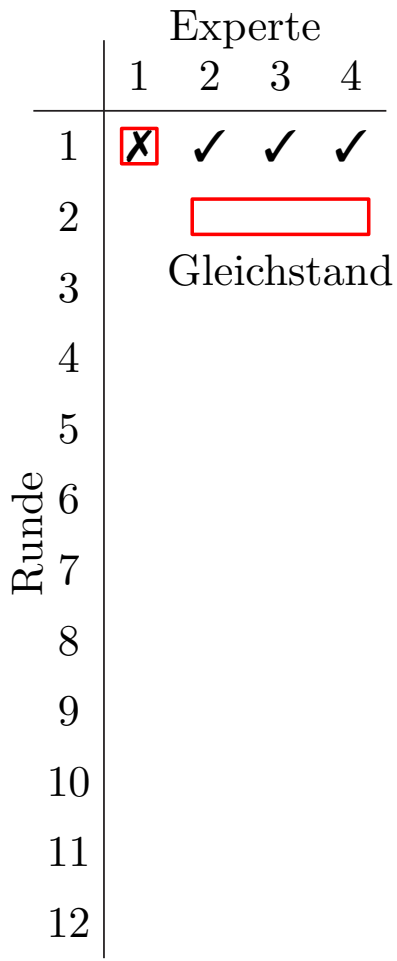
$\Rightarrow$ k mal schlechter als der beste Experte.

Runde	Experte 1	2	3	4
1	✗	✓	✓	✓
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

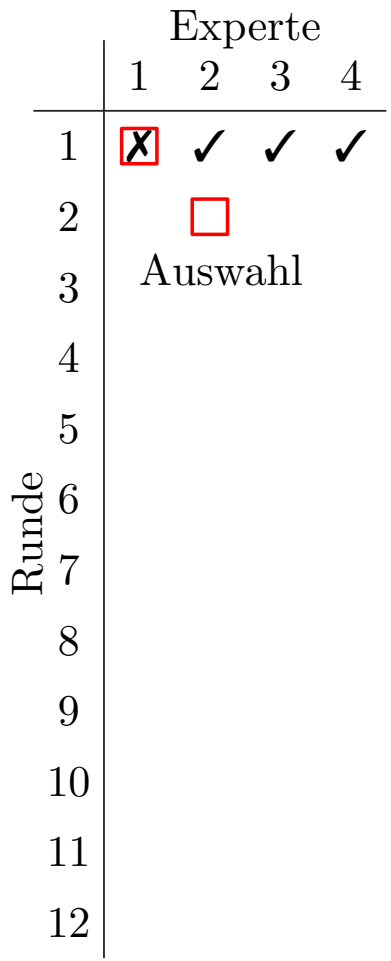
$\Rightarrow$ k mal schlechter als der beste Experte.



Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

$\Rightarrow$ k mal schlechter als der beste Experte.



Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

$\Rightarrow$ k mal schlechter als der beste Experte.

Runde	Experte 1	2	3	4
1	✗	✓	✓	✓
2	✓	✗	✓	✓
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Expertenauswahl

- k Experten, t Runden
- Strategie: Wähle immer Antwort des Experten, der in der Vergangenheit am häufigsten richtig lag.
- Deterministisch: Bei Gleichstand vorher klar, welcher Experte gewählt wird.
- Lasse den Experten falsch liegen, der gewählt wird.
  - Alle anderen liegen richtig.
- Jeder Experte liegt $\frac{t}{k}$ mal falsch.
- Algorithmus liegt immer falsch (t mal).

$\Rightarrow$ k mal schlechter als der beste Experte.

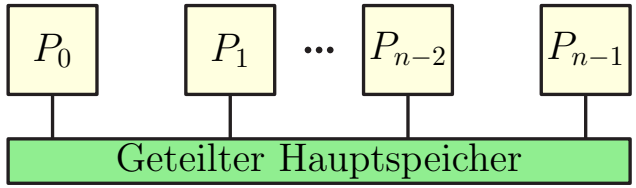
Runde	Experte 1	2	3	4
1	✗	✓	✓	✓
2	✓	✗	✓	✓
3	✓	✓	✗	✓
4	✓	✓	✓	✗
5	✗	✓	✓	✓
6	✓	✗	✓	✓
7	✓	✓	✗	✓
8	✓	✓	✓	✗
9	✗	✓	✓	✓
10	✓	✗	✓	✓
11	✓	✓	✗	✓
12	✓	✓	✓	✗

Parallelverarbeitung

Modelle

PRAM

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Verteilter gemeinsamer Speicher

(symmetrisch) gemeinsamer Speicher

BulkSynchronousParallel

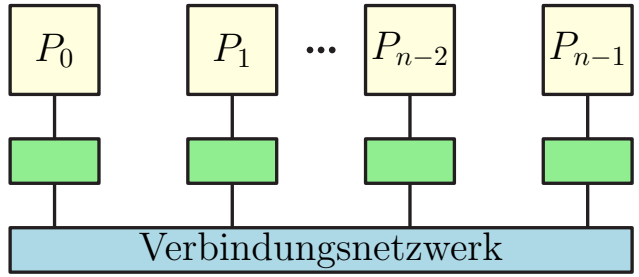
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Verteilter gemeinsamer Speicher

(symmetrisch) gemeinsamer Speicher

BulkSynchronousParallel

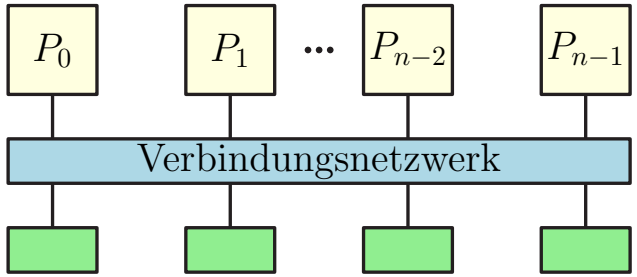
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Verteilter gemeinsamer Speicher

(symmetrisch) gemeinsamer Speicher

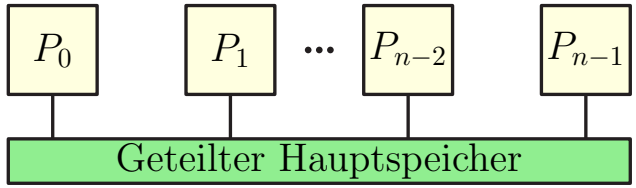
BulkSynchronousParallel

- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

PRAM

Speicherkonflikte

PRAM



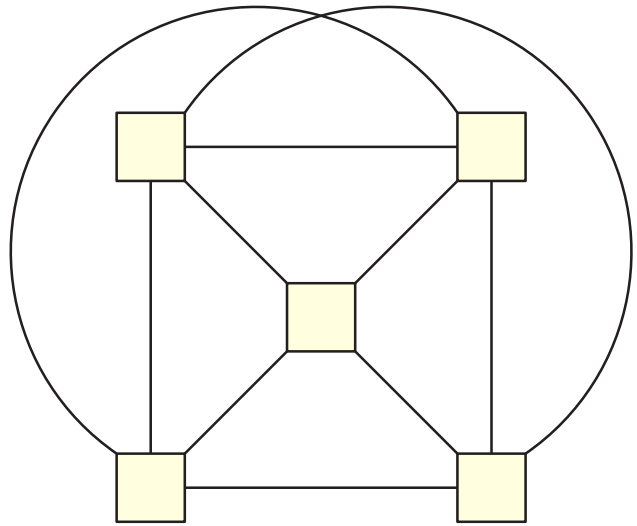
- klassifiziert nach Lese- (*read*) und Schreibzugriff (*write*)
- betrachte gleichzeitigen (*concurrent*) oder exklusiven Zugriff (*exclusive*)
  - EREW
  - ERCW schwachsinn
  - CREW
  - CRCW
    - **common**: alle müssen den gleichen Wert schreiben
    - **arbitrary**: Wert eines zufälligen Prozessors
    - **priority**: Wert mit kleinster Prozessor-ID
    - **combine**: Aggregation der Werte (z.B. Summe)

