Rechenmodell

Parallel Random Access Machine (PRAM)

- p (sequentielle) Prozessoren (*processing elements*, PEs)
- Gemeinsamer globaler Speicher
- Synchroner Takt
- Globaler Speicher kann prinzipiell von allen Prozessoren gleichzeitig benutzt werden (lesend / schreibend)
- Evtl. Einschränkungen hinsichtlich gleichzeitiger Benutzung derselben Speicherstelle
- Single Instruction Multiple Data (SIMD) (aber MIMD auch möglich)
- feinkörnig (niedriger Quotient aus Kommunikationskosten und Berechnungskosten)

PRAM-Modelle

(der Vollständigkeit halber)

Handhabung des gleichzeitigen Zugriffs auf dieselbe Speicherstelle:

- Exclusive Read, Exclusive Write: EREW
- Concurrent Read, Exclusive Write: CREW
- Concurrent Read, Concurrent Write: CRCW

Ziele von PRAM-Algorithmen

Entwurfsmotivation:

- **Nicht:** Hoher Realismus
- Nützliche algorithmische parallele Konzepte entwerfen
- Grad an Parallelität in Problem bestimmen

Ziele von PRAM-Algorithmen

Entwurfsmotivation:

- **Nicht:** Hoher Realismus
- Nützliche algorithmische parallele Konzepte entwerfen
- Grad an Parallelität in Problem bestimmen

Ziel für Algorithmus:
Schnell fertig sein mit "vernünftigem Gesamtaufwand"!

Summe auf der PRAM

Erste Version

- **Gegeben:** Zahlenfolge $a_1, \ldots, a_n$
- **Gesucht:** $\sum_{i=1}^{n} a_i$

Summe auf der PRAM

Erste Version

- **Gegeben:** Zahlenfolge $a_1, \ldots, a_n$
- **Gesucht:** $\sum_{i=1}^{n} a_i$
- Zwei PEs: Jeder summiert die Hälfte, einer die beiden Teilergebnisse
- Zeit $O(n/2) = O(n)$ mit 2 PEs

Summe auf der PRAM

Erste Version

- **Gegeben:** Zahlenfolge $a_1, \ldots, a_n$
- **Gesucht:** $\sum_{i=1}^{n} a_i$
- Zwei PEs: Jeder summiert die Hälfte, einer die beiden Teilergebnisse
- Zeit $O(n/2) = O(n)$ mit 2 PEs
- 4 PEs: Jeder summiert ein Viertel, Teilergebnisse wie vorher
- Zeit $O(n/4) = O(n)$ mit 4 PEs

Summe auf der PRAM

Erste Version

- **Gegeben:** Zahlenfolge $a_1, \ldots, a_n$
- **Gesucht:** $\sum_{i=1}^{n} a_i$
- Zwei PEs: Jeder summiert die Hälfte, einer die beiden Teilergebnisse
- Zeit $O(n/2) = O(n)$ mit 2 PEs
- 4 PEs: Jeder summiert ein Viertel, Teilergebnisse wie vorher
- Zeit $O(n/4) = O(n)$ mit 4 PEs

Gretchenfrage: Wie weit können wir das treiben?

