Philipps-Universität Marburg

Informatik
Prof. Dr. R. Loogen

Sortieralgorithmen der Familia Radix-Sort

Seminararbeit
Seminar zur Parallelen Programmierung WS 2005-2006

Svetlana Tumanskaya

17.01.2006

Kurzzusammenfassung

Sortieralgorithmen insgesamt sind für die Informatik sehr wichtig. Die Benutzung des eines oder anderen Algorithmus ist von dem Typen und von der Menge zu sortierender Daten abhängig sowie von der Architektur des Rechners.

In dieser Arbeit wird der Sortieralgorithmus Radix-Sort diskutiert. Dieser Algorithmus wird öfter als andere Algorithmen auf den parallelen Rechnern benutzt, weil er einfacher zu realisieren ist. Die Hauptidee der Radix-Sort ist es die Behälter zu sortieren, um die Prozessoren als Behälter vorzustellen.

Am Anfang wird kurz über den Sequentiellet Radix-Sort berichtet, danach wird der parallele Radix-Sort dargestellt. Da der parallele Radix-Sort, det man von einer sequentiellen Version herleiten kann, ein Paar Probleme hat, werden noch weitere Sortieralgorithmen vorgestellt, die den parallelen Radix-Sort als Ausgangspunkt haben. Es wird über ***Load Balanced Radix Sort*** berichtet, danach kommen wir zu dem ***C^3-Radix-Sort*** und zeigen auch ***Sample Sort***. Die Auswahl dieser drei Sortieralgorithmen ist nicht Zufall, weil auf Grund dieser drei Verfahren der schnellste Sortieralgorithmus entwickelt wurde, der auf einem 64-Bit Speicher realisiert ist. Am Ende der Arbeit zeigen wir den Sortieralgorithmus ***PCS-Radix-Sort*** sowie welche Vorteile von allen drei Vorgängern in diesem Algorithmus realisiert wurden.

Inhaltverzeichnis

Kurzzusammenfassung 2
Inhaltverzeichnis 3
Sequentielle Radix-Sort 4
Load Balanced Radix Sort 4
Was ist Load Unbalanced Radix Sort oder Parallele Radix Sort? 5
Lösung der unbalancierter Problem 6
Vorteile 7
Nachteile 8
Communikation und Cache Conscious Radix Sort (C^3-Radix-Sort) 8
Nachteile 8
Sample Sort 9
Vorteile 10
Nachteile 10
Zusammenfassung: PCS-Radix-Sort 10
Beispiel der PCS-Radix-Sort 10
Algorithmus in Detail 11
Vergleich den Komponenten mit den vorgezeigten Algorithmen 13
Fazit 14
Literatur 15
Abbildungsverzeichnis: 15

Sequentielle Radix-Sort

Zuerst wollen wir den sequentiellen Algorithmus vorzeigen. Besonderheit dieser Algorithmus ist es, man zerlegt zuerst die zu sortierenden Daten in Teilschlüssel und danach sortiert man diese Teilschlüssel, dabei fängt man mit dem Schlüssel mit der niedrigste Priorität an.

Der Name "Radixsort" kommt von radix (dt. Basis), da das Verfahren auf der Darstellung g(x) der Werte der Elemente beruht. Wobei die Basis während eines Sortiervorganges immer konstant bleibt. Der Algorithmus wurde zum Sortieren von Lochkarten entwickelt.[1]

Als Beispiel zeigen wir die Sortierung von sechs 4-Stelligen Dezimalzahlen. (Siehe Abbildung 1a). Man zerlegt diese Zahlen in folgende Teilschlüssel: 1.te Ziffer, 2.te Ziffer, 3.te Ziffer, 4.te Ziffer. Die erste Ziffer haben dabei die höchste und die letzte Ziffer die niedrigste Priorität. Man beginnt die Zahlen bezüglich der niedrigsten Priorität zu sortieren. Die Zahlen werden dabei in der Ausgabe komplett sortiert. Das gleiche passiert mit den Buchstabenfolgen. (Abbildung 1b)

