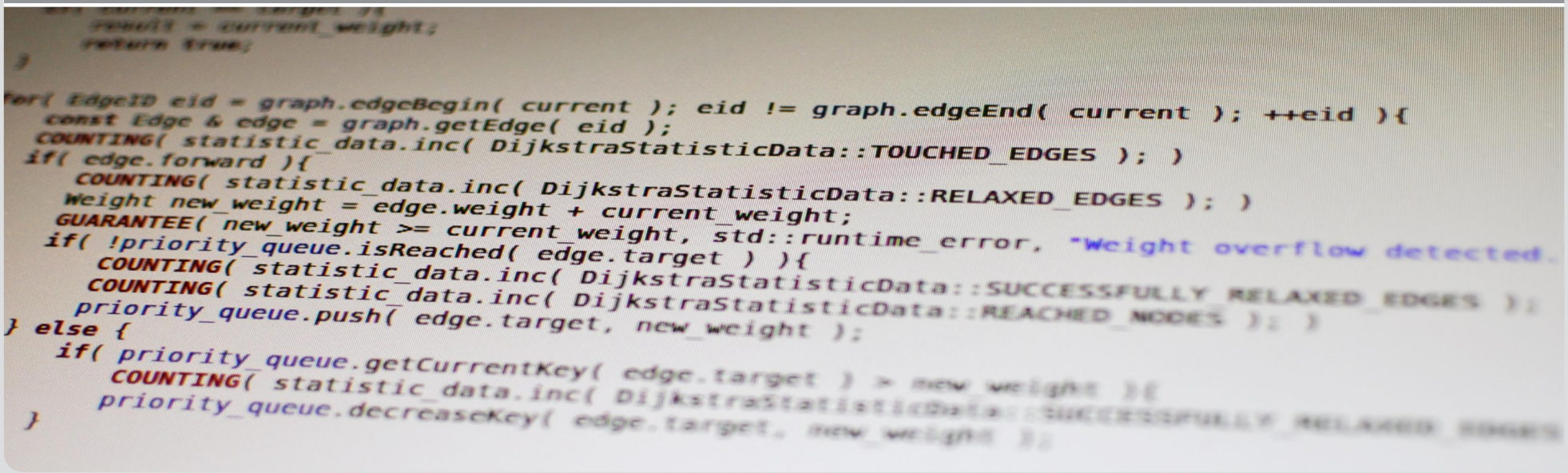
Karlsruher Institut für Technologie

Übung 8 – Algorithmen II

Moritz Laupichler, Hans-Peter Lehmann – *{moritz.laupichler, hans-peter.lehmann}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS22.php`

Institut für Theoretische Informatik - Algorithmik II



Organisatorisches

- Anmeldung zur Klausur im CAS ist freigeschaltet
- Besprechung ÜB 3 $\rightarrow$ nächstes Mal

Themenübersicht

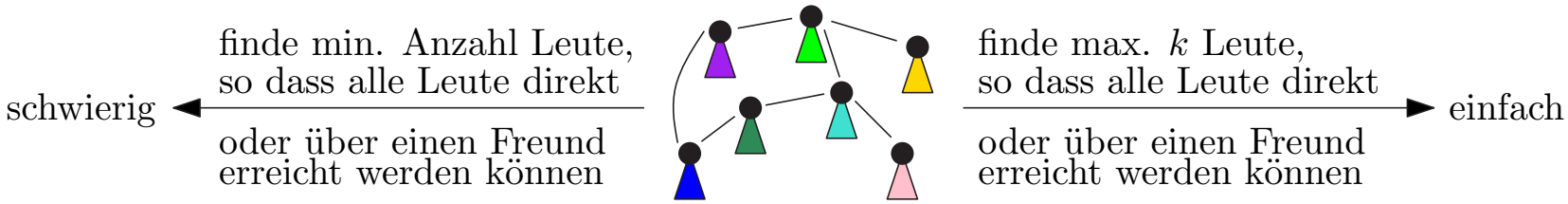
- Parameterisierte Algorithmen
- Parallele Algorithmen

Parametrisierte Algorithmen

***fixed parameter tractable* (FPT)**

Warum Probleme parametrisieren?

- es gibt “schwierige” Probleme z.B. Minimum Independent Set
  - → allgemeine Instanzen haben zu lange Berechnungszeit
- Kann man Spezialfälle eventuell effizient berechnen?
  - → Identifizierung zusätzlicher Parameter k der Problemstellung
  - → falls “Komplexität” in diesen Parametern k steckt, effiziente Lösungen für $k = const.$!



Parametrisierte Algorithmen

Definition

- Ein Problem heißt fixed parameter tractable, wenn es eine Laufzeit

$$T(n,k) = \mathcal{O}(f(k) \cdot \mathrm{p}(n))$$

hat, mit $f(\cdot)$ berechenbar, $p(\cdot)$ Polynom.

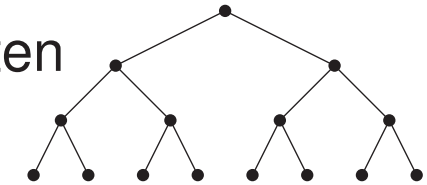
$f(\cdot)$ darf nicht von n abhängen und $p(\cdot)$ nicht von k; häufig Entscheidungsprobleme

Parametrisierte Algorithmen

Techniken

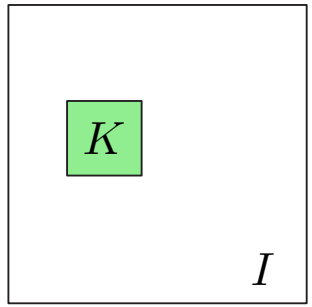
Tiefenbeschränkte Suche

- erschöpfendes Aufzählen und Testen aller Möglichkeiten
  $\rightarrow$ mit geeignetem Suchbaum beschränkter Tiefe
  k gibt Hinweis, wie weit man in die Tiefe gehen muss



Kernbildung

- Probleminstanz auf (schwierigen) Problemkern reduzieren
- Problemkern mit anderer Technik lösen



Parametrisierte Algorithmen

Schiebepuzzle

Problemstellung

- gegeben $n \times n$ Schiebepuzzle, $k \in \mathbb{N}$
- entscheide, ob das Puzzle in $\leq k$ Zügen gelöst werden kann
  - das Puzzle ist gelöst, wenn die Teile sortiert sind
  - Loch wird pro Zug eine Position horizontal oder vertikal verschoben

Algorithmus A

- es gibt ≤ 4 Möglichkeiten in jedem Zug, k Züge
  - baue Suchbaum (Höhe k, Verzweigungsgrad ≤ 4)
    $\rightarrow$ Baumgröße $\mathcal{O}(4^k)$
  - teste jeden Knoten auf korrekte Lösung
    $\rightarrow$ Aufwand $\mathcal{O}(n^2) \in \mathcal{O}(poly(n))$

24	8	13	12	20
11	2		17	21
7	15	14	19	5
6	10	3	9	1
4	23	11	18	22

$\Rightarrow$ Gesamtaufwand: $\mathcal{O}(4^k n^2) \Rightarrow$ FPT

$T(n,k) = 4T(n,k-1) + poly(n)$

Parametrisierte Algorithmen

Schiebepuzzle

Problemstellung

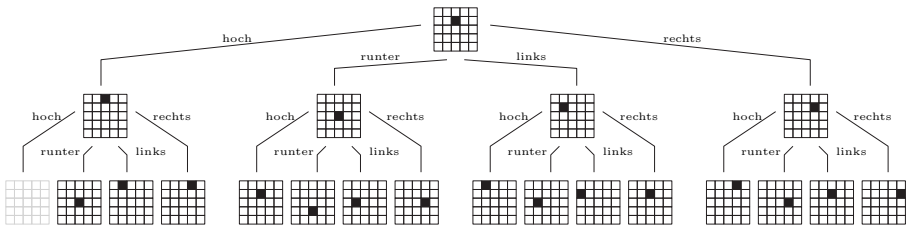
- gegeben $n \times n$ Schiebepuzzle, $k \in \mathbb{N}$
- entscheide, ob das Puzzle in $\leq k$ Zügen gelöst werden kann
  - das Puzzle ist gelöst, wenn die Teile sortiert sind
  - Loch wird pro Zug eine Position horizontal oder vertikal verschoben

