Effiziente externe Algorithmen durch Simulation paralleler Algorithmen

Olaf Bonorden

Dieser Text ist eine Zusammenfassung der Simulationen, die in den Papern „Efficient External Memory Algorithms by Simulating Coarse–Grained Parallel Algorithms“ ([DDH97]) und „Reducing I/O Complexity by Simulating Coarse Grained Parallel Algorithms“ ([DHMD99]) vorgestellt werden. Das erstgenannte Paper erweitert das BSP*–Modell für externe Algorithmen und stellt eine prohabilistische Simulation von BSP*–Algorithmen für Ein– und Multiprozessorcomputern vor. Im zweiten Paper wird eine deterministische Simulation von CGM–Algorithmen beschrieben.

Beide Simulationen führen zu effizienten externen Algorithmen.

1 Modelle

1.1 Das BSP*–Modell

Das BSP*–Modell ist eine in [BDM98] beschriebene Erweiterung des in [GV94] vorgestellten BSP–Modells um blockweise Kommunikation.

Im BSP–Modell besteht ein Parallelrechner aus unabhängigen Prozessoren mit lokalem Speicher, die vollständig durch ein Kommunikationsnetzwerk verbunden sind (Distributed Memory Machine, s. Abbildung 1). Es existiert im Gegensatz zum PRAM-Modell kein globaler Speicher. Die Topologie des Netzwerkes wird nicht untersucht, es wird lediglich angenommen, daß jeder Prozessor jedem anderen Nachrichten schicken kann.

Ein BSP–Algorithmus ist in *Supersteps* unterteilt, abwechselnd Berechnungs– und Kommunikations–Supersteps. In den Berechnungs–Supersteps wird in jedem Prozessor unabhängig

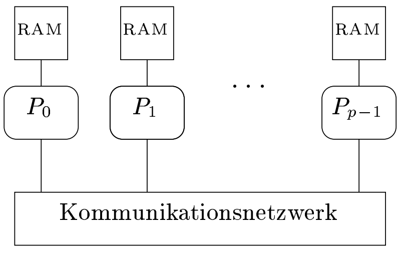


Abbildung 1: Distributed Memory Machine

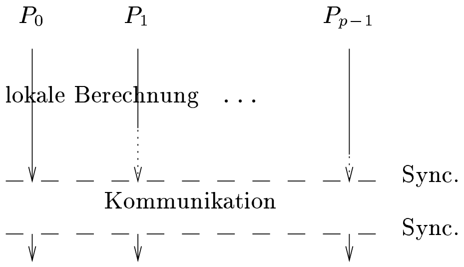


Abbildung 2: BSP Supersteps

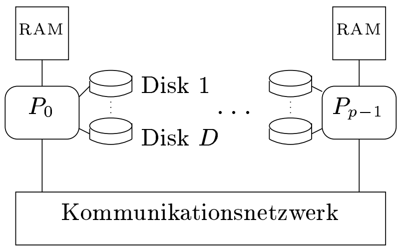


Abbildung 3: EM–BSP* Modell

lokal gerechnet, in den Kommunikations–Supersteps werden Nachrichten ausgetauscht. Zwischen den Supersteps synchronisieren sich alle Prozessoren mittels einer Barriere (s. Abbildung 2).

Um die Laufzeit eines Algorithmus zu bestimmen, beschreibt man jeden Computer mit folgenden Parametern:

p: Anzahl der Prozessoren

b: minimale Blockgröße für effiziente Kommunikation

g: Zeit pro übertragenem Block (in Prozessor–Befehlen)

L: Latenzzeit, minimale Kosten für Synchronisation

Mit diesen Parametern kann man die Laufzeit eines Superstep–Paares bestimmen: Sei w das Maximum der Anzahl ausgeführter Operationen und h das Maximum gesendeter plus empfangener Worte (Maxima jeweils über alle Prozessoren). Dann kostet ein Berechnungs– und ein Kommunikations–Superstep $w + g\frac{h}{b} + L$ Zeit. Im BSP*–Modell wird dabei jede Nachricht, die kleiner als b Worte groß ist, als b Worte gezählt. Hiermit soll modelliert werden, daß das Versenden kurzer Nachrichten auf den meisten reallen Parallelrechnern nicht die maximale Bandbreite erreicht.