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex
  - Telefon
  - Duplex



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

Kosten

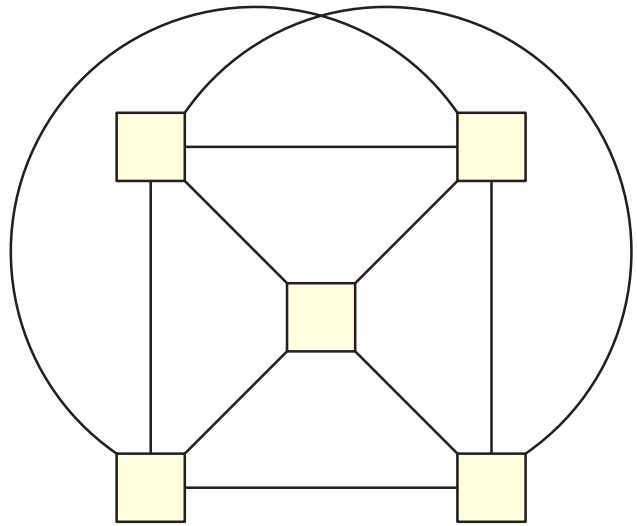
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

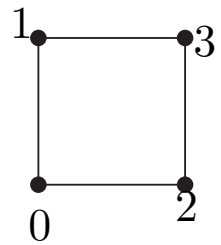
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

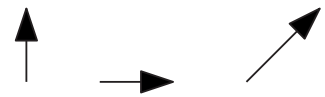
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

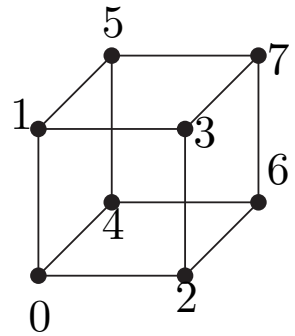
Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

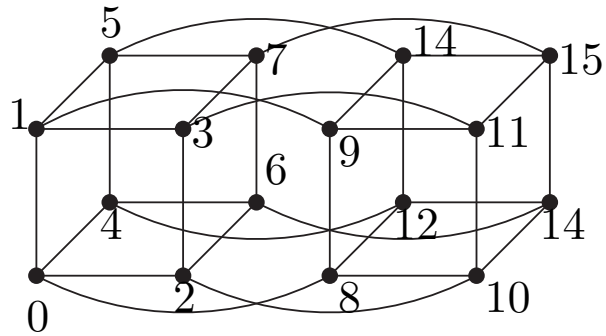
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

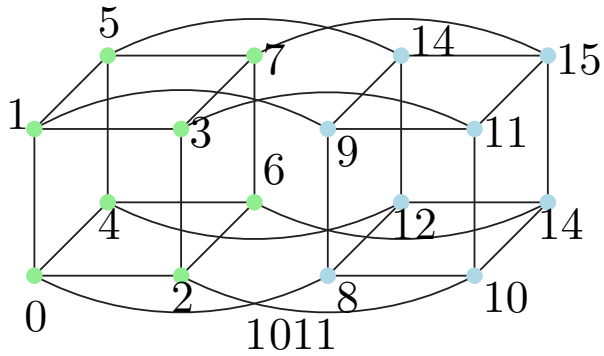
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

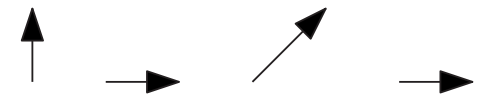
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

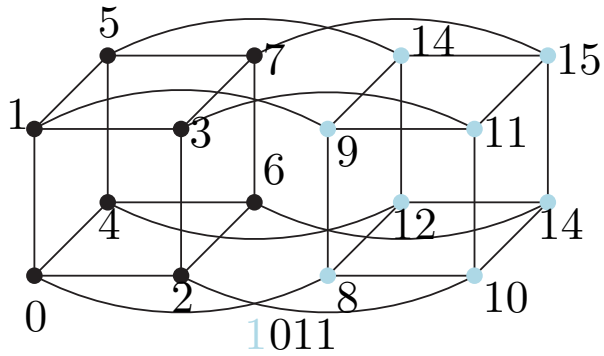
Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

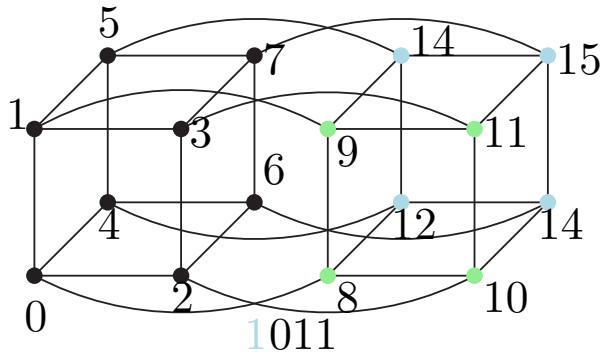
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

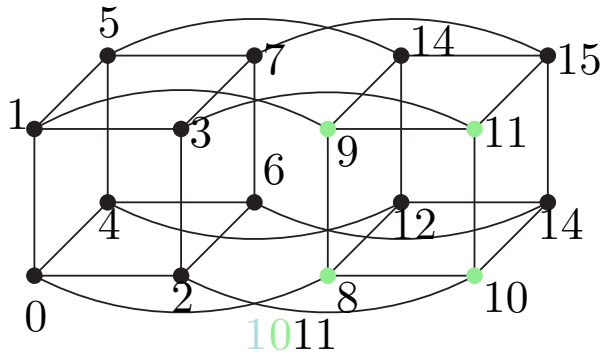
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

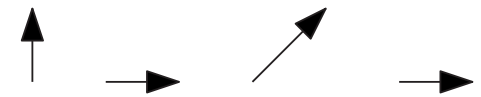
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

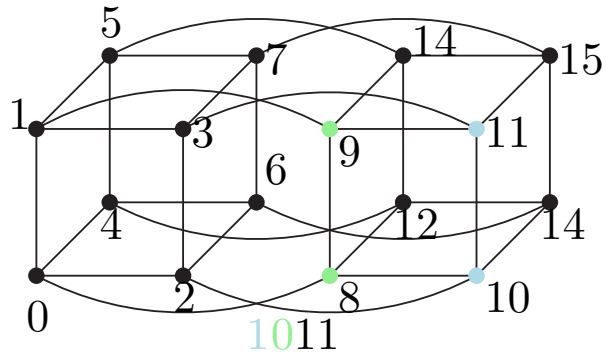
Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

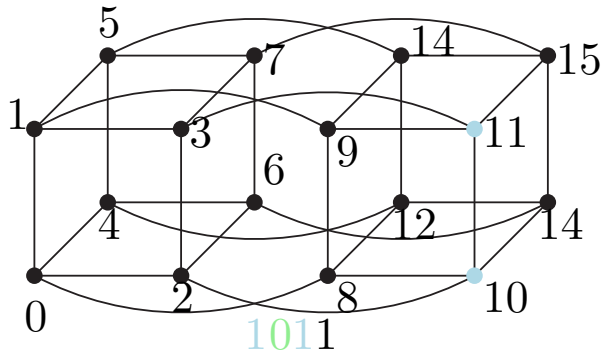
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

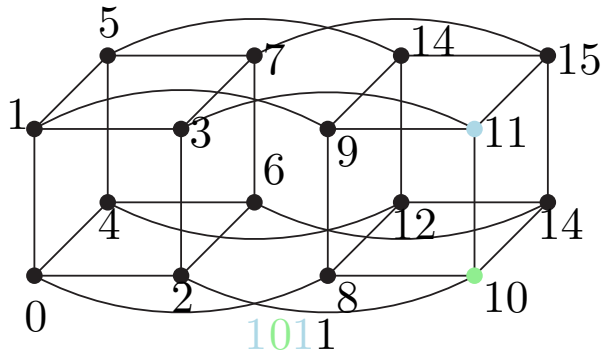
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