Algorithmus A

- es gibt ≤ 4 Möglichkeiten in jedem Zug, k Züge
  - baue Suchbaum (Höhe k, Verzweigungsgrad ≤ 4)
    $\rightarrow$ Baumgröße $\mathcal{O}(4^k)$
  - teste jeden Knoten auf korrekte Lösung
    $\rightarrow$ Aufwand $\mathcal{O}(n^2) \in \mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtaufwand: $\mathcal{O}(4^k n^2) \Rightarrow$ FPT

$T(n,k) = 4T(n,k-1) + poly(n)$

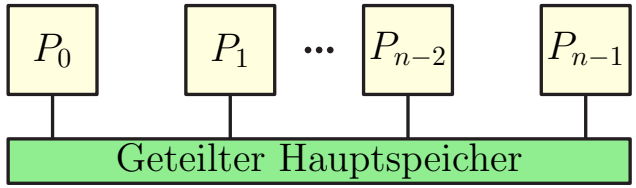


Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



(symmetrisch) gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

Bulk**S**ynchronous**P**arallel

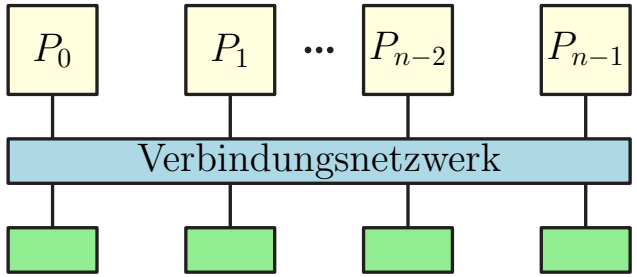
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



(symmetrisch) gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

Bulk**S**ynchronous**P**arallel

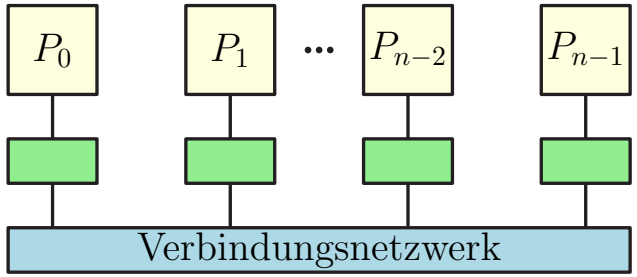
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



(symmetrisch) gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

Bulk**S**ynchronous**P**arallel

- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Verbindungsnetzwerke

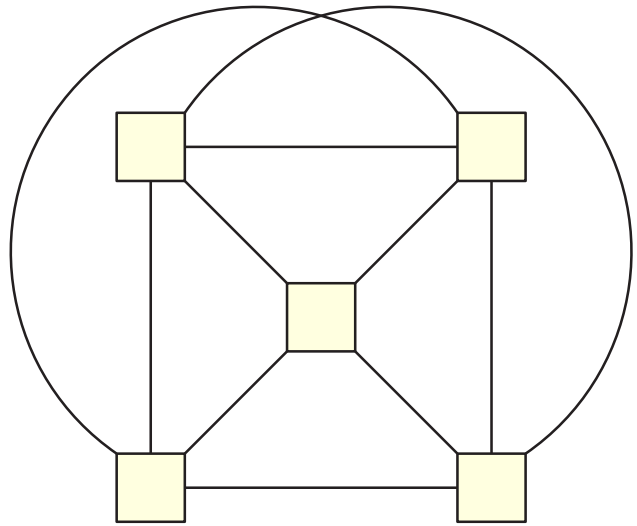
Struktur

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

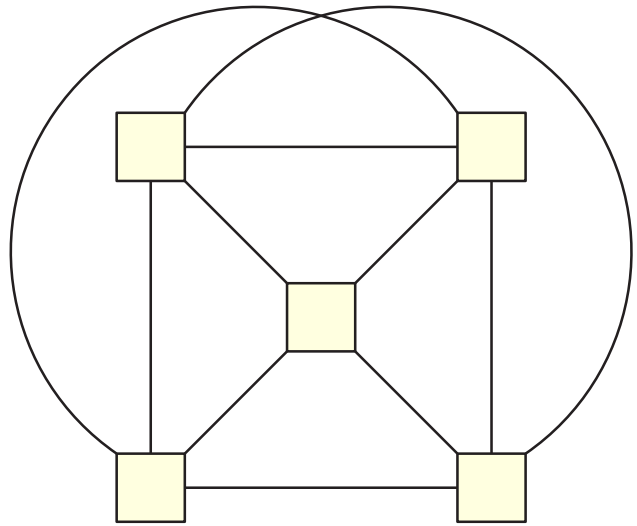
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

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

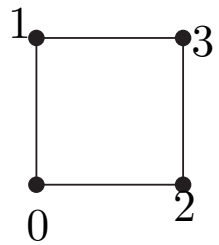
Struktur

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

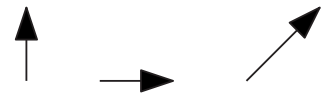
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