1.2 Das EM–BSP* Modell

Das EM–BSP* Modell erweitert das BSP*–Modell um externen Speicher. Jeder der p Prozessoren besitzt lokal D eigene Festplatten, auf die parallel zugegriffen werden kann. Zur genauen Beschreibung werden weitere Parameter eingeführt:

D: Anzahl der Festplatten pro Prozessor

M: Größe des lokalen Hauptspeichers pro Prozessor

B: Blockgröße der Festplatten, Daten werden immer in ganzen Blöcken gelesen/geschrieben

G: Zeit in Prozessorbefehlen, die der Transfer eines Blockes von/auf eine Platte benötigt

Zusätzlich zu den im BSP–Modell beschriebenen Kosten pro Superstep addieren sich noch die I/O–Kosten $G \cdot \max_i\{w_{IO}^i\}$ mit w_{IO}^i Anzahl der I/O–Zugriffe in Prozessor i.

Definition 1 *Sei $\mathcal{A}$ der beste sequentielle Algorithmus für ein festes Problem, sei $T(\mathcal{A})$ die Laufzeit im worst case. Sei $c \geq 1$ konstant. Ein EM–BSP* Algorithmus $\mathcal{A}'$ heißt c–*optimal*, falls gilt:*

- *Das Verhältnis zwischen Berechnungszeit von $\mathcal{A}'$ und $T(\mathcal{A})/p$ ist $c + o(1)$.*
- *Das Verhältnis zwischen Kommunikationszeit von $\mathcal{A}'$ und $T(\mathcal{A})/p$ ist $o(1)$.*
- *Das Verhältnis zwischen I/O–Zeit von $\mathcal{A}'$ und $T(\mathcal{A})/p$ ist $o(1)$.*

2 Prohabilistische Simulation von BSP*–Algorithmen

2.1 Simulation mit einem Prozessor

Im folgenden sei ein BSP*–Algorithmus gegeben, der v Prozessoren und auf jedem Prozessor μ Speicherzellen (Kontext) benutzt. Dieser soll in einen sequentiellen EM–Algorithmus umgewandelt werden. Dazu wird jeder Kommunikations–Superstep in einen Sende– und einen Empfangsschritt zerlegt. Ein Berechnungs– und ein Kommunikations–Superstep werden nach folgendem Schema nacheinander für alle v Prozessoren simuliert (jeweils k gleichzeitig):

1. *Lesephase*: Lade die Kontexte und alle empfangenen Nachrichten für die k zu simulierenden Prozessoren von den Festplatten
2. *Berechnungsphase*: Simuliere die lokalen Berechnungen
3. *Schreibphase*: Schreibe die Kontexte und die gesendeten Nachrichten auf die Festplatten

Um zu einem effizienten EM–Algorithmus zu gelangen, muß in den Lese– und Schreibphasen gleichmäßig auf alle D Festplatten parallel zugegriffen werden. Die gelingt durch die Datenorganisation des im folgenden beschriebenen Algorithmus 1, SeqSuperstep.

Definition 2 *Eine gegebene Menge von Daten ist in* Blockformat :$\Leftrightarrow$ *die Daten sind aufgeteilt in Einheiten der Größe mindestens B.*

Definition 3 *Daten im Blockformat sind im* aufeinanderfolgendem Format, *falls*

1. *die Anzahl der Blöcke pro Festplatte ist $\leq 1+$minimale Anzahl Blöcke pro Platte, und*
2. *die Blöcke sind in aufeinanderfolgenden Spuren auf den Festplatten gespeichert.*

Dies bedeutet, daß Daten im aufeinanderfolgenden Format schnell von allen Festplatten parallel gelesen werden können.

Algorithmus 1 simuliert je einen Berechnungs– und einen Kommunikations–Superstep.