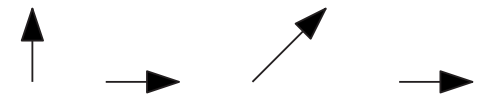
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

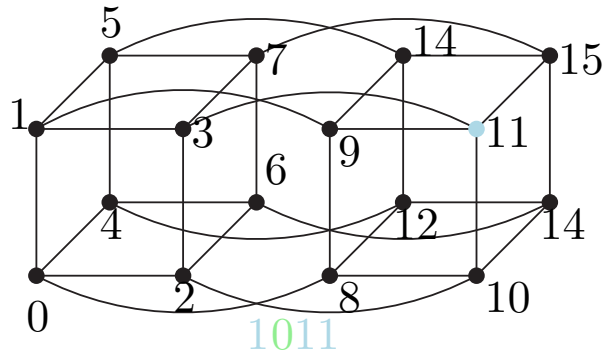
Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

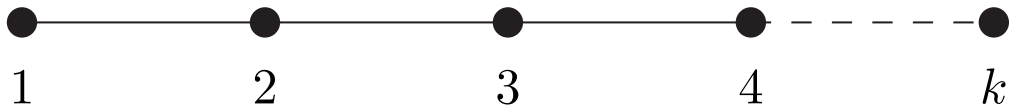
- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Nachrichtenaustausch: Pipelining



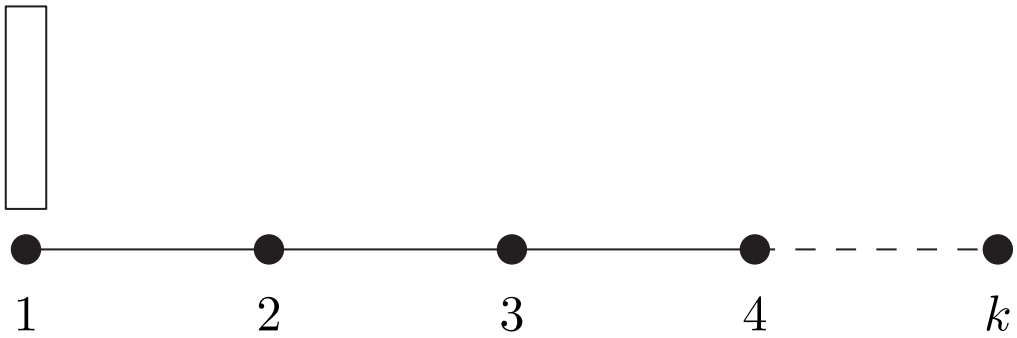
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



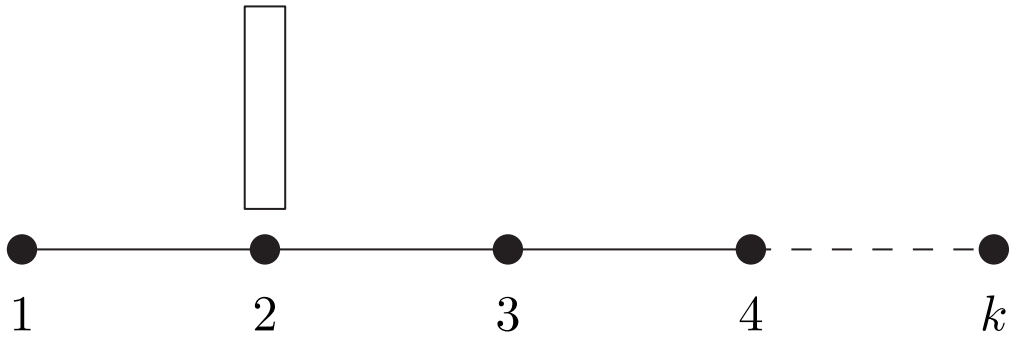
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



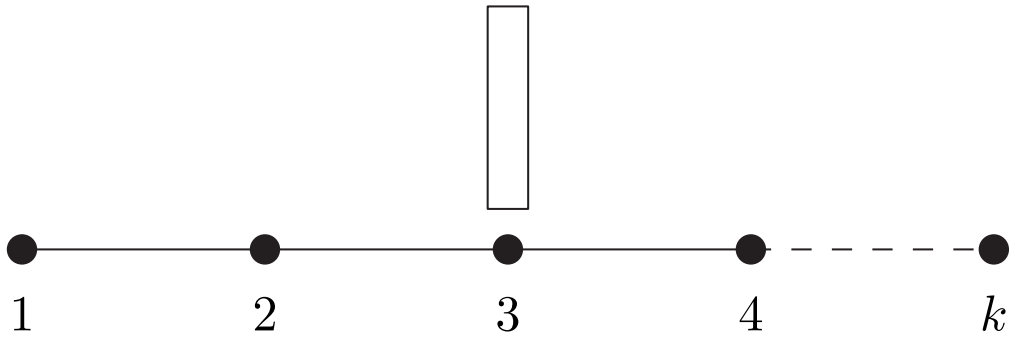
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



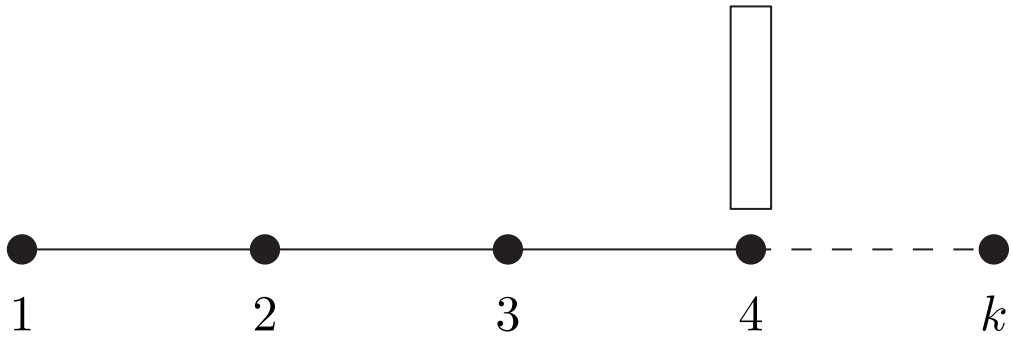
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



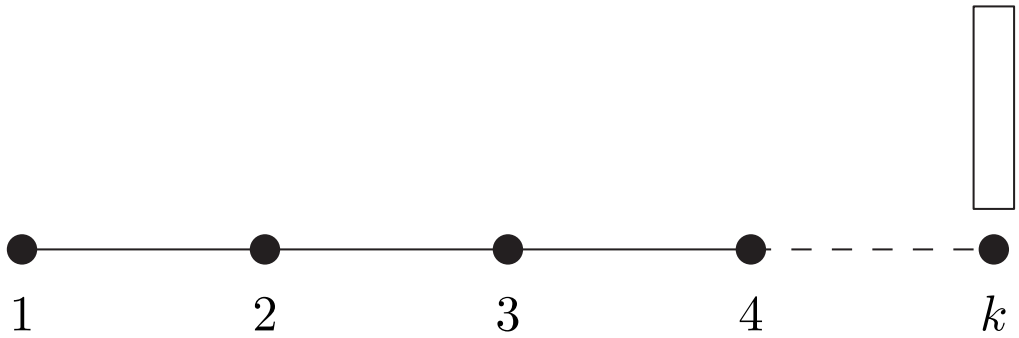
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