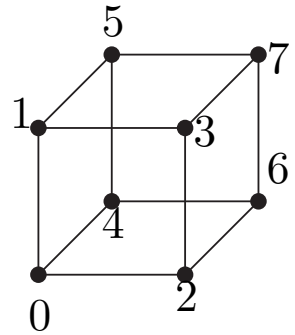
Vollverkabelt

- 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

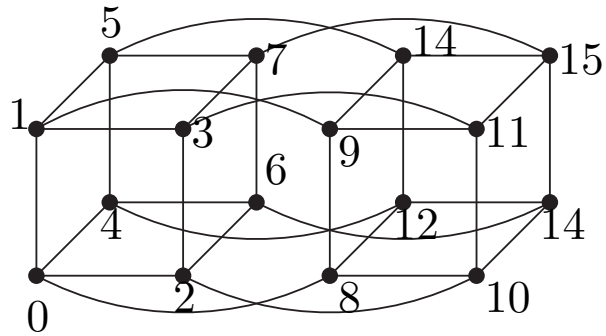
Struktur

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

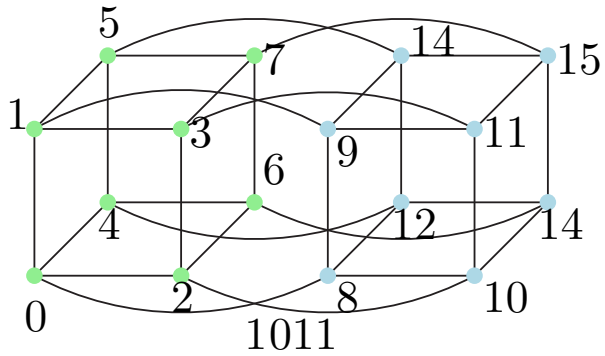
Struktur

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

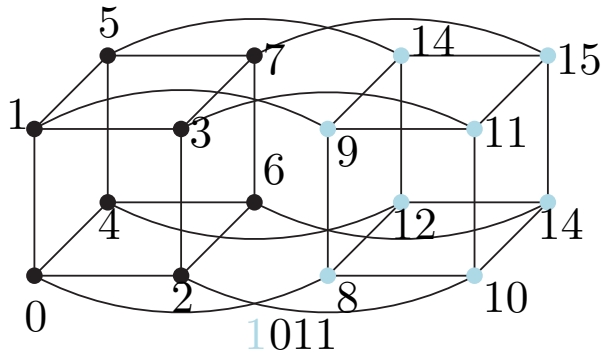
Struktur

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

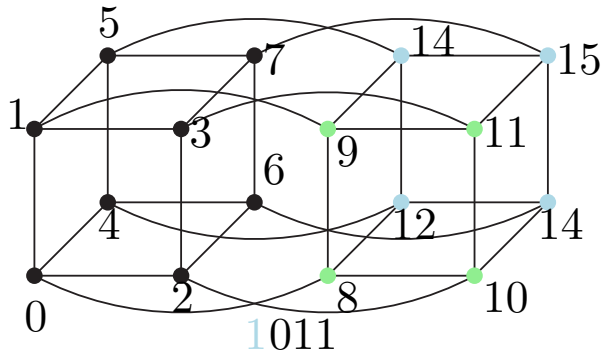
Struktur

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

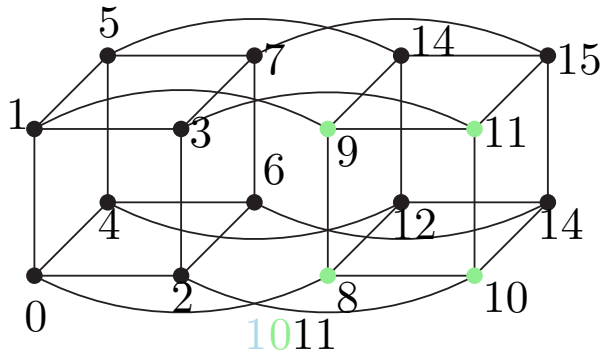
Struktur

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

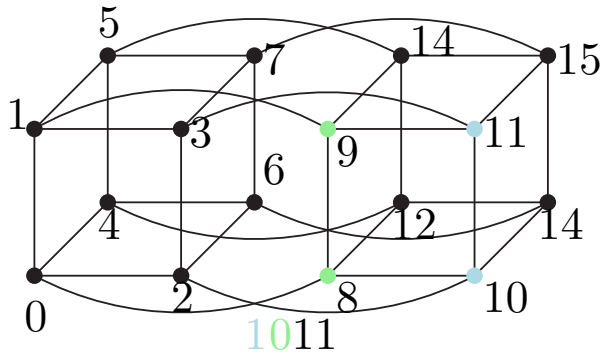
Struktur

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

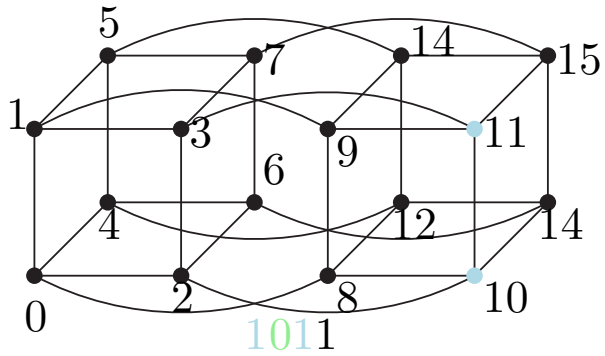
Struktur

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

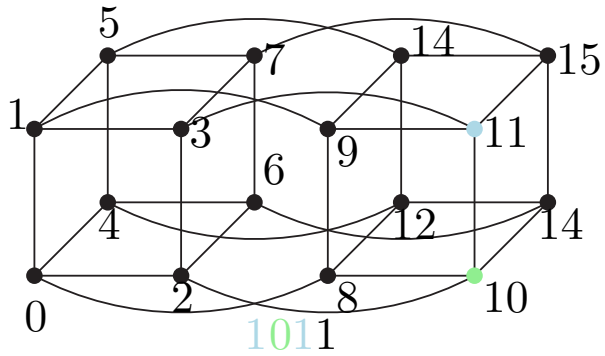
Struktur

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

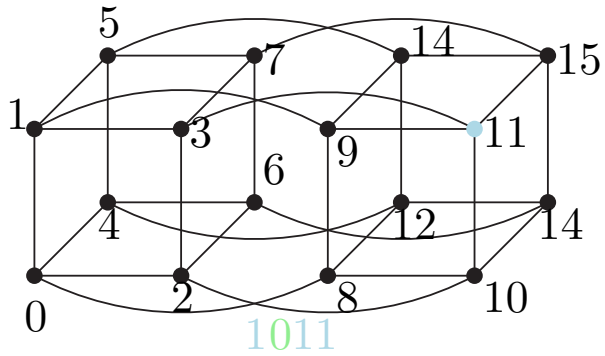
Struktur

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
  $***1** \leftrightarrow ***0**$

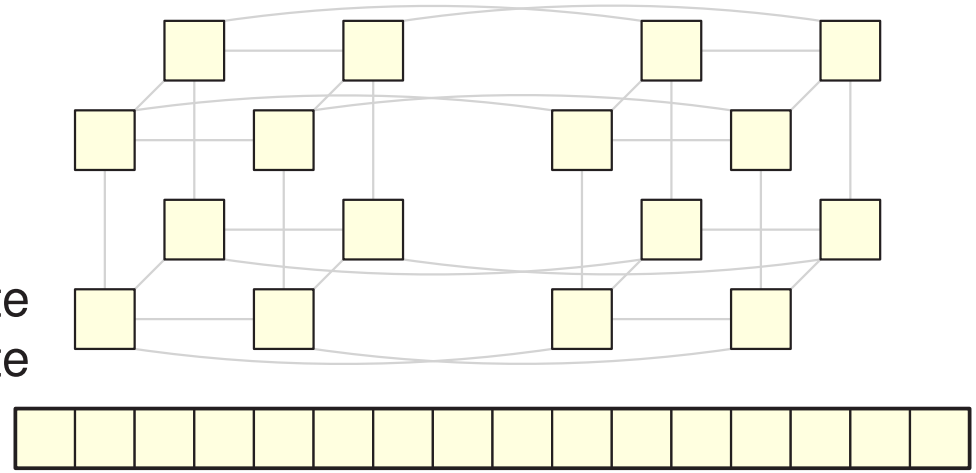


Kosten

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

Anwendungen

Präfixsumme - Hypercube



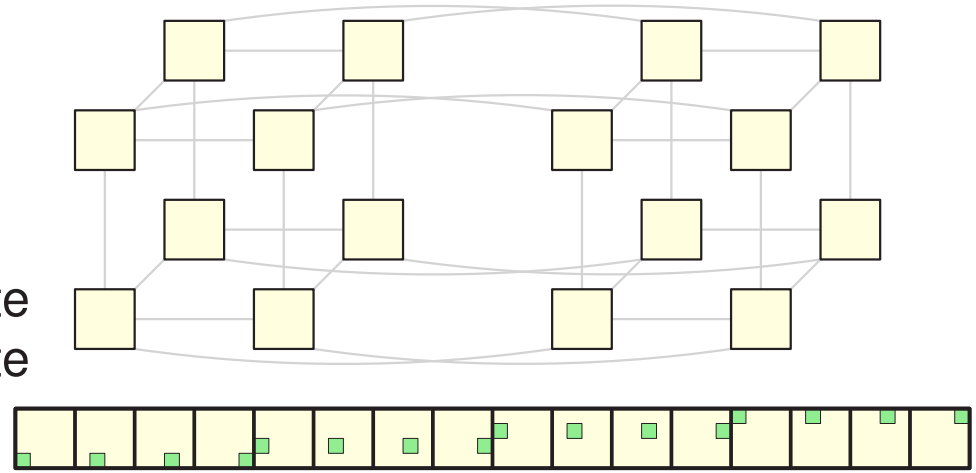
- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme

- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

Präfixsumme - Hypercube



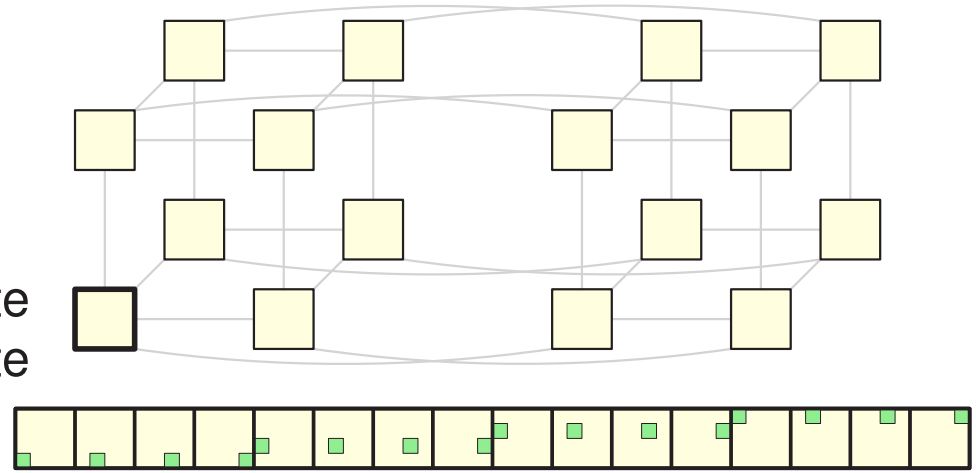
- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme

- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

Präfixsumme - Hypercube



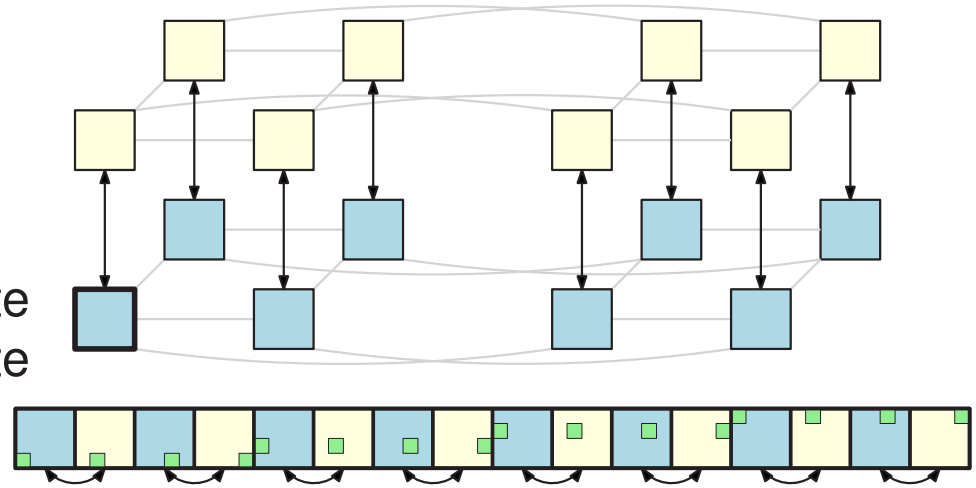
- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme

- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

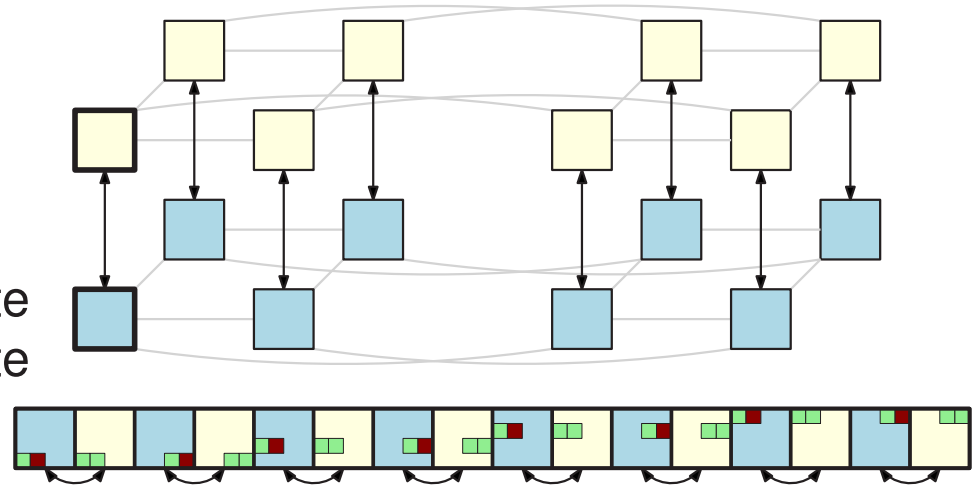
Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

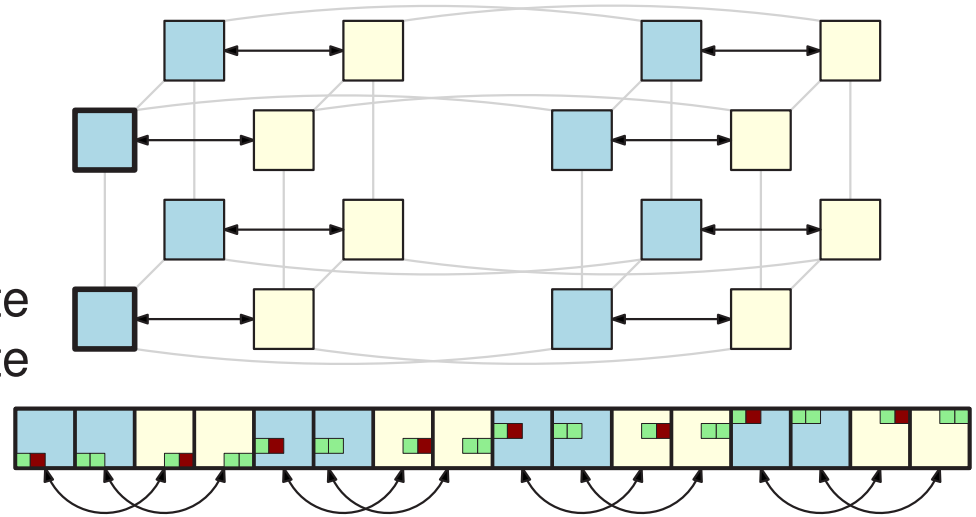
Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

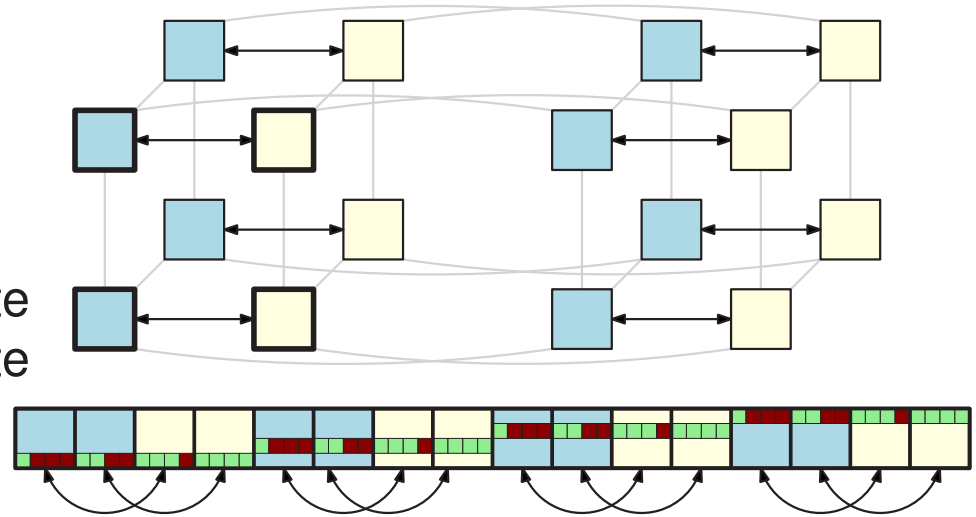
Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

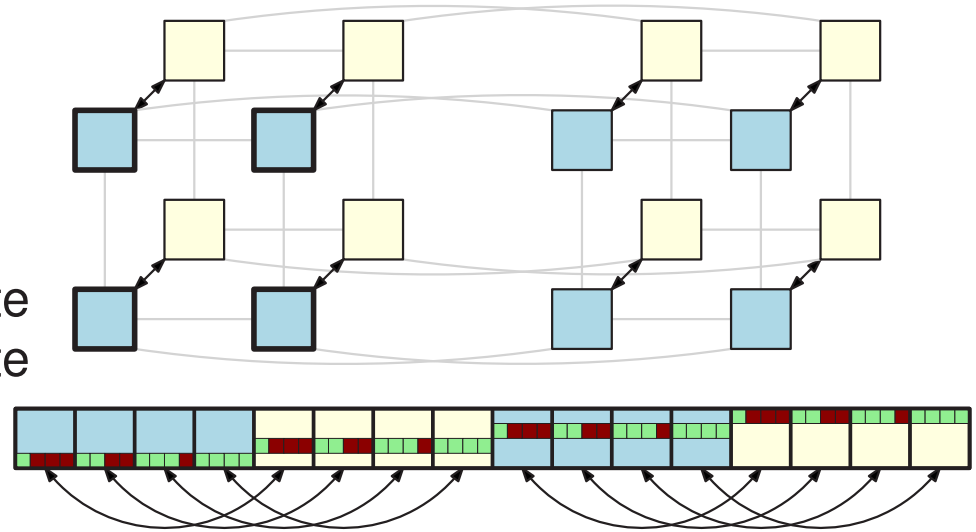
Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