Summe auf der PRAM

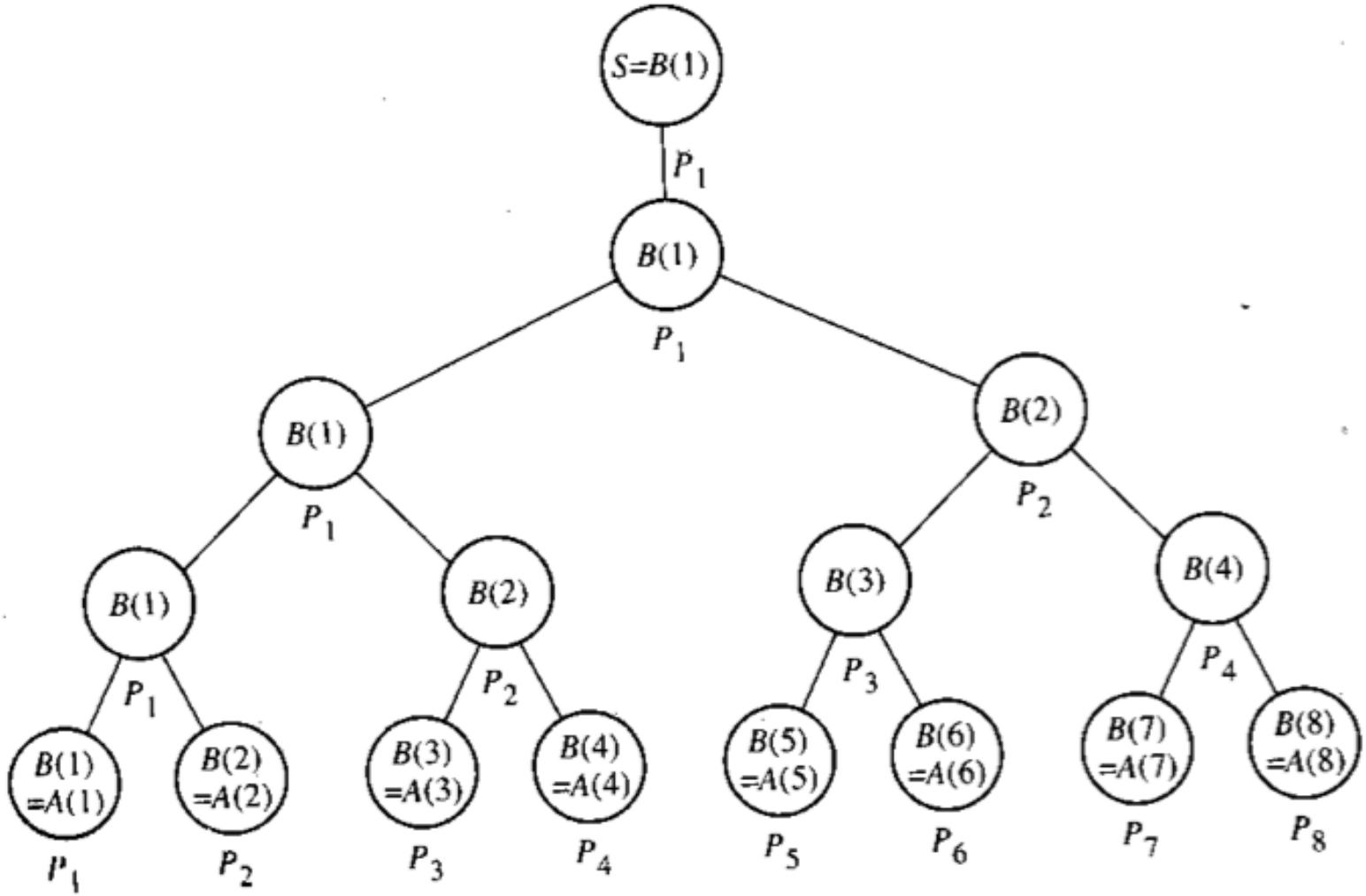
n Prozessoren mit Index $1 \leq i \leq n$

Function PRAM-Sum(A : array of size $n = 2^k$)

```
// copy A into B
global read(A(i), a)
global write(a, B(i))
for h = 1 to log n do
    if (i ≤ n/2^h) then
        global read(B(2i − 1), x)
        global read(B(2i), y)
        z = x + y
        // Write into first half of current array part
        global write(z, B(i))
if (i = 1) then
    global write(z, S)
return S
```