Details des Algorithmus 1, SeqSuperstep

Schritte 2 und 6: Da die Kontextgröße der einzelnen virtuellen Prozessoren bekannt und konstant ist, können die Kontexte deterministisch auf die Festplatten verteilt werden.

Algorithmus 1 SeqSuperstep

Eingabe: $\forall i \in \{0,\ldots,v-1\}$: Kontext von virtuellem Prozessor i, und $\forall i \in \{0,\ldots,\frac{v}{k}-1\}$ empfangene Nachrichten der Prozessoren $ik,\ldots,(i+1)k-1$, jeweils in aufeinanderfolgendem Format.

Ausgabe: Die geänderten Kontexte der Prozessoren und die gesendeten Nachrichten, aufgeteilt in $\frac{v}{k}$ Gruppen für jeweils k Prozessoren, jeweils in aufeinanderfolgendem Format.

```
1: for i = 0 to v/k − 1 do
2:    Lese Kontexte der virtuellen Prozessoren V_{ik}, ..., V_{(i+1)k−1} von den Festplatten in den Speicher
3:    Lese die von den k simulierten Prozessoren empfangenen Nachrichten in den Speicher
4:    Simuliere die lokalen Berechnungen der k virtuellen Prozessoren
5:    Schreibe die Blöcke der gesendeten Nachrichten auf die Festplatten (Details s. Text)
6:    Schreibe die geänderten Kontexte V_{ik}, ..., V_{(i+1)k−1} auf die Festplatten
7: end for
8: Reorganisiere die gespeicherten Nachrichten in aufeinanderfolgendem Format für jede Zielgruppe von k Prozessoren (mittels Algorithmus 2)
```

Schritt 5: Nach dem BSP* Modell haben Nachrichten die Länge $\geq B$. Jede Nachricht wird in einzelne Blöcke der Größe B zerteilt. Sei γ die maximale Kommunikationsgröße in Worten. Die Blöcke werden in $k\frac{\gamma}{B}/D$ Runden auf die Festplatten geschrieben. In jeder Runde werden also D Blöcke geschrieben. Diese werden mittels einer zufälligen Permutation auf die Festplatten verteilt. Auf jeder Festplatte werden die Blöcke in D Körbe nach der Zieladresse eingeteilt. In jeden der Körbe kommen also die Nachrichten für $\frac{v}{D}$ aufeinanderfolgende Prozessoren. Die Körbe werden mittels verketteter Listen verwaltet. Dieses Format heißt *verkettete Listen–Format* (s. Abbildung 4).

Schritt 8: Reorganisation der Daten mittels Algorithmus 2. Abbildung 5 zeigt ein Beispiel für die Speicherung der Nachrichten vor und nach Phase II. Die Zahlen innerhalb der Blöcke kennzeichnen die Zieladressen.

Satz 1 *Ein c–optimaler v–Prozessor BSP* Algorithmus mit Berechnungszeit $\beta+\lambda L$, Kommunikationszeit $g\alpha/b+\lambda L$ und lokaler Speichergröße μ kann auf einem 1–Prozessor EM–BSP* Computer mit Berechnungszeit $(1+o(1))v\beta$ und I/O–Zeit $O(G\lambda\frac{lv\mu}{DB})$ mit Wahrscheinlichkeit $1-e^{-\Omega(l\log l\cdot\log(M/B))}$ für eine Konstante $l\geq 1$, $\beta=\omega(\lambda\mu)$, $v\geq kD\log\frac{M}{B}$, $M=\Theta(k\mu)$, $b\geq B$ und k eine beliebige ganze Zahl, simuliert werden.*

Insbesondere gilt: Die Simulation eines c–optimalen BSP Algorithmus führt zu einem c–optimalen EM–BSP* Algorithmus.*

Beweis: s. [DDH97]

2.2 Simulation mit mehreren Prozessoren

Diese Simulation (s. Algorithmus 3, ParSuperstep), verläuft ähnlich dem sequentiellen Fall. Jeder Prozessor simuliert in einer Runde k der v Prozessoren, so daß ein Superstep nach $\frac{v}{pk}$ Runden beendet ist. Da hierbei Nachrichten zwischen virtuellen Prozessoren, die auf verschiedenen realen Prozessoren simuliert werden, Kommunikation erfordern, kann es zu unbalanciertem Kommunikationsverhalten kommen, falls die Nachrichten sofort verschickt werden.