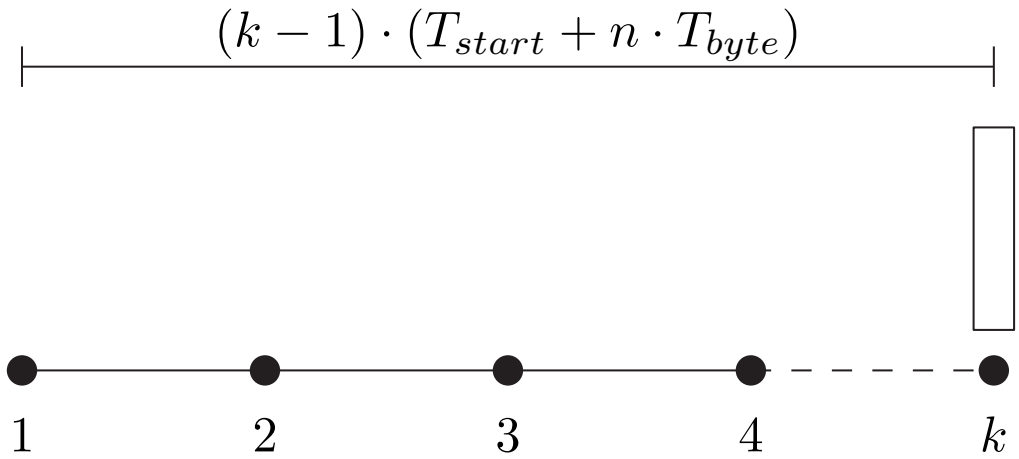
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



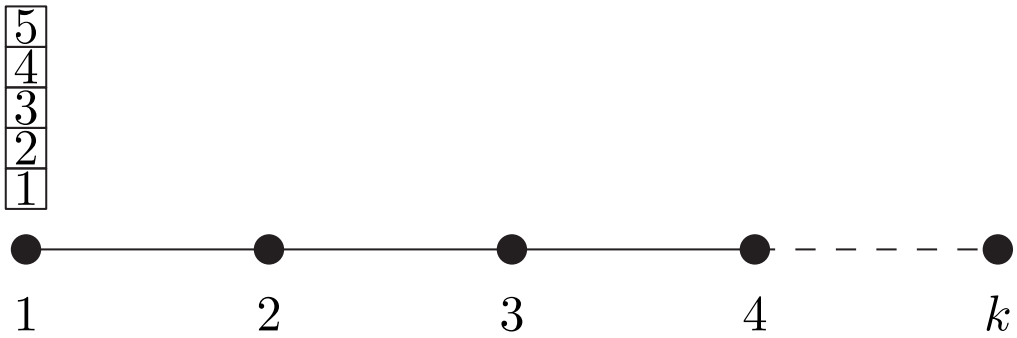
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



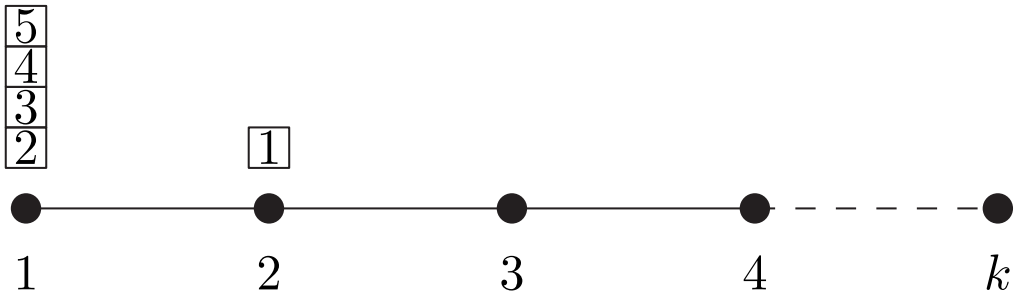
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



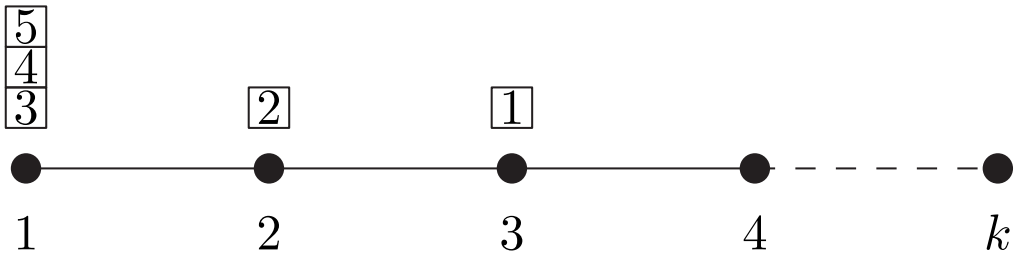
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



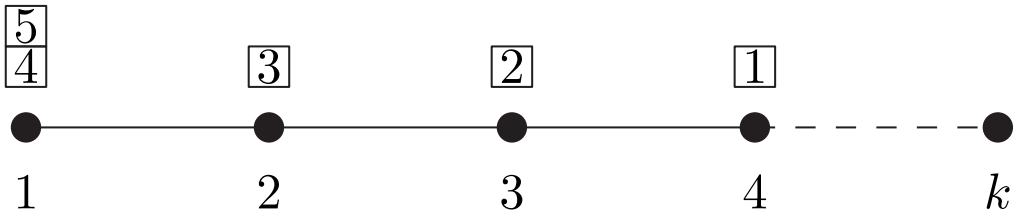
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



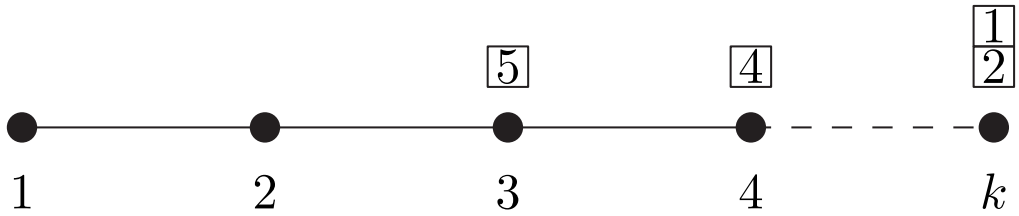
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



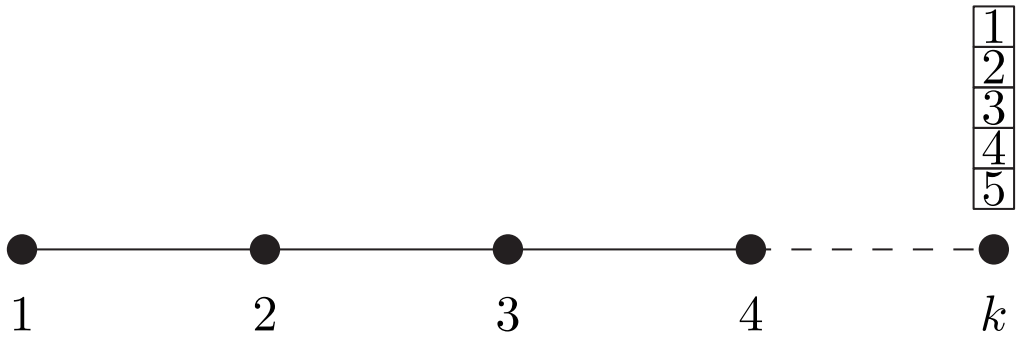
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