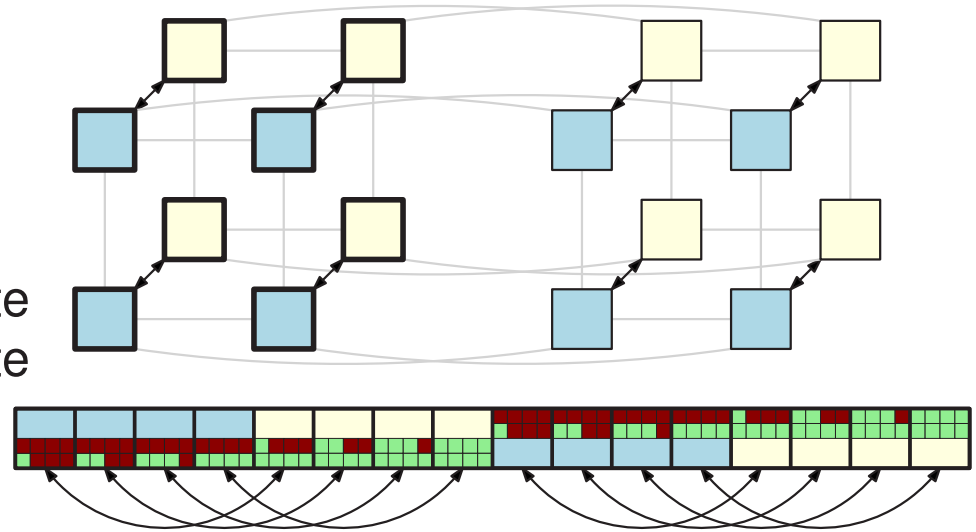
Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

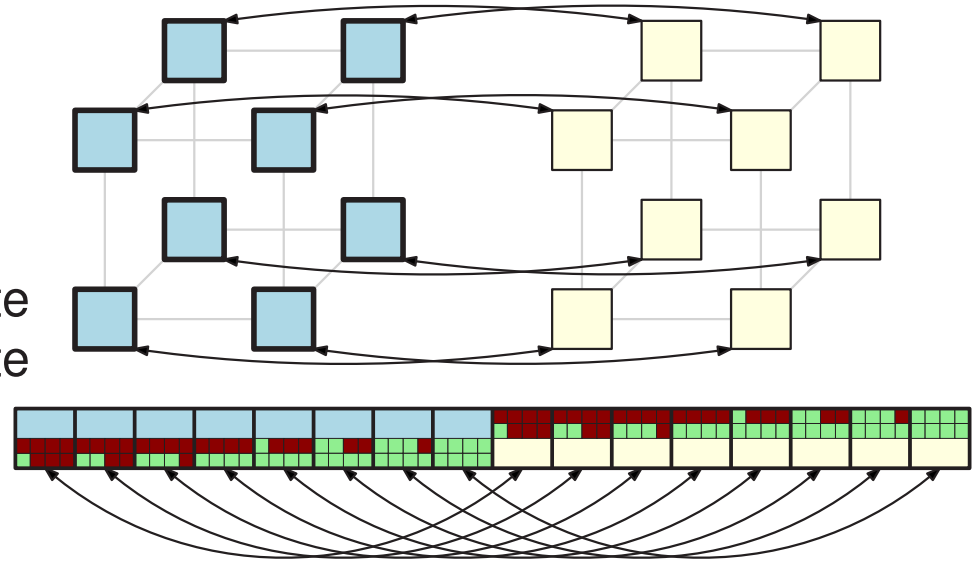
Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

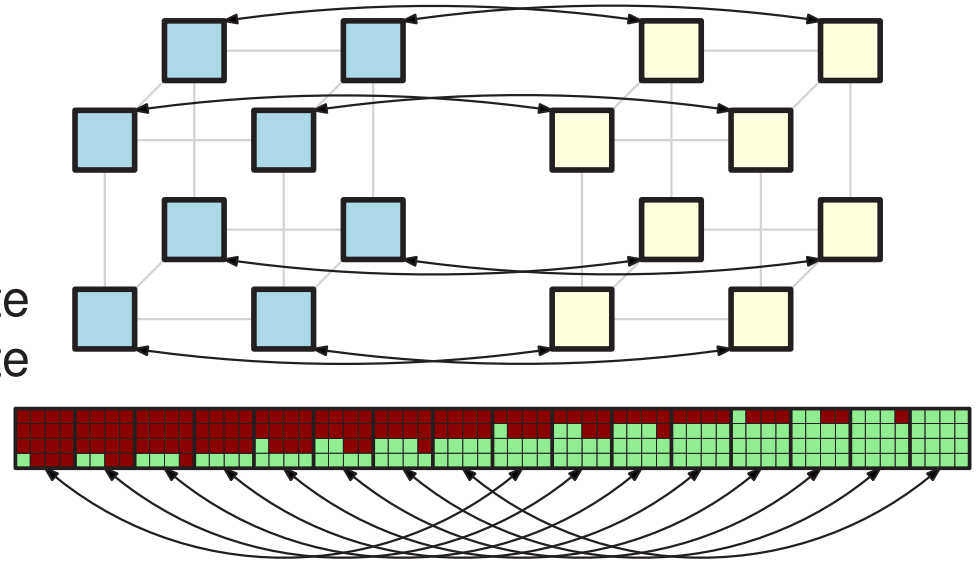
Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

Präfixsumme - Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

PRAM Präfixsumme

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

Aufwärtsphase

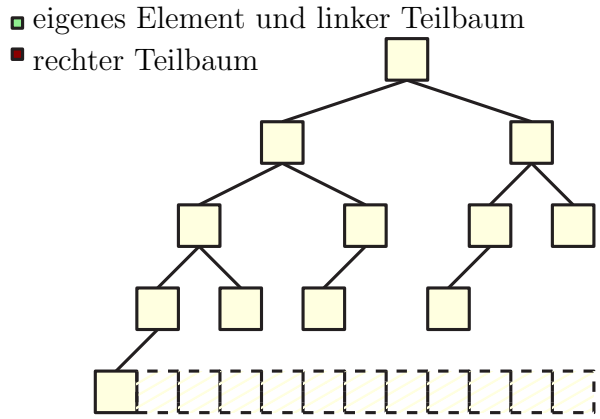
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

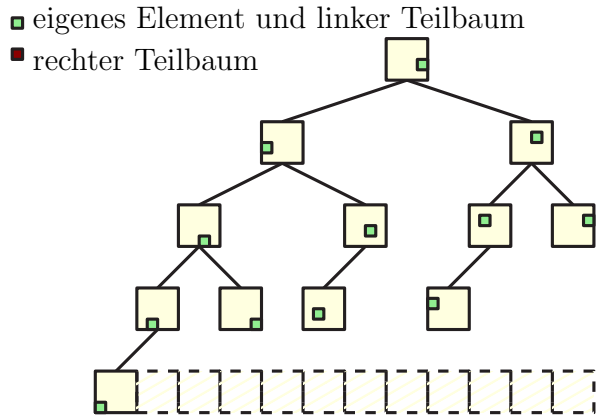
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

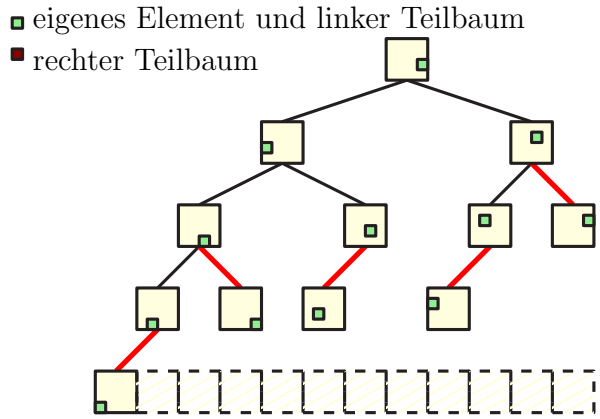
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