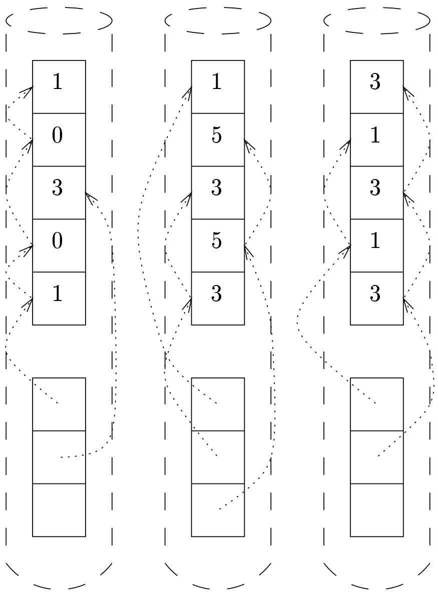


Abbildung 4: verkettete Listen–Format, $D = 3$, $v = 6$

Algorithmus 2 SimulateRouting

Eingabe: D Körbe mit Nachrichten im verketteten Listen–Format
Ausgabe: $\frac{v}{k}$ Gruppen von Nachrichten im aufeinanderfolgendem Format

{Phase I, kopiere Korb i auf Festplatte i}
for $j = 1$ to maximale Anzahl Blöcke pro Korb **do**
  for $d = 0$ to $D - 1$ **in parallel do**
    lese Block b_d von Korb d von Festplatte $(d + j) \bmod D$
    schreibe Block b_d auf Festplatte d auf die nächste freie Spur
  end for
end for
{Phase II, kopiere Körbe in aufeinanderfolgendes Format}
for $j = 0$ to $\frac{v\gamma}{DB}$ **do**
  for $d = 0$ to $D - 1$ **in parallel do**
    lese den j–ten Block von Festplatte d
    schreibe den Block auf Platte $(d + j) \bmod D$ auf Spur $i \left\lceil \frac{v\gamma}{D^2 B} \right\rceil + \left\lfloor \frac{j}{D} \right\rfloor$
  end for
end for

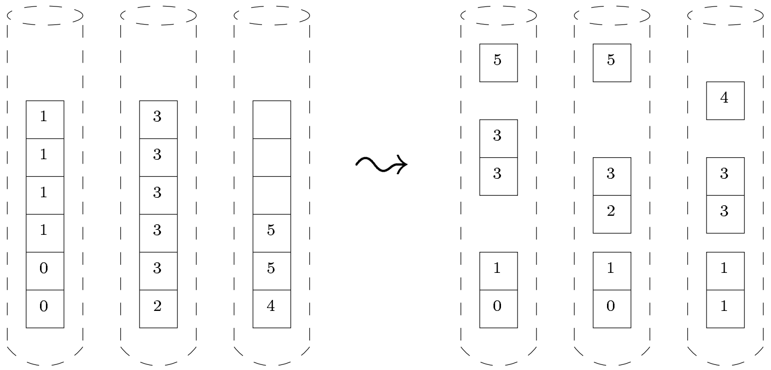


Abbildung 5: Reorganisation der Nachrichten durch Algorithmus 2, Phase II, die Zahlen sind die Zielprozessoren der Nachrichtenblöcke

---

Algorithmus 3 ParSuperstep

Eingabe: Jeder Prozessor hat $\frac{k\gamma}{B}$ Blöcke von jedem Stapel und die Kontexte der von ihm simulierten k Prozssoren, verteilt über die lokalen Festplatten
Ausgabe: geänderte Kontexte und Nachrichten für die nächste Runde

```
for j = 1 to v/(pk) do
  for i = 0 to p − 1 in parallel do
    Lese Blöcke für Stapel j und sende sie an die richtigen Zielprozessoren
    Simuliere die Berechnungen der virtuellen Prozessoren i(v/p), ..., (i + 1)(v/p) − 1.
    Teile alle gesendeten Nachrichten in Blöcke der Größe b und sende sie zu zufälligen
    Prozessoren
  end for
end for
```

Daher werden nach jeder Runde alle Nachrichten in Blöcke aufgeteilt an einen zufälligen Prozessor geschickt.