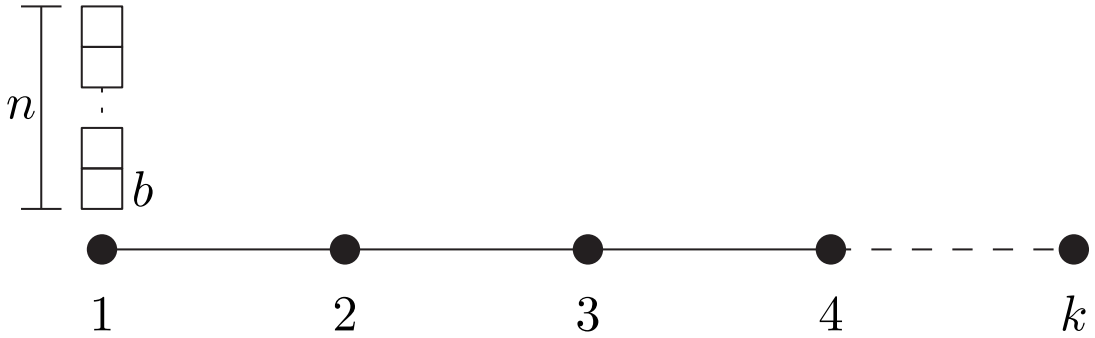
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



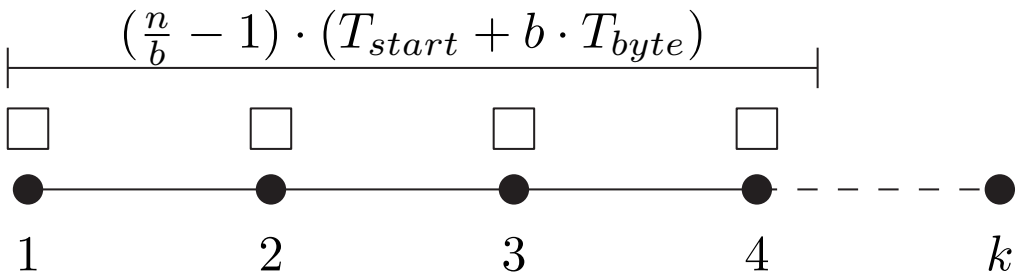
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



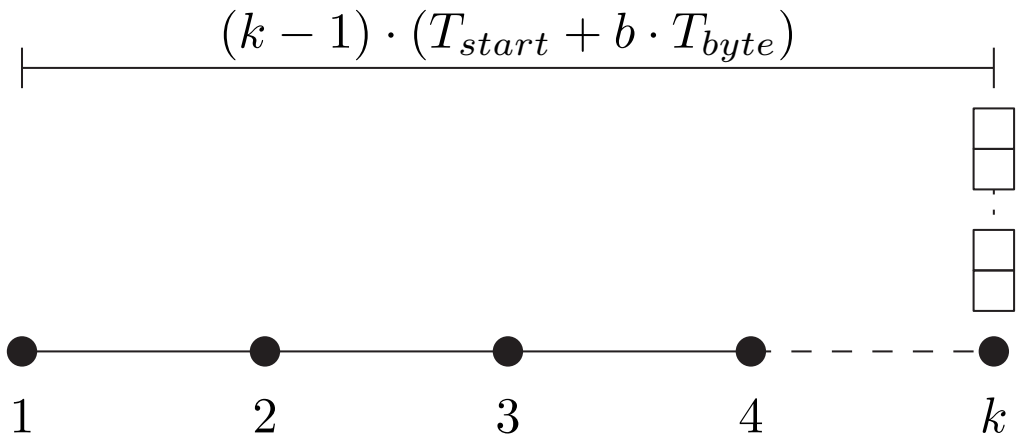
- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Nachrichtenaustausch: Pipelining



- k Prozessoren in einer “Linie” angeordnet.
  - Kommunikation nur mit direkten Nachbarn.

$$T(n) = (\frac{n}{b} + k - 2) \cdot (T_{start} + b \cdot T_{byte})$$

$$T'(n) = (k-2) \cdot T_{byte} - \frac{n}{b^2} \cdot T_{start} \stackrel{!}{=} 0$$

$$\Leftrightarrow b = \sqrt{\frac{n \cdot T_{start}}{(k-2) \cdot T_{byte}}}$$

Anwendungen

PRAM Präfixsumme

- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

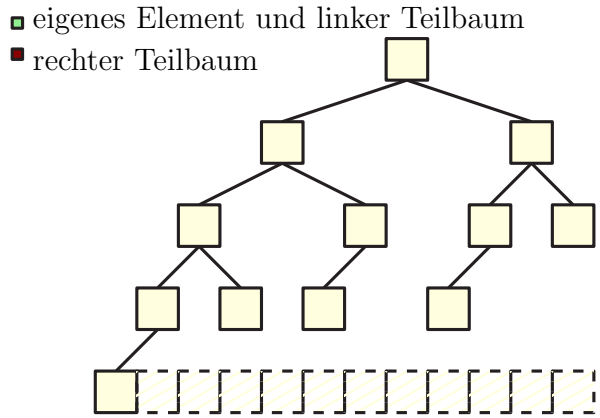
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

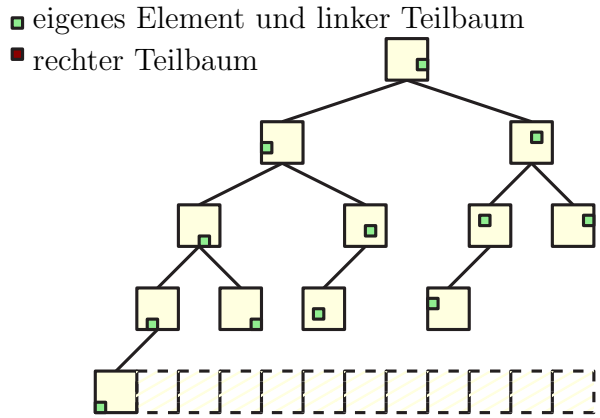
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

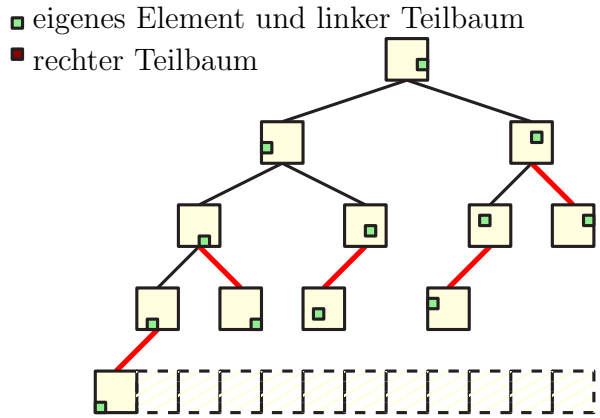
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

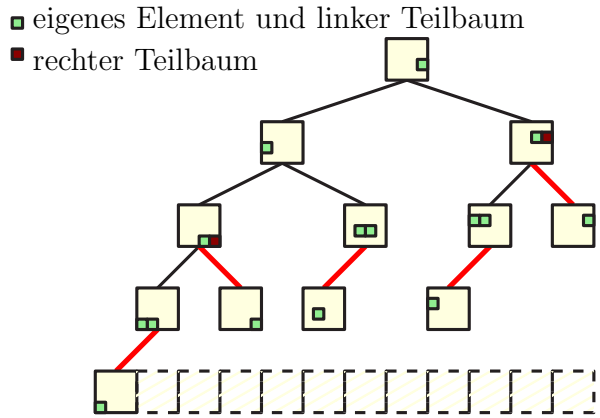
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

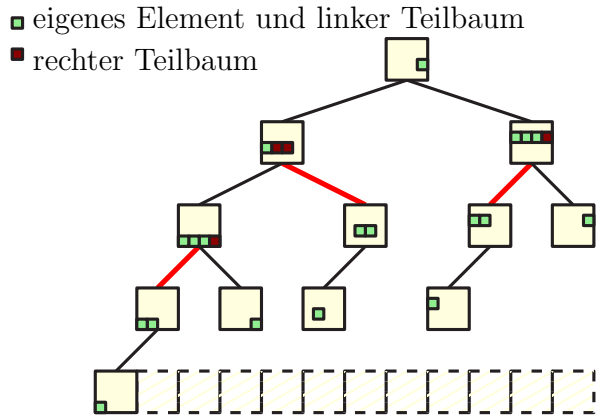
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