Summe auf der PRAM

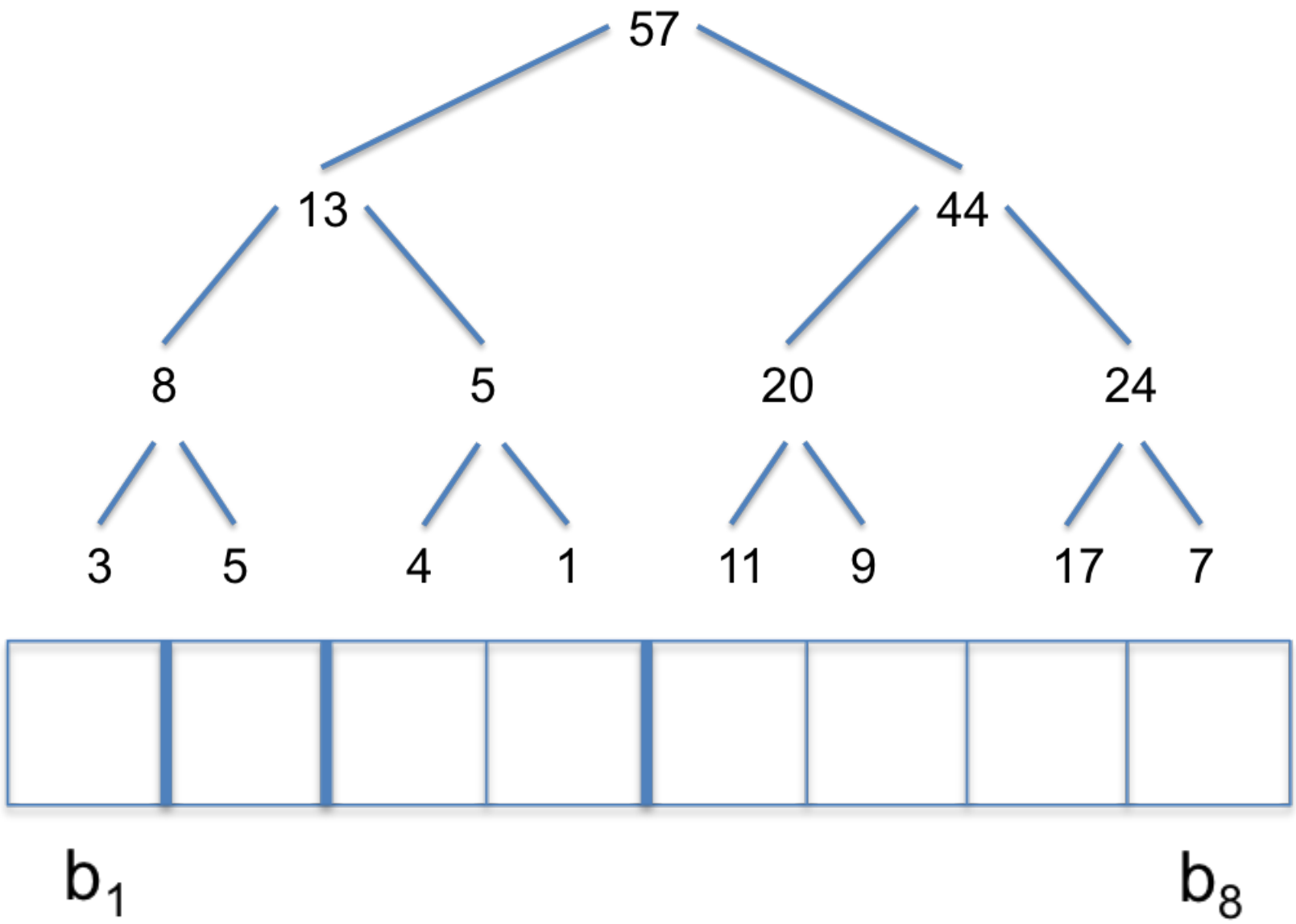
Illustration



[Jájá: An Introduction to Parallel Algorithms. S. 14, Abb. 1.4]

Summe auf der PRAM

Beispiel



Summe auf der PRAM

Diskussion

- Jede Baumebene wird parallel abgearbeitet

$\Rightarrow$ Jede Baumebene braucht konstante parallele Zeit

$\Rightarrow$ Parallele Laufzeit $O(\log n)$ (Baumhöhe)

Summe auf der PRAM

Diskussion

- Jede Baumebene wird parallel abgearbeitet

$\Rightarrow$ Jede Baumebene braucht konstante parallele Zeit

$\Rightarrow$ Parallele Laufzeit $O(\log n)$ (Baumhöhe)

- **Aber:** n PEs unrealistisch (und unnötig)
- Teuer: n PEs bei Laufzeit $O(\log n)$

Summe auf der PRAM

Diskussion

- Jede Baumebene wird parallel abgearbeitet

$\Rightarrow$ Jede Baumebene braucht konstante parallele Zeit

$\Rightarrow$ Parallele Laufzeit $O(\log n)$ (Baumhöhe)

- **Aber:** n PEs unrealistisch (und unnötig)
- Teuer: n PEs bei Laufzeit $O(\log n)$

Aber was heißt teuer, was günstig?