Nach der letzten Runde reorganisiert jeder Prozessor die empfangenen Nachrichten und schickt sie an den richtigen Zielprozessor. Die Nachrichten werden dabei in $\frac{v}{pk}$ Stapeln gespeichert, der j–te Stapel speichert die Nachrichten, die an Prozessoren gesendet werden, die in der j–ten Runde simuliert werden.

3 Deterministische Simulation

In diesem Abschnitt wird die Simulation von CGM–Algorithmen (aus [DHMD99]) beschrieben. Ein CGM–Algorithmus ist ein BSP–Algorithmus, in dem in jedem Superstep genau eine h–Relation mit $h = O\left(\frac{N}{V}\right)$ geroutet wird. Eine h–Relation routen bedeutet, daß jeder Prozessor $\leq h$ Worte sendet und $\leq h$ Worte empfängt.

3.1 Ein–Prozessor Simulation

Ein Problem bei der Simulation ist, daß zwar jeder Prozessor nur h Worte sendet bzw. empfängt, allerdings können die versandten Nachrichten unbalanciert sein. Der erste Algo-

Algorithmus 4 BalancedRouting

Eingabe: Jeder der v Prozessoren hat $\frac{\bar{n}}{v}$ Daten, eingeteilt in v Nachrichten beliebiger Länge, $msg_{i,j}$ sei die Nachricht von Prozessor i an j

Ausgabe: v Nachrichten in zwei Runden an Zielprozessoren geschickt

```
1:  {Superstep 1}
2:  for i = 0 to v − 1 in parallel do
3:    Reserviere v lokale Behälter
4:    for j = 0 to v-1 do
5:      for l = 0 to |msg_{i,j}| do
6:        Kopiere das l–te Wort von msg_{i,j} in Behälter (i + j + l) mod v
7:      end for
8:      for j = 0 to v − 1 do
          Sende Behälter j an Prozessor j
9:      end for
10:   end for
11: end for
12: {Superstep 2}
13: for j = 0 to v − 1 in parallel do
14:   Prozessor j reorganisiert die in Schritt 8 empfangenen Nachrichten in die Behälter seiner
      Zielprozessoren
15:   Prozessor j schickt Behälter k an Prozessor k für 0 ≤ k < v
16: end for
```

rithmus (Algorithmus 4, BalancedRouting) der Simulation ersetzt einen Kommunikations–Superstep durch zwei Runden, in denen die Kommunikation ausgeglichener ist.

Satz 2 *Seien v Prozessoren mit je $\frac{\bar{n}}{v}$ zu sendenen Daten gegeben, kein Prozessor sei Empfänger von mehr als $\bar{h}$ Daten. Die Kommunikation kann so in zwei Supersteps aufgeteilt werden, daß die Nachrichten im ersten Superstep eine Größe von $\frac{\bar{n}}{v^2} - \frac{v-1}{2}$ bis $\frac{\bar{n}}{v^2} + \frac{v-1}{2}$ und im zweiten Superstep eine Größe von $\frac{\bar{h}}{v} - \frac{v-1}{2}$ bis $\frac{\bar{h}}{v} + \frac{v-1}{2}$ haben.*

Beweis: s. [DHMD99]