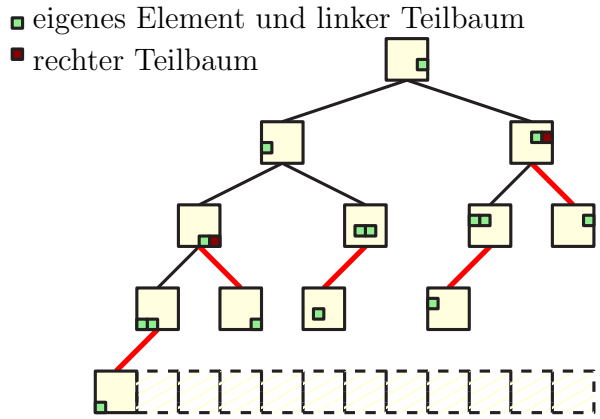
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

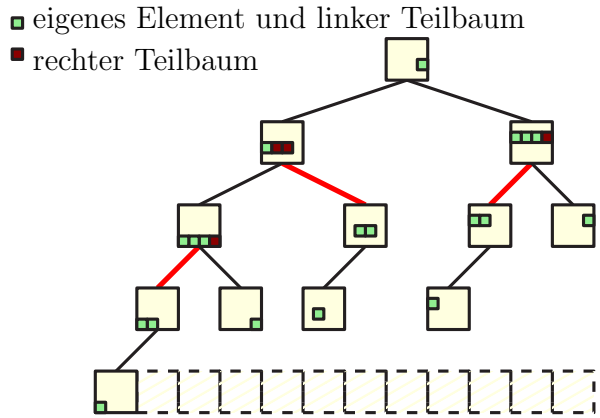
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

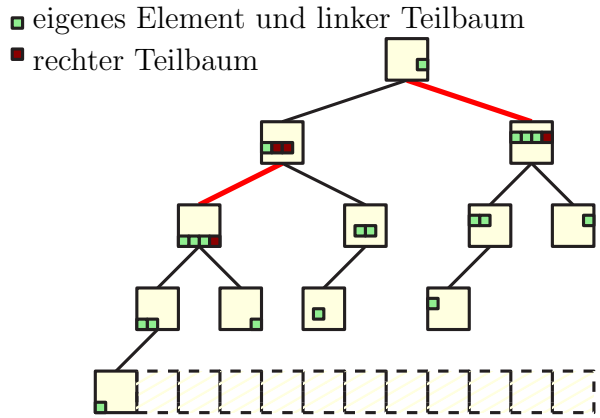
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

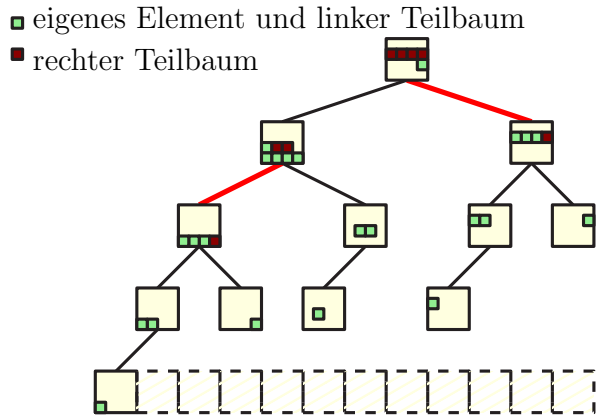
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

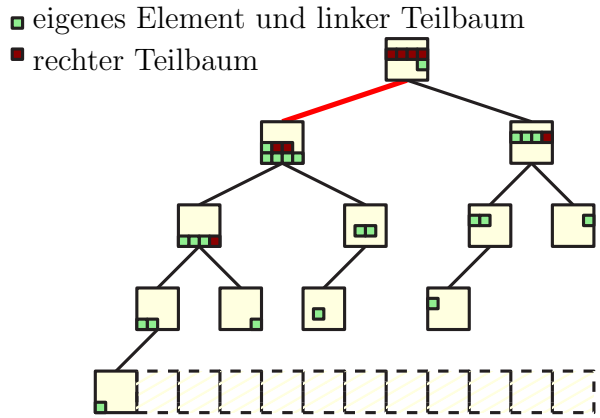
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

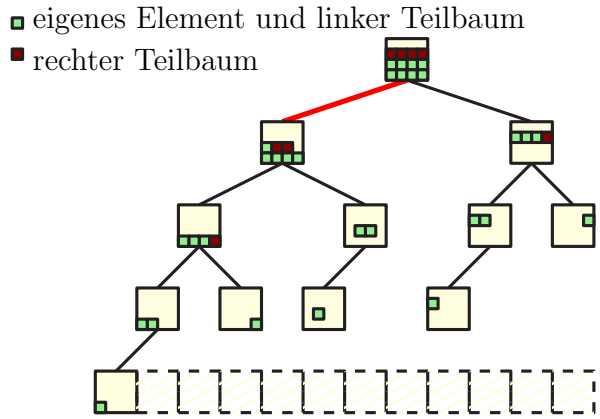
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

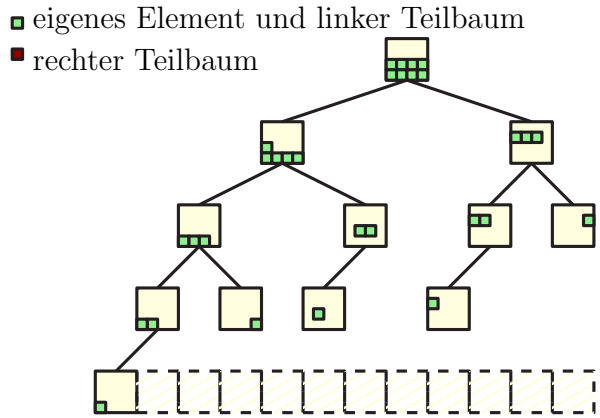
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

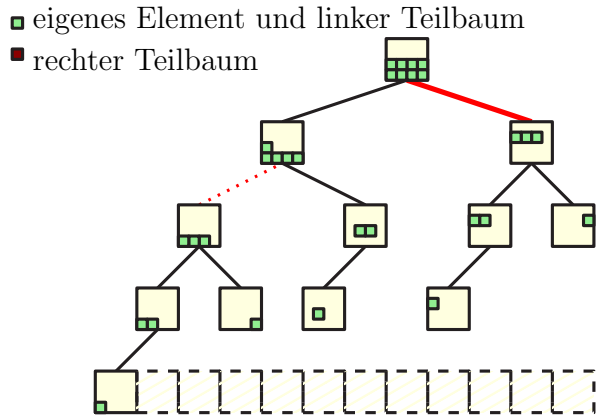
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

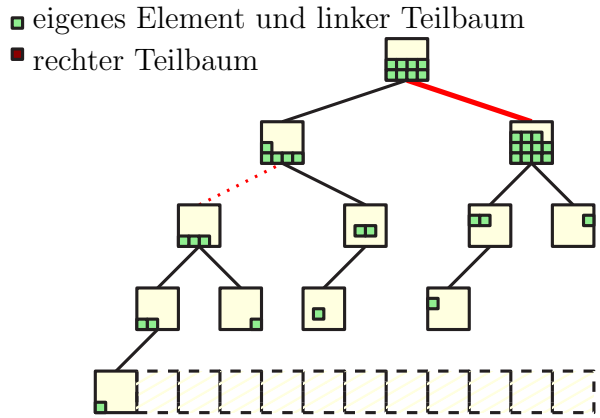
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

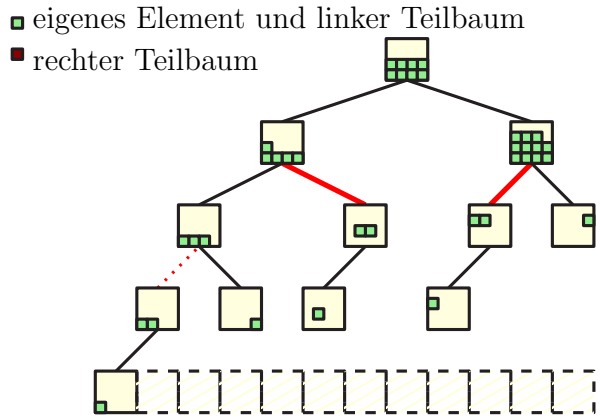
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