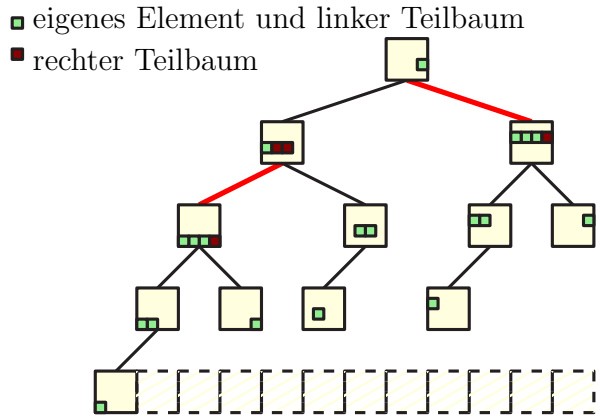
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

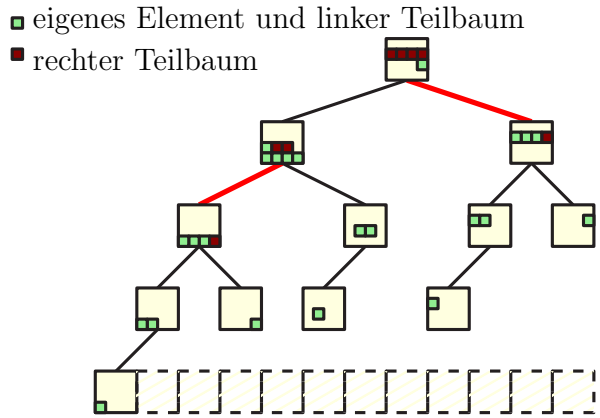
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

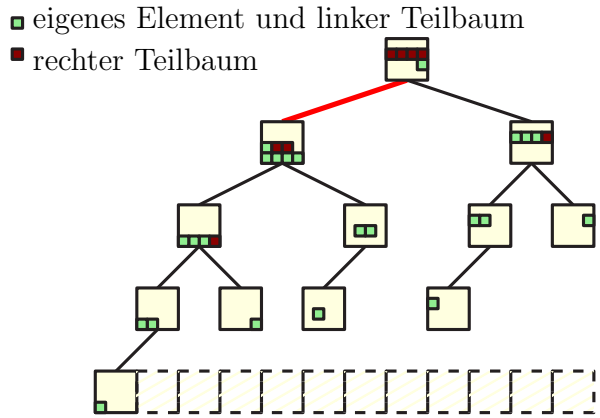
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

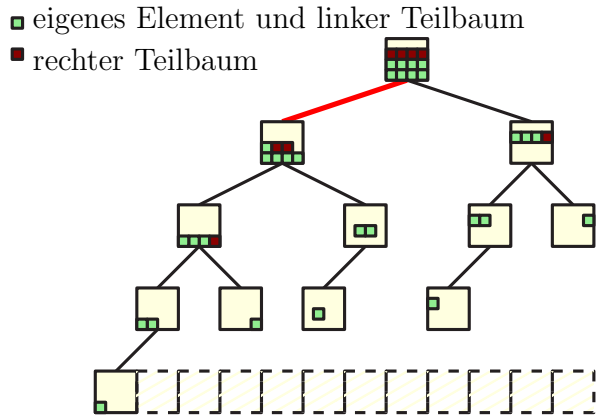
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

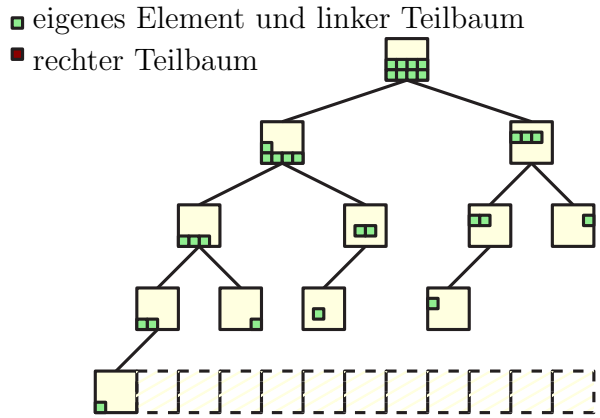
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

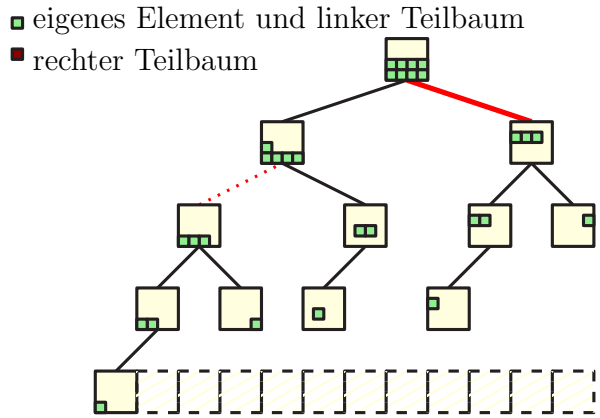
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

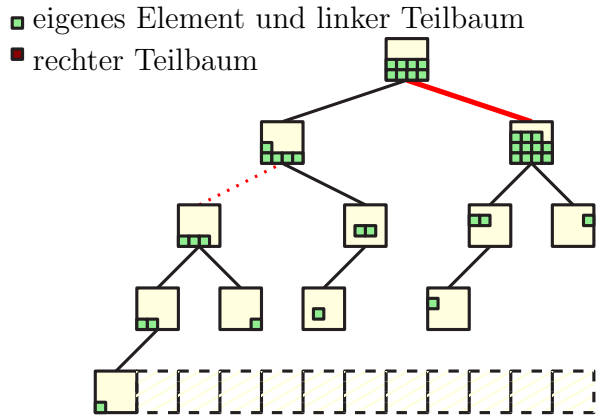
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

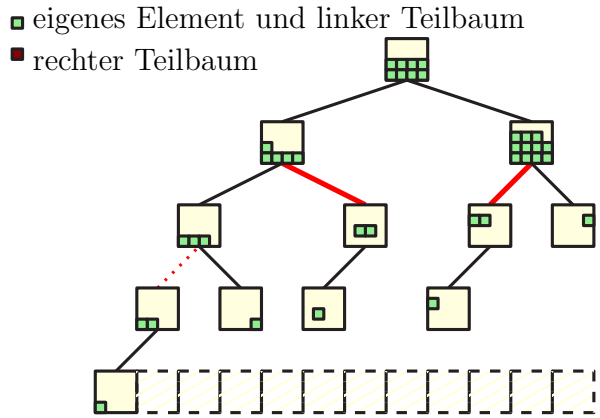
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

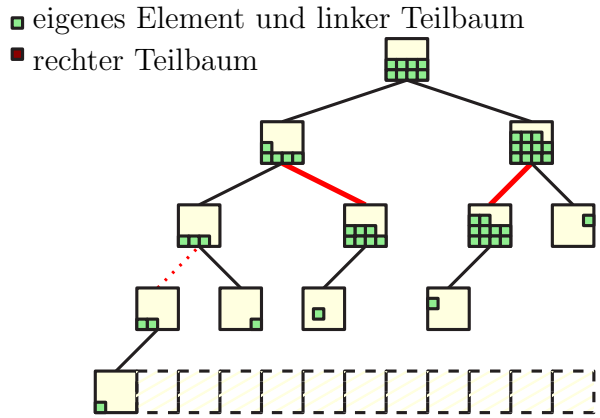
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