Abbildung 1 Beispiel sequentieller Radix-Sort a) Zahlen b) Buchstabenfolgen

a)

3841	3841	9528	9272	3641
9833	3641	9833	9528	3841
6787	9272	3841	3641	6787
9272	9833	3641	6787	9272
9528	6787	9272	9833	9528
3641	9528	6787	3841	9833

b)

hefe	gagA	baCh	bAch	Aber
bach	gehA	abEr	cAfe	Bach
gaga	hefE	heFe	gAga	Cafe
cafe	cafE	caFe	aBer	Gaga
geha	bacH	gaGa	hEfe	Geha
aber	abeR	geHa	gEha	Hefe

Es ist offensichtlich, dass man die Teilschlüssel mit einem einfachem Sortierungsalgorithmus sortieren muss. In der Literatur werden verschiedene Methoden angeboten. Zum Beispiel Counting-Sort[2] oder Bucketsort[3]. Die Komplexität des sequentiellen Radix-Sort ist von dem Algorithmus für Sortierung der Teilschlüssel abhängig. Zum Beispiel sei die d-stellige Zahlen zur Basis k gegeben, also die Wörter der große d aus einem Alphabet der Länge k. Dann braucht man zum Sortieren jedes Teilschlüssels mit Counting-Sort die Zeit O(n+k). Also insgesamt benötigt man die Zeit O(d*(n+k)), wobei n die Anzahl der zu sortierende Daten ist.

Load Balanced Radix Sort

1

2[4] Radix Sort und Couniting Sort. Robert Hilbrich http://www.informatik.hu-berlin.de/top/mitarbeiter/schmidt/lehre/progveri/radixsort.pdf

3 [5] Radix Sort http://www-lehre.inf.uos.de/~ainf/2004/skript/node65.html

[6] Radix Sort http://www.linux-related.de/coding/sort/sort_radix.htm

Dieser Sortieralgorithmus ermöglicht die gleichmäßige Verteilung der Schlüssel zwischen den Prozessoren. Die Idee des Algorithmus ist es zuerst alle Schlüssel zwischen Prozessoren zu verteilen und danach Verteilung der Schlüssel zwischen benachbarten Prozessoren so, dass am Ende jeder Prozessor gleich viele Schlüssel bekommt. Dieser Art der Radix-Sort war angeboten, um das Problem mit parallelem unbalancierte Radix-Sort zu lösen.

Was ist Load Unbalanced Radix Sort oder Parallele Radix Sort?

Um über den balancierten Radix-Sort zu sprechen, möchten wir zuerst der unbalancierte parallele Radix Sort vorstellen. Wenn man ein paar Schlüssel parallel sortieren möchte, sollte man schon über einem Sortieralgorithmus denken. Einen guten Lösung dafür ist es Radix Sort. Der größte Vorteil dieser Sortieralgorithmus ist es, man Sortiert nicht alle Daten, sondern eine Teil auf jedem Prozessor, was natürlich zu dem Zeitgewinn führt. Die Abbildung 2 zeigt ein Beispiel der Verteilung von 40 Schlüsseln zwischen 4 Prozessoren. P0,…,P3 sind dabei die Prozessoren und B0,…,B3 sind die festgelegte Segmenten (Behälter). Am Anfang werden alle Schlüssel so verteilt, dass jeder Prozessor genau ein Viertel aller Schlüssel bekommt. (Im Beispiel 10 Schlüssel). Weiter werden die Segmente so vertauscht, dass jedem Prozessor genau einen Segment zugehört. In diesem Fall wird das Ergebnis unbalanciert sein, da in diesem Fall wird die Anzahl der Schlüssel bei jedem Prozessor unterschiedlich sein.