Bewertung paralleler Algorithmen

Man könnte die Zahl der Speicherzugriffe zählen...

- Die sind in der Realität besonders teuer...
- Aber: Operationen verkomplizieren den Pseudocode

Bewertung paralleler Algorithmen

Man könnte die Zahl der Speicherzugriffe zählen...

- Die sind in der Realität besonders teuer...
- Aber: Operationen verkomplizieren den Pseudocode

Asymptotisch äquivalent:

- $P(n)$ Prozessoren und $T(n)$ Laufzeit
- $C(n) = P(n)T(n)$ Kosten und $T(n)$ Laufzeit
- $O(T(n)P(n)/p)$ Laufzeit für beliebige Zahl $p \leq P(n)$ von Prozessoren
- $O(\frac{C(n)}{p} + T(n))$ Laufzeit für beliebige Zahl p von Prozessoren

Das Prinzip von Arbeit und Laufzeit

(Work-Time-Principle)

- **Arbeit** (work): Gesamtzahl der durchgeführten Operationen (über alle Prozessoren)
- **Laufzeit** (time): Zahl von synchronen Zeitschritten, nach denen der Algorithmus terminiert

Das Prinzip von Arbeit und Laufzeit
(Work-Time-Principle)

- **Arbeit** (work): Gesamtzahl der durchgeführten Operationen (über alle Prozessoren)
- **Laufzeit** (time): Zahl von synchronen Zeitschritten, nach denen der Algorithmus terminiert

Grober Entwurf:

- Beschreibe Algorithmen als Folge von Zeitschritten.
- Jeder Zeitschritt kann eine beliebige Zahl von nebenläufigen Operationen beinhalten.

Das Prinzip von Arbeit und Laufzeit
(Work-Time-Principle)

- **Arbeit** (work): Gesamtzahl der durchgeführten Operationen (über alle Prozessoren)
- **Laufzeit** (time): Zahl von synchronen Zeitschritten, nach denen der Algorithmus terminiert

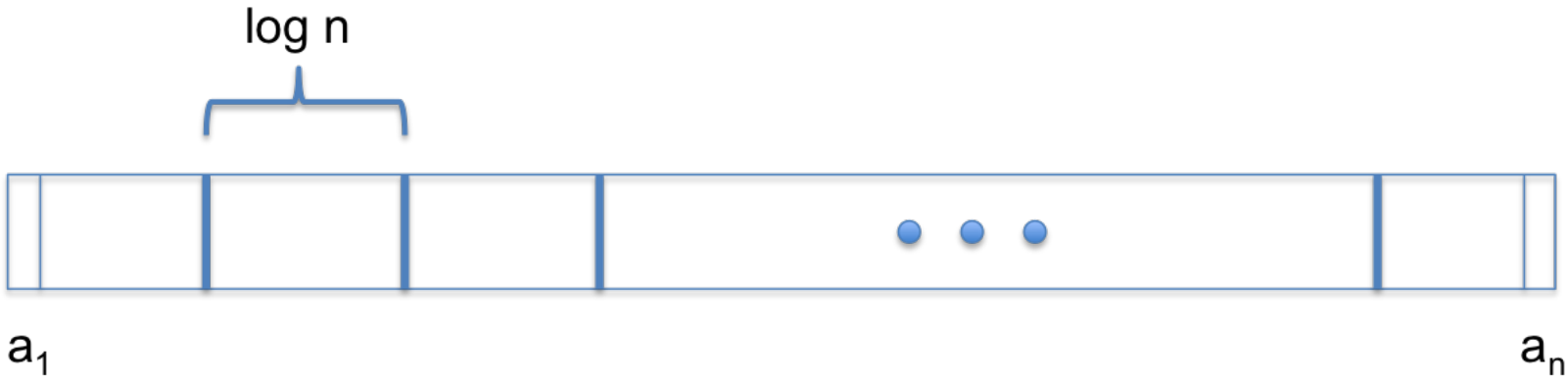
Grober Entwurf:

- Beschreibe Algorithmen als Folge von Zeitschritten.
- Jeder Zeitschritt kann eine beliebige Zahl von nebenläufigen Operationen beinhalten.