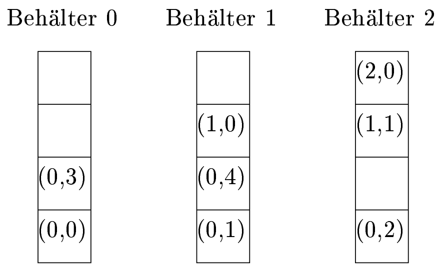


Abbildung 6: Aufteilung der Nachrichten in Behälter auf einem Prozessor durch Algorithmus 4, (i, j) bezeichnet Wort j der Nachricht an Prozessor i

Algorithmus 5 Deterministische Simulation eines Supersteps

Eingabe: Kontexte und empfangene Nachrichten in aufeinanderfolgendem Format
Ausgabe: geänderte Kontexte und gesendete Nachrichten in aufeinanderfolgendem Format

1: **for** $i = 0$ to $v - 1$ **do**
2: Lese Kontext des virtuellen Prozessors i in den Speicher
3: Lese empfangene Blöcke für Prozessor i von den Festplatte
4: Simuliere lokale Berechnungen von Prozessor i
5: Schreibe gesendete Nachrichten in gestaffeltem Format (s. Text)
6: Schreibe geänderten Kontext auf die Festplatten
7: **end for**

Details Schritt 5 von Algorithmus 5

Die Daten sollen so gespeichert werden, daß alle Nachrichten, die an einen Prozessor gesendet werden, im aufeinanderfolgenden Format gespeichert werden. Dies wird wie folgt erreicht: Sei b' die maximale Größe einer Nachricht in Blöcken. Die Nachrichten mit Ziel Prozessor j beginnen bei auf Festplatte $d_j = jb' \bmod D$. Mit $b' \bmod D \neq 0$ gilt also, daß die d_j gleichmäßig über die Festplatten verteilt sind. In jedem Zielbereich werden die Nachrichten im aufeinanderfolgenden Format mit Abstand b' Blöcke gespeichert, d.h. der q–te Block der Nachricht von Prozessor i nach Prozessor j steht in dem $(ib' + q)$-ten Block.

Um dieses Format zu erreichen, werden alle gesendeten Blöcke in eine Schlange geschrieben. Immer wenn in der Schlange ein Block eingefügt wird, der auf die gleiche Festplatte soll, wie ein anderer, schon in der Schlange enthaltener Block, wird die Schlange parallel auf die Festplatten geschrieben. Da die von Prozessor i gesendeten Nachrichten an j auf Platte $d_j + ib' \bmod D$ beginnen und die d_j gleichmäßig auf die Platten verteilt sind, können die von Prozessor i gesendeten Nachrichten parallel auf die D Festplatten geschrieben werden.

Die hier vorgestellte Art der Speicherung der Nachrichten benötigt den doppelten Platz, da Prozessor i die gesendeten Nachrichten abspeichern muß, bevor Prozessor $i + 1$ die Nachrichten des aktuellen Supersteps empfangen hat. Dies kann durch Abwechslung der Abfolge aufeinanderfolgendes Lesen, gestaffeltes Schreiben und gestaffeltes Lesen, aufeinanderfolgendem Schreiben vermieden werden.

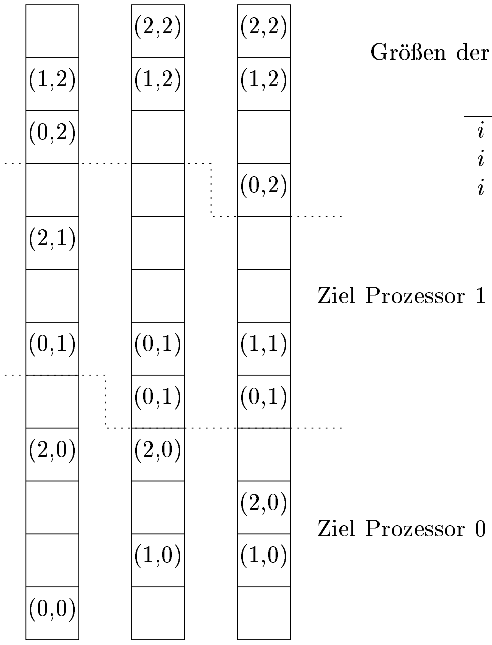
Satz 3 *Ein v–Prozessor–CGM–Algorithmus $\mathcal{A}$ mit λ Supersteps, lokaler Speichergröße μ, Laufzeit $\beta + g \cdot O(\frac{N}{v}) + \lambda L$ und Nachrichtengröße $\Theta(\frac{N}{v^2})$ kann auf einer Ein–Prozessor EM-CGM mit Zeit $v\beta + O(\lambda v \mu) + G\lambda \cdot O(\frac{v\mu}{DB})$ für $M = \Theta(\mu)$, $B = O(\frac{N}{v^2})$ und $N = \Omega(vDB)$ simuliert werden.*