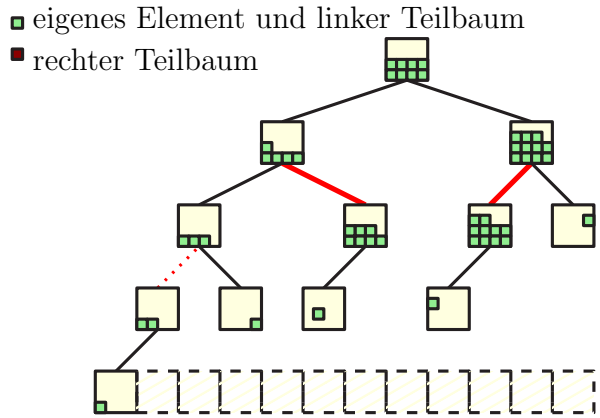
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

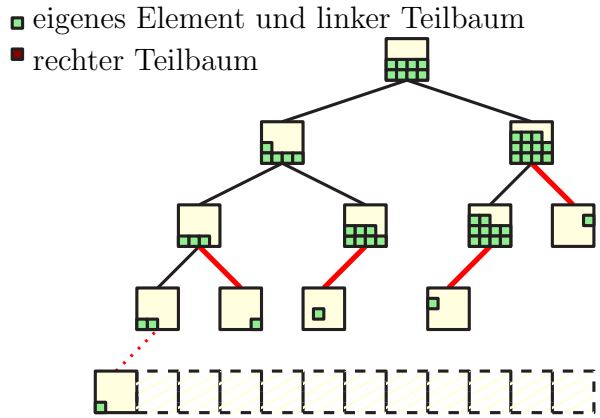
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

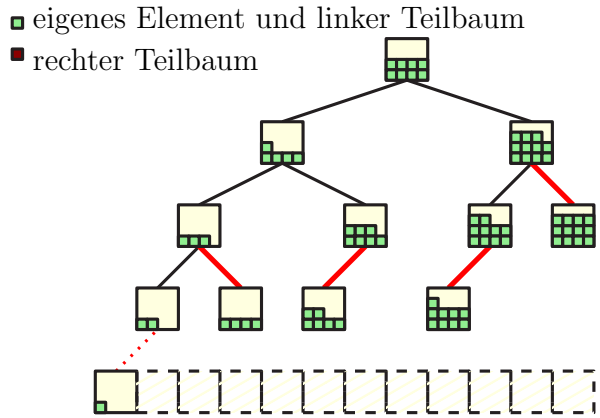
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer linker Teilbaum und größerer rechter Teilaum Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

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

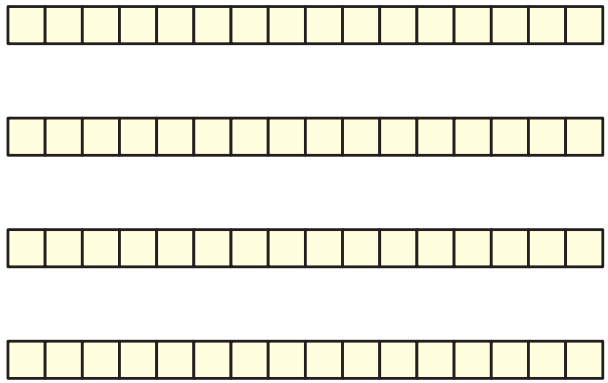
- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