Ziel: Parameter p loswerden, allgemeine Lösung

Summe auf der PRAM

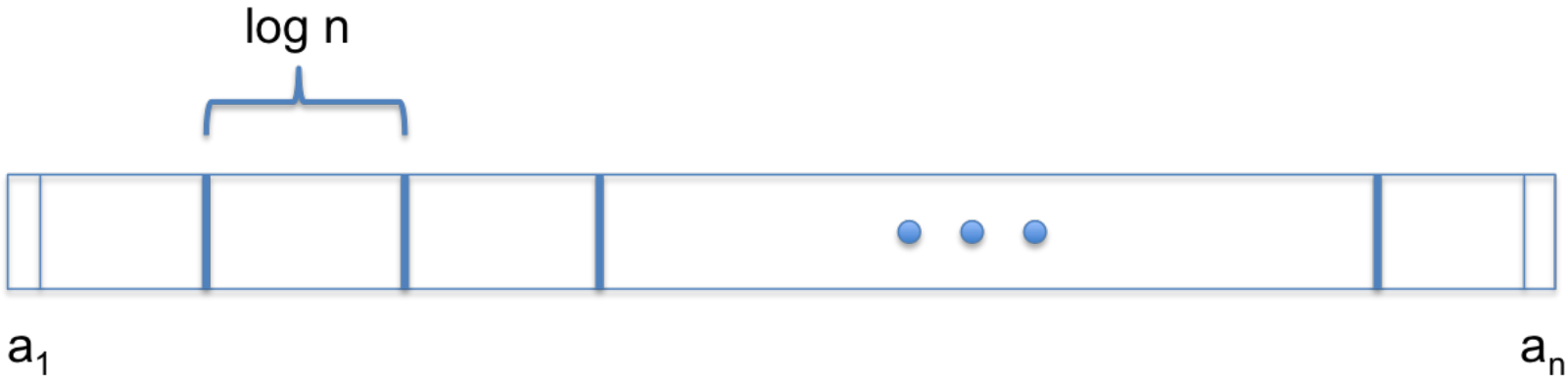
Zweite Version, noch unter Angabe der Prozessorenzahl



- $O(n/\log n)$ Prozessoren, für jeden einen Abschnitt
- Abschnittsgröße: $O(\log n)$
- In jedem Abschnitt sequentielle Summenbildung

Summe auf der PRAM

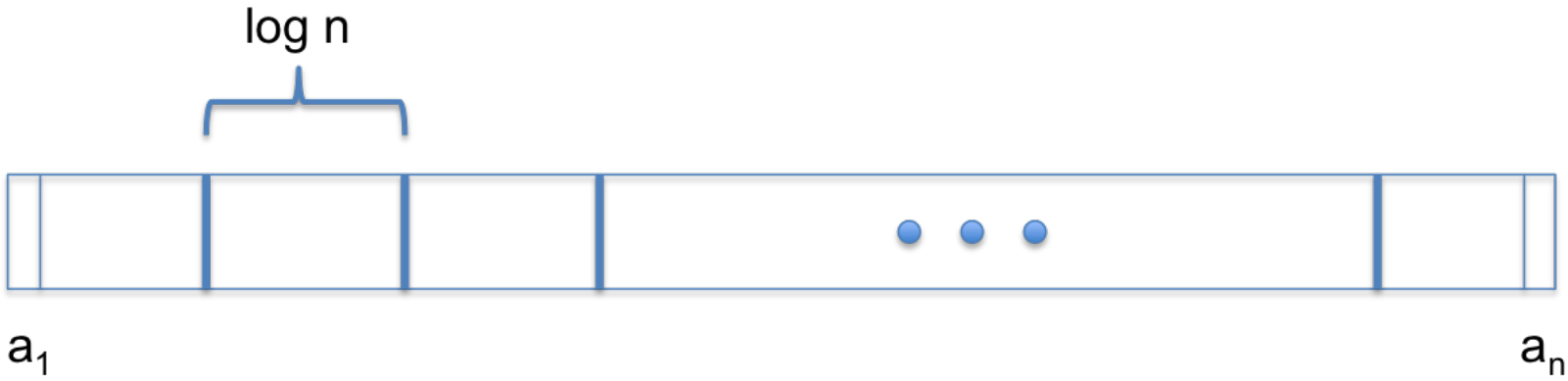
Zweite Version, noch unter Angabe der Prozessorenzahl