Insbesondere führt die Simulation zu einem c–optimalen EM–CGM Algorithmus, falls $\mathcal{A}$ c–optimal, $\beta = \omega(\lambda\mu)$ und $G = DB \cdot o\left(\frac{\beta}{\lambda\mu}\right)$.

Beweis: s. [DHMD99]

3.2 Multi–Prozessor Simulation

Der parallele Simulationsalgorithmus 6 arbeitet ähnlich dem sequentiellen. Der einzige Unterschied ist, daß Nachrichten, die an einen virtuellen Prozessor gesendet werden, der auf einem anderen physikalischen Prozessor simuliert wird, als echte Nachrichten geschickt werden müssen.



Größen der Nachrichten in Blöcken ($|msg_{i,j}|$)

	$j=0$	$j=1$	$j=2$
$i=0$	1	4	2
$i=1$	2	1	3
$i=2$	3	1	2

Abbildung 7: Gestaffeltes Format der Nachrichten in Schritt 4, Algorithmus 5, (i,j) bezeichnet Block einer Nachricht von i an j, $b' = 4$

Algorithmus 6 Deterministische Simulation eines Supersteps, mehrere Prozessoren

---

Eingabe: Kontexte aller virtuellen Prozessoren und empfangene Nachrichten in aufeinanderfolgendem Format
Ausgabe: geänderte Kontexte und gesendete Nachrichten in aufeinanderfolgendem Format
1: **for** $i = 0$ to $p-1$ **in parallel do**
2:   **for** $j = 0$ to $\frac{v}{p} - 1$ **do**
3:     Lese Kontext des virtuellen Prozessors $\frac{v}{p}i + j$ in den Speicher
4:     Lese empfangene Blöcke für Prozessor $\frac{v}{p}i + j$ von den Festplatte
5:     Simuliere lokale Berechnungen von Prozessor $\frac{v}{p}i + j$
6:     Sende alle Nachrichten zu den Prozessoren, auf denen die virtuellen Zielprozessoren simuliert werden
7:     Schreibe in Zeile 6 empfangene Nachrichten in gestaffeltem Format auf die Festplatten, analog Algorithmus 5
8:     Schreibe geänderten Kontext auf die Festplatten
9:   **end for**
10: **end for**

Literatur

[BDM98] Bäumker, Armin, Wolfgang Dittrich and Friedhelm Meyer auf der Heide: *Truly Efficient Parallel Algorithms: 1-optimal Multisearch for an Extension of the BSP Model.* Theoretical Computer Science, 203, 1998.

[DDH97] Dehne, Frank, Wolfgang Dittrich and David Hutchinson: *Efficient External Memory Algorithms by Simulating Coarse-Grained Parallel Algorithms.* In *Annual ACM Symposium on Parallel Algorithms and Architecture.* ACM, 1997.

[DHMD99] Dehne, Frank, David Hutchinson, Anil Maheshwari and Wolgang Dittrich: *Reducing I/O Complexity by Simulating Coarse Grained Parallel Algorithms.* In *Annual ACM Symposium on Parallel Algorithms and Architecture.* ACM, 1999.

[GV94] Gerbessiotis, Alexandros V. and Leslie Valiant: *Direct Bulk-Synchronous Parallel Algorithms.* Journal of Parallel and Distributed Computing, 22:251–267, 1994.