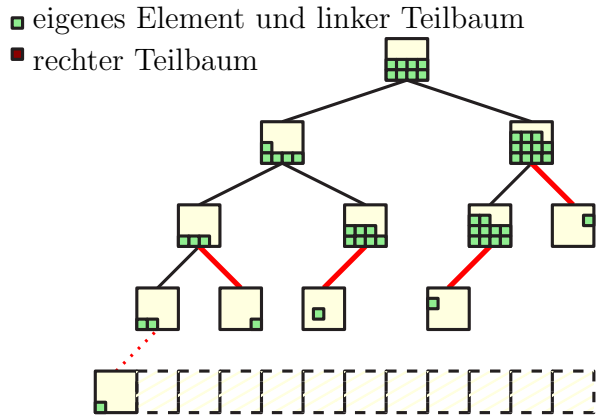
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

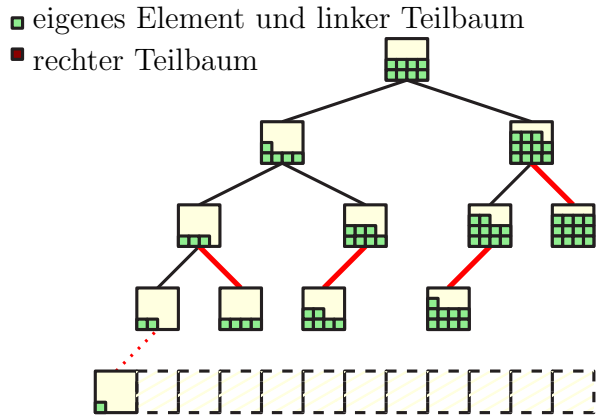
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

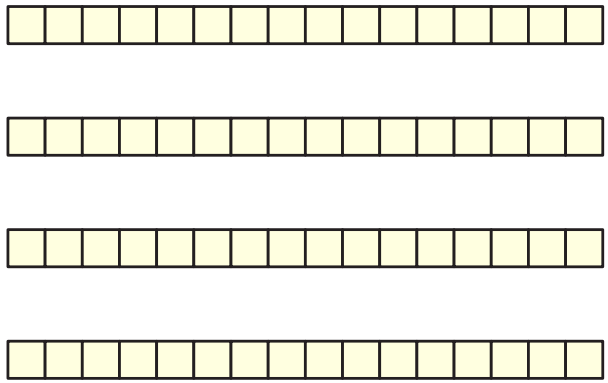
- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