- $O(n/\log n)$ Prozessoren, für jeden einen Abschnitt
- Abschnittsgröße: $O(\log n)$
- In jedem Abschnitt sequentielle Summenbildung
- Wie summiert man die Teilergebnisse jedes Abschnitts?

Summe auf der PRAM

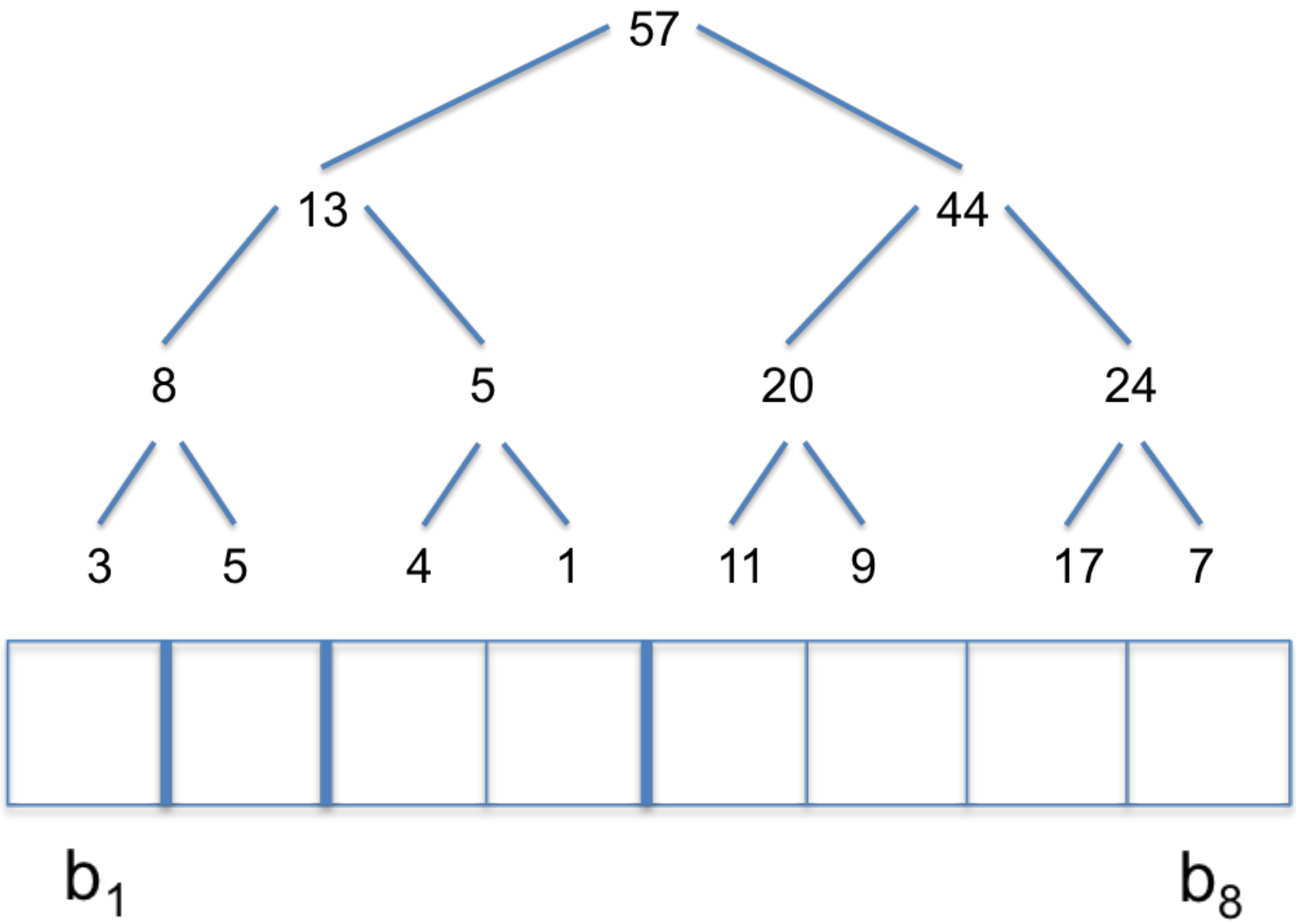
Zweite Version, noch unter Angabe der Prozessorenzahl