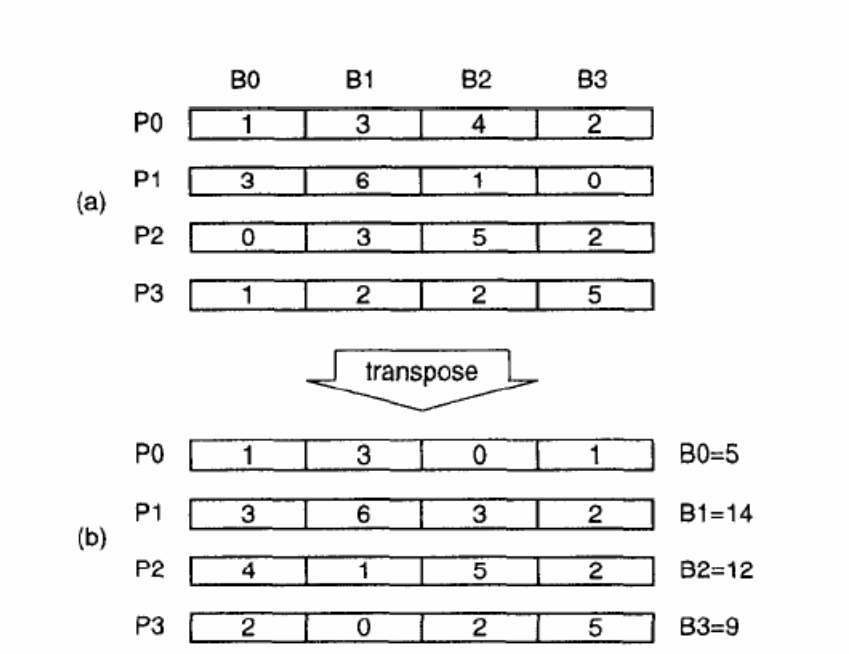

Abbildung 2 unbalancierte Radix-Sort[4]

Also unbalancierte Radix-Sort hat drei Problemen:

1. Radix Sort ausführt m globale Kommunikationsschritten, also einen für jede zu sortierende Zahl. Also im schlimmsten Fall wird jede zu sortierende Zahl

4 [2] s. 306

m Mal vertauscht. Für den Fall, dass jede Zahl 64 Bit hat, ist es besonders schlecht, weil man zu viele Kommunikationsschritte braucht.

2. Radix Sort ist unbalanciert für jeden Schlüssel, wenn alle Daten global schief sind. Es ist besonders für die 64-Bit-Zahlen aktuell, weil wahrscheinlich mehrere Zahlen ungleich lang sein werden.
3. Anzahl der Schlüssel für die Kommunikation in jedem Schritt wird sehr groß sein, besonders wenn die Daten global schief sind, also wenn eine lokale Segment einer Prozessor zu viele Schlüsseln hat und muss mit anderem Prozessor kommunizieren.

Lösung des unbalancierter Problems

Also alle Daten in die balancierte Form zu bringen ist sehr wichtig für die gute Arbeit der Algorithmus. Dafür werden die folgende Schritte benötigt:

Sei m – Anzahl der Daten, die sortiert werden;
P – Anzahl der Prozessoren.

Angenommen, sind alle Daten in unbalancierte Form zwischen den Prozessoren verteilt. Um die balancierte Form zu bekommen, müssen wir in jedem Prozessor *m/P* Daten bekommen. Also nehmen wir m/P Schlüsseln für den ersten Prozessor. Hier können drei Fälle passieren:

1. Fall *In erstem Prozessor sind genau m/P Schlüssel* – Das ist das leichteste Fall, da wir einfach in nächstem Prozessor weiter prüfen, ob er m/P Schlüsseln hat.
2. Fall *In erstem Prozessor sind mehr als m/P Schlüssel* – In diesem Fall lassen wir m/P Schlüsseln in diesem Prozessor und übernehmen die restlichen Schlüssel am Anfang der Daten in nächstem Prozessor.
3. Fall *In erstem Prozessor sind weniger als m/P Schlüssen* – In diesem Fall müssen wir die Fehlenden Schlüssel von dem nächsten Prozessor holen.

Ein Beispiel dieser Verfahren haben