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

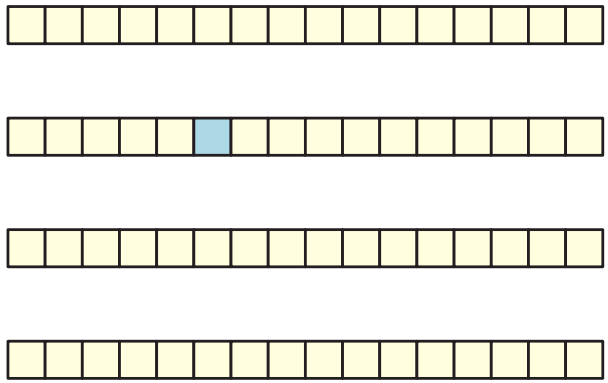


Anwendungen

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

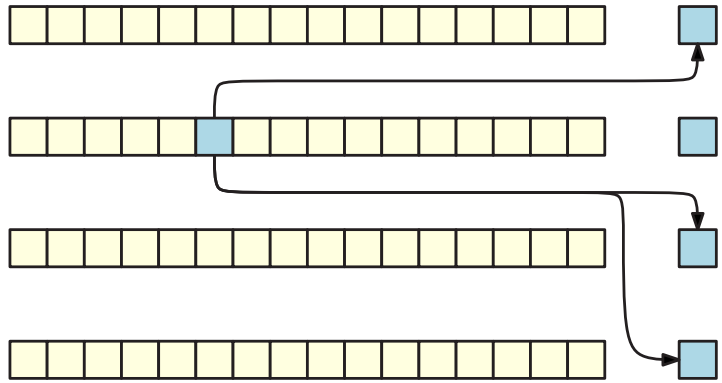


Anwendungen

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

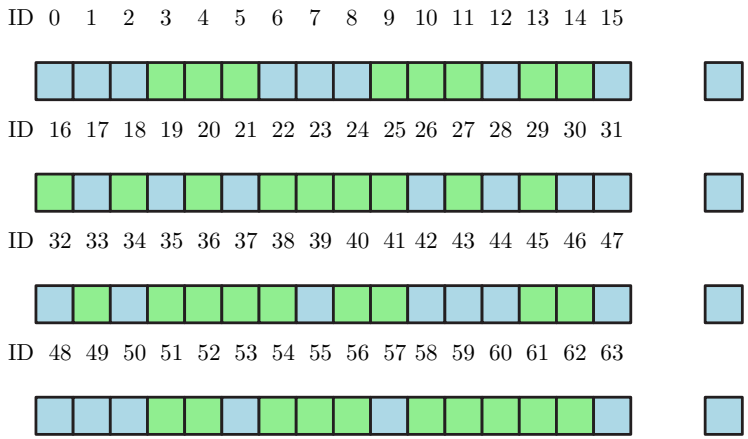


Anwendungen

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

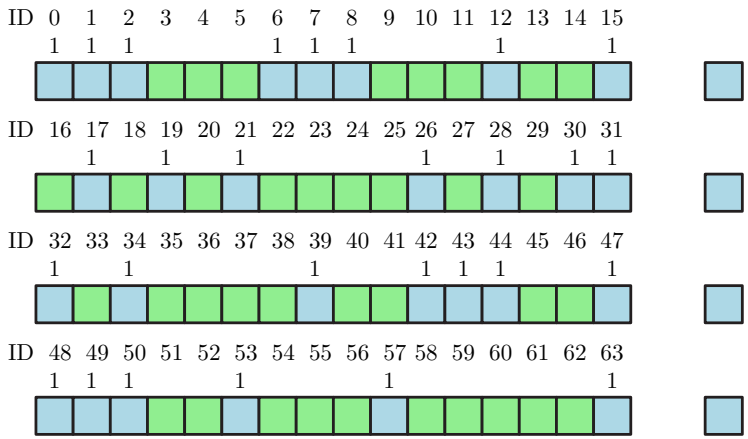


Anwendungen

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

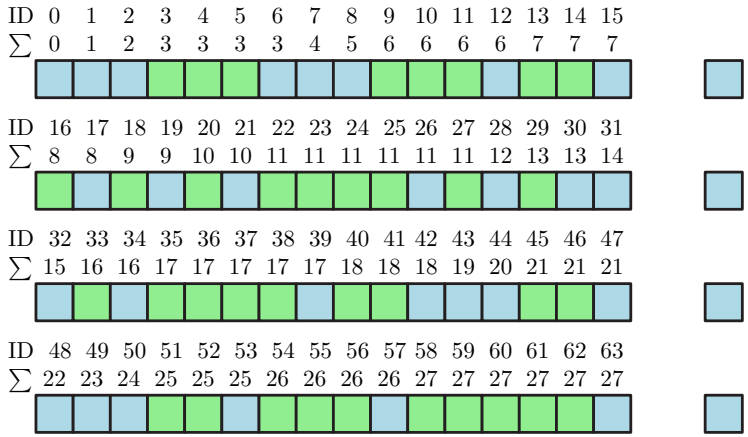


Anwendungen

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

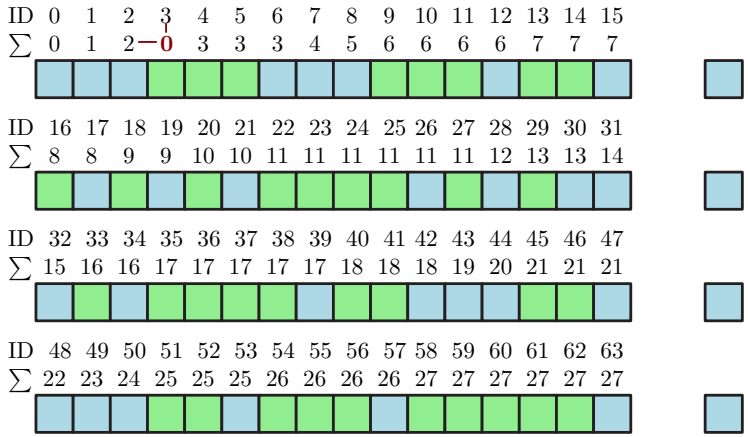


Anwendungen

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

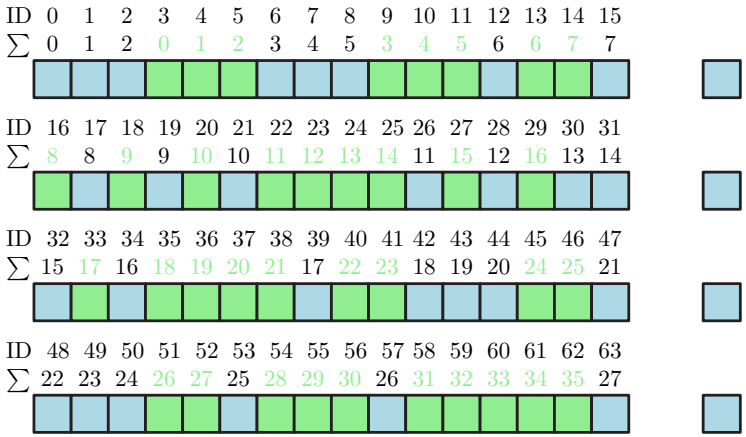


Anwendungen

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

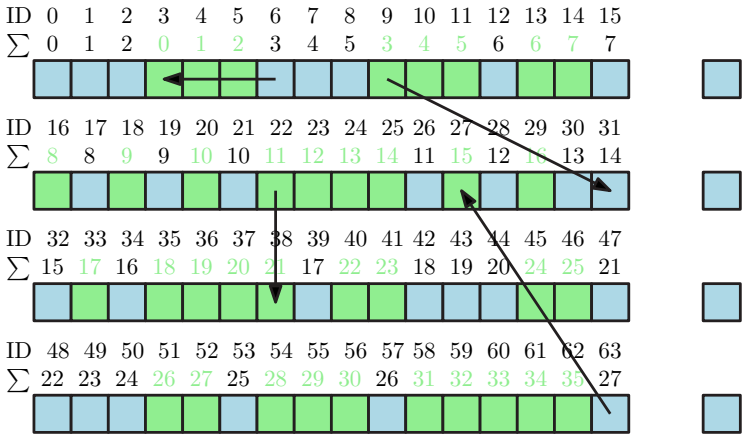


Anwendungen

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

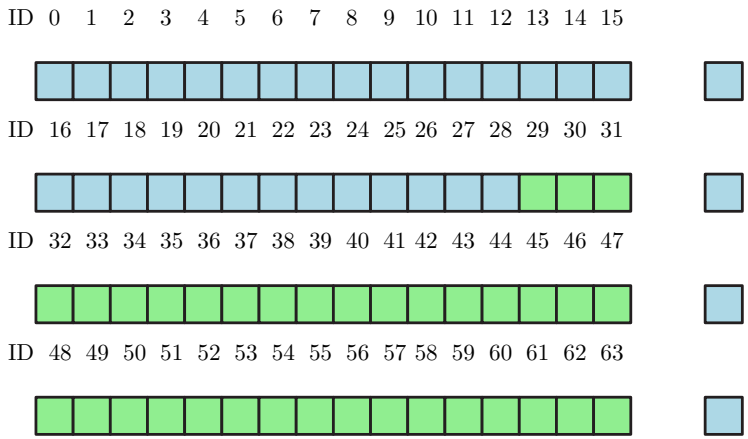


Anwendungen

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

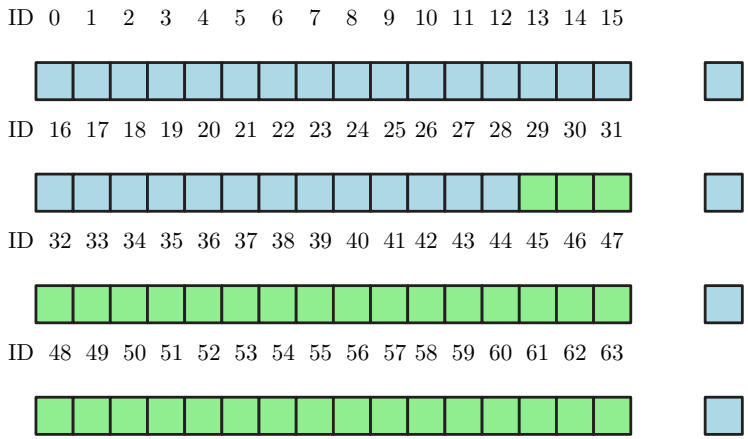


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

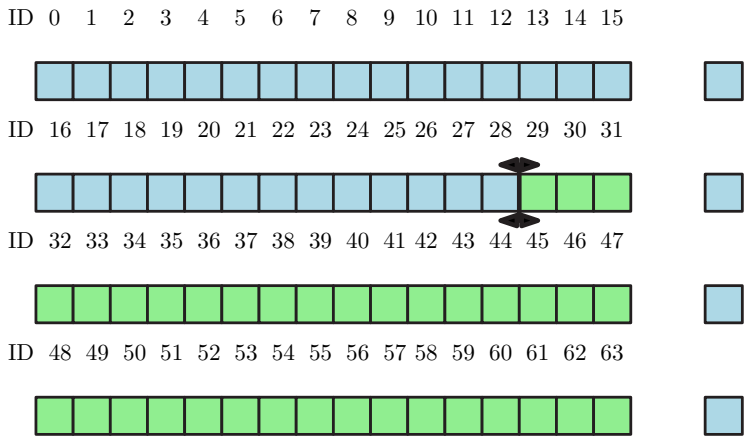
1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer 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

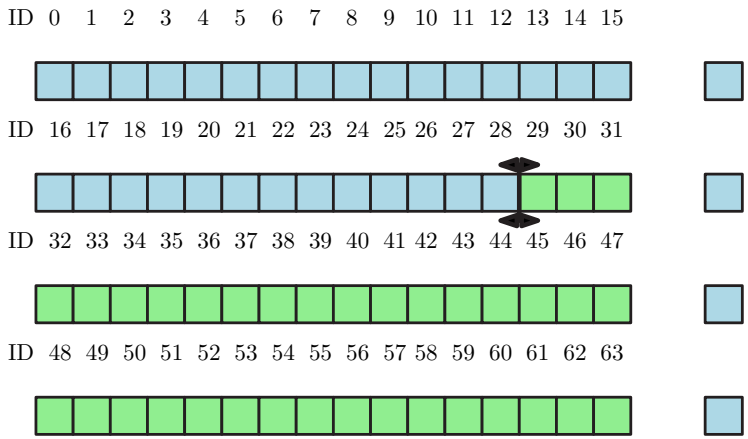


1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer 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
- Intel Thread Building Block Library (TBB)
  - `https://software.intel.com/en-us/tbb`
  - mehr objektorientiert als *OpenMP*
  - enthält konkurrente Datenstrukturen und viele parallele Primitive
    - Konkurrente *Queues*, *Arrays*, ...
    - Parallel *Sort*, *For*, *While*, ...
    - Komplexe *Dataflow*-Graphen
    - Geschachtelter und rekursiver Parallelismus
- Message Passing Interface (MPI)
  - `https://www.mcs.anl.gov/research/projects/mpi/`
  - Standard für verteilte Programmierung
  - implementiert die gängisten Kommunikationsprimitiven (`C`-Style Interface)

Ende!

Feierabend!