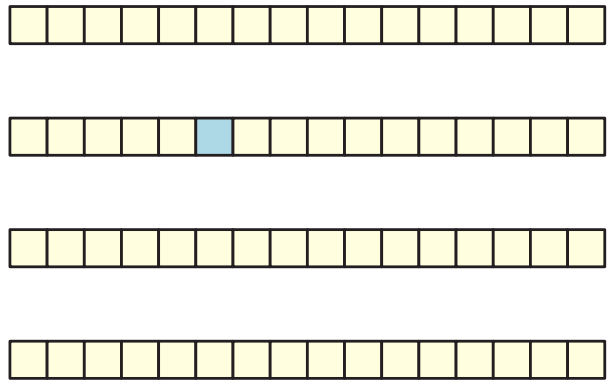


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

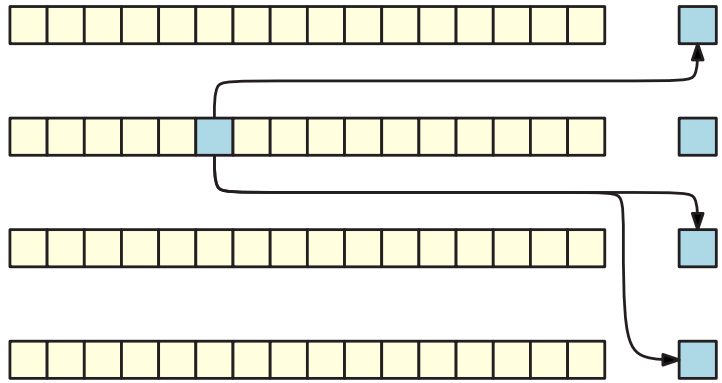


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

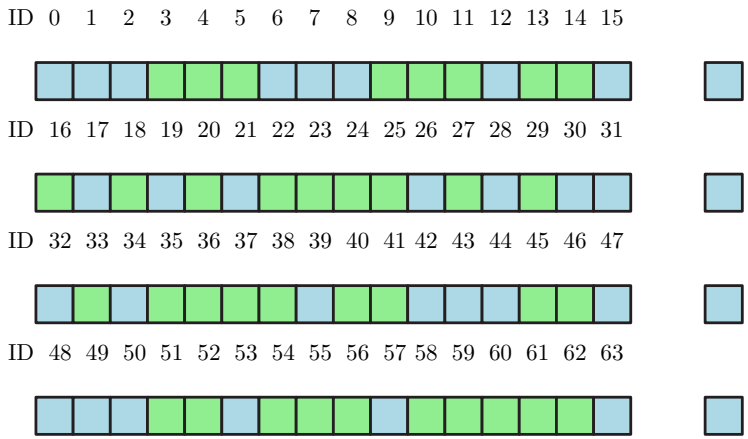


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

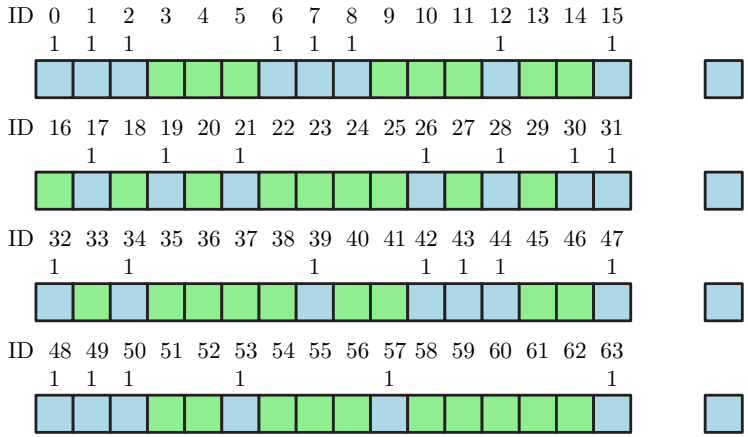


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

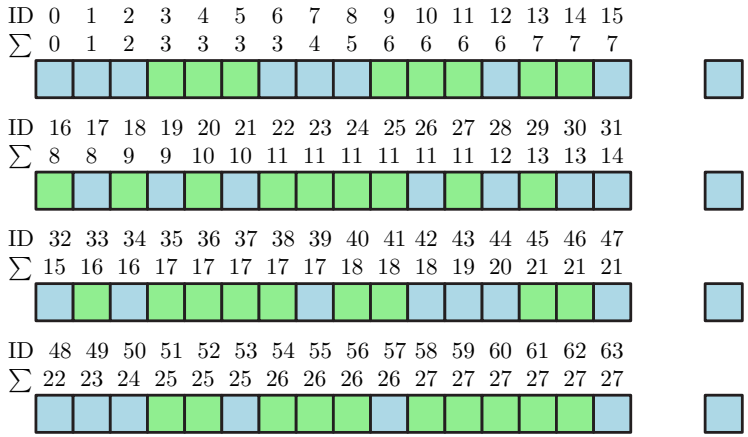


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

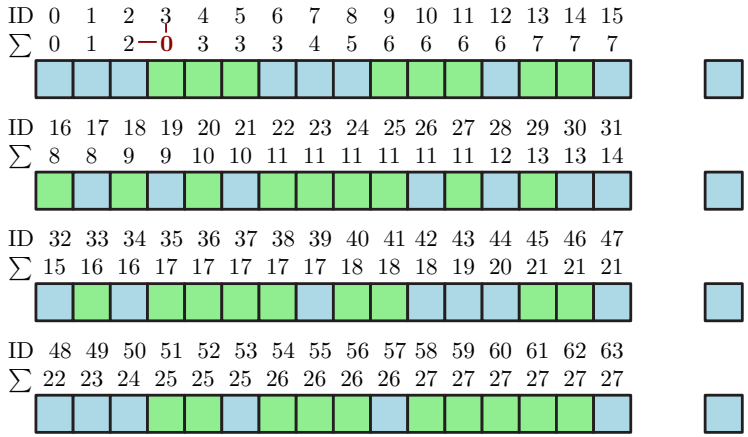


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

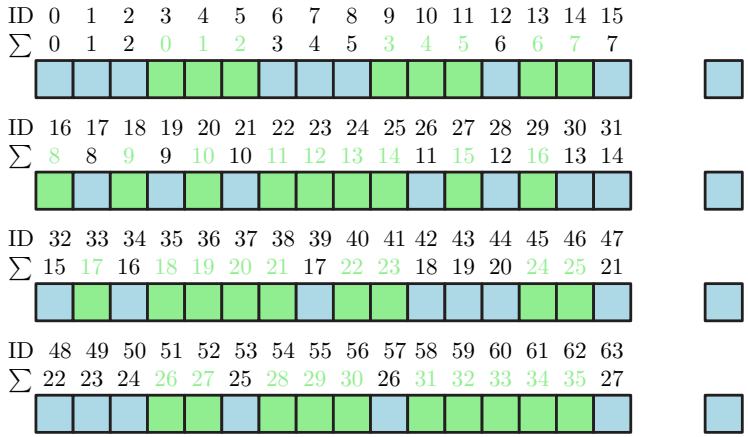


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

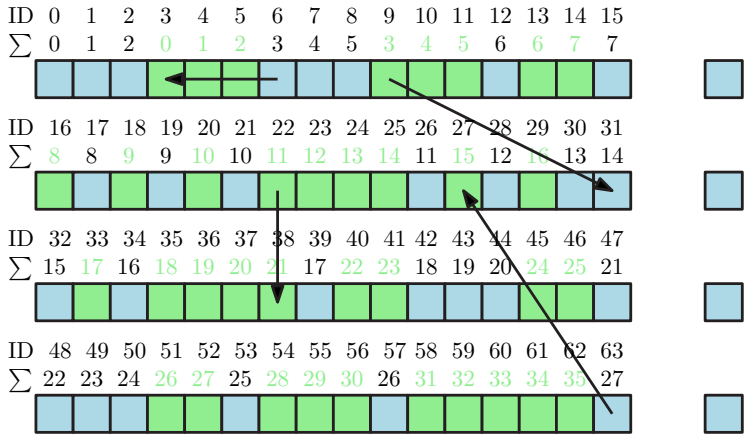


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

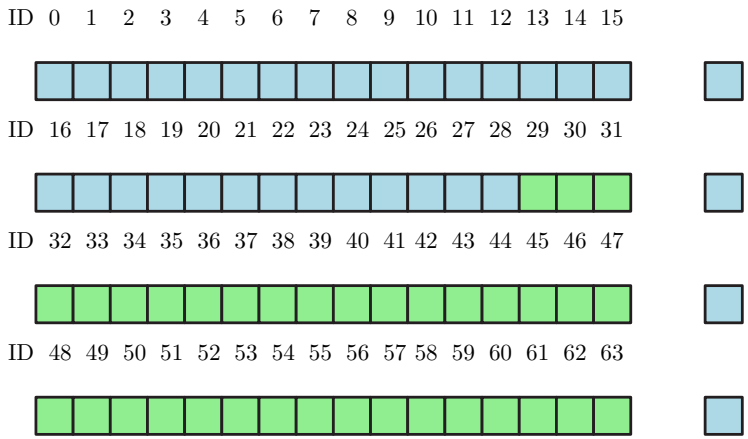


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

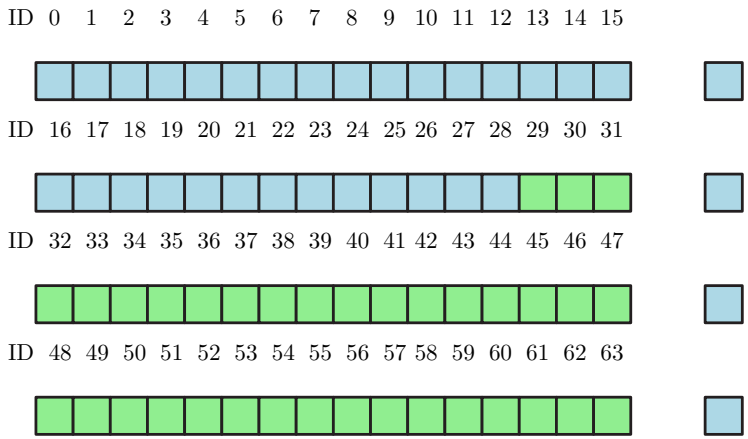


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

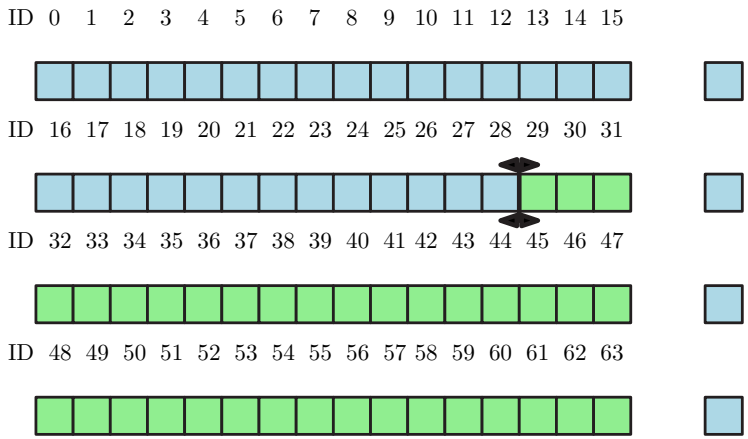


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

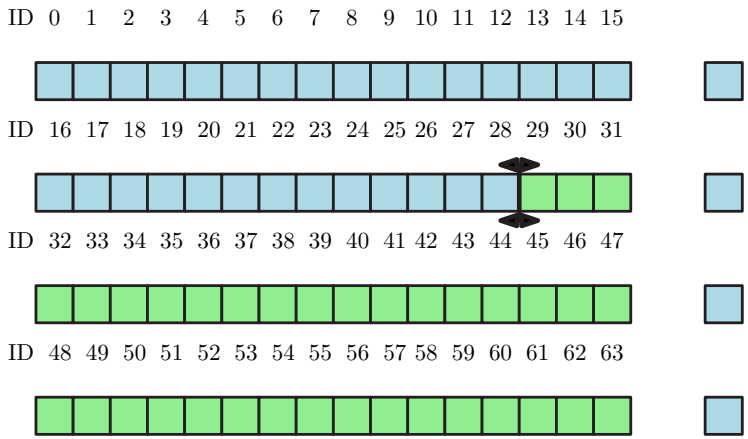
1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion



Anwendungen

Paralleler Quicksort

Rekursives Verfahren



1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion

Parallele Programmierung

Ein Einstieg

Einstieg in parallele Programmierung?

- openMP
  - `www.openmp.org`
  - enthalten im GCC Compiler
  - Parallelität über Preprozessorflags #pragma omp parallel
- Boost Threads
  - `www.boost.org`
  - Bibliothek, eigenes Management
- Grafikkarten für die, die es anspruchsvoller mögen
  - `http://developer.nvidia.com/getting-started-parallel-computing`
  - cudafähige Grafikkarte nötig
  - Nebeneffekte durch ungewohnte Grafikkartenhardware
- Message Passing Interface (MPI)
  - z.B. `https://www.open-mpi.org/`
  - Keine Threads, kein gemeinsamer Speicher, sondern getrennte Prozesse
  - Explizite Aufrufe, wenn Prozesse miteinander kommunizieren sollen

Ende!

Feierabend!