- $O(n/\log n)$ Prozessoren, für jeden einen Abschnitt
- Abschnittsgröße: $O(\log n)$
- In jedem Abschnitt sequentielle Summenbildung
- Wie summiert man die Teilergebnisse jedes Abschnitts?
- Idee: Rekursion!

Summe auf der PRAM

Beispiel



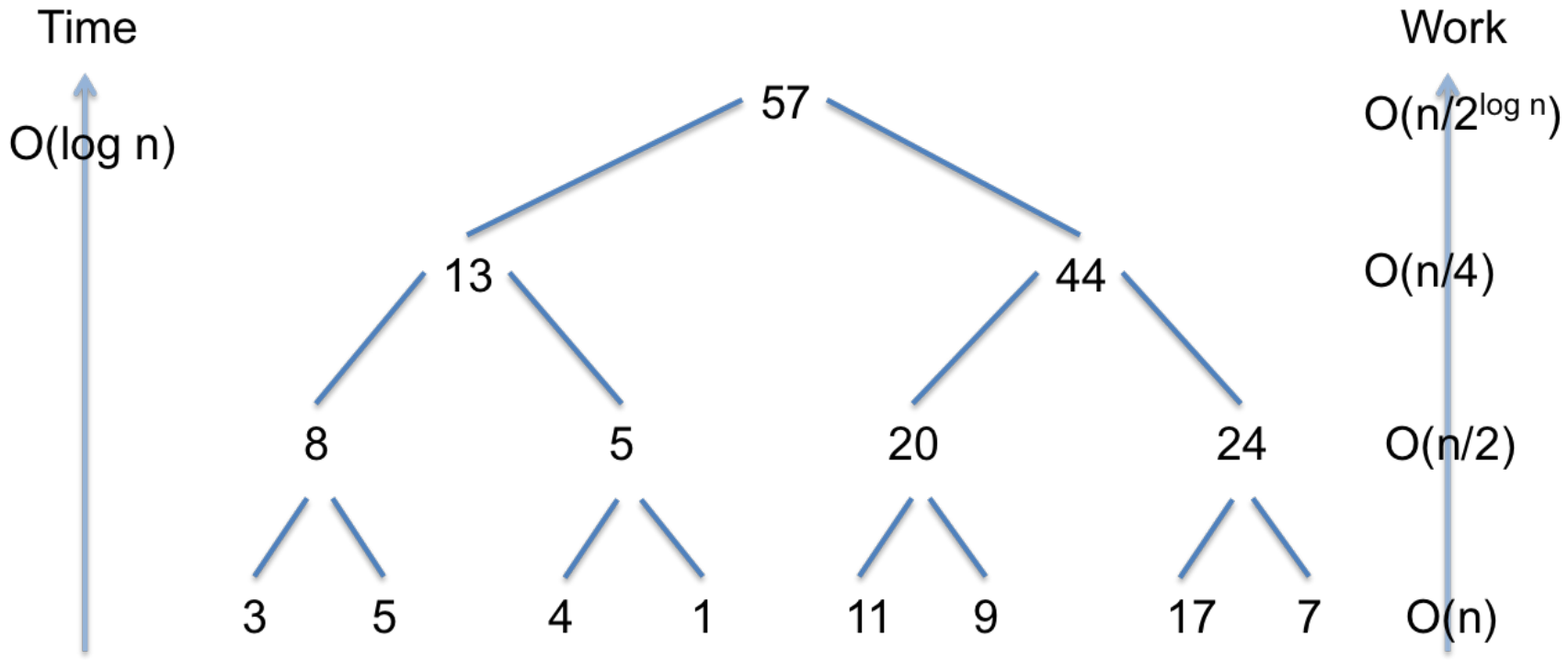
Summe auf der PRAM

ohne Abhängigkeit von p und mit impliziten Speicheroperationen

Function PRAM-Sum-simplified(A : array of size $n = 2^k$)
    for $1 \leq i \leq n$ **pardo**
        $B(i) = A(i)$
    for $h = 1$ **to** $\log n$ **do**
        for $1 \leq i \leq n/2^h$ **pardo**
            $B(i) = B(2i-1) + B(2i)$
    return $B(1)$

Summe auf der PRAM

Beispiel



- Laufzeit: $T(n) = O(\log n)$
- Arbeit: $W(n) = n + \sum_{j=1}^{\log n}(n/2^j) + 1 = O(n)$

Das Prinzip von Arbeit und Laufzeit (Work-Time-Principle)

Feiner Entwurf:

- Algorithmus A benötige $T(n)$ Laufzeit und $W(n)$ Arbeit.
- Fast immer:
  A lässt sich auf p PRAM-Prozessoren so abarbeiten, dass

$$\leq \left\lfloor \frac{W(n)}{p} \right\rfloor + T(n)$$

  parallele Schritte benötigt werden.

Das Prinzip von Arbeit und Laufzeit (Work-Time-Principle)

Feiner Entwurf:

- Algorithmus A benötige $T(n)$ Laufzeit und $W(n)$ Arbeit.
- Fast immer:
  A lässt sich auf p PRAM-Prozessoren so abarbeiten, dass
  $$\leq \left\lfloor \frac{W(n)}{p} \right\rfloor + T(n)$$
  parallele Schritte benötigt werden.

$\Longrightarrow$ Details des Scheduling uninteressant,
übernimmt Laufzeitumgebung

Optimalität

Sei $T^*(n)$ die sequentielle Zeitkomplexität eines Problems.

Simulation eines optimalen parallelen Algorithmus
mit Laufzeit $T(n)$ auf PRAM möglich in Zeit

$$T_p(n) = O(\frac{T^*(n)}{p} + T(n))$$

- Beschleunigung (engl.: *speedup*):
  $S_p := \frac{\text{Laufzeit des schnellsten sequentiellen Algorithmus}}{\text{Laufzeit des Algorithmus mit } p \text{ Prozessoren}}$
- Optimaler Speedup: $S_p(n) = \Theta(p)$ für $p = O\left(\frac{T^*(n)}{T(n)}\right)$

Optimalität

Sei $T^*(n)$ die sequentielle Zeitkomplexität eines Problems.

Simulation eines optimalen parallelen Algorithmus
mit Laufzeit $T(n)$ auf PRAM möglich in Zeit

$$T_p(n) = O(\frac{T^*(n)}{p} + T(n))$$

- Beschleunigung (engl.: *speedup*):
  $S_p := \frac{\text{Laufzeit des schnellsten sequentiellen Algorithmus}}{\text{Laufzeit des Algorithmus mit } p \text{ Prozessoren}}$
- Optimaler Speedup: $S_p(n) = \Theta(p)$ für $p = O\left(\frac{T^*(n)}{T(n)}\right)$

$\Rightarrow$ Beispiel Summe (klar: $T^*(n) = O(n)$):

- $T(n) = O(\log n)$
- $p = O(n/\log n)$
- Speedup $S_p(n) = O(n/\log n) = O(p)$ (optimal!)

Diskussion

- **Frage:** Warum kann der Speedup nie über p liegen?

Diskussion

- **Frage:** Warum kann der Speedup nie über p liegen?
- **Letzliches Entwurfsziel** für die PRAM:
  Schnellste parallele Laufzeit
  bei optimaler Arbeit = optimalem Speedup!

Präfixsumme

Entwurfsprinzip: Baumparadigma

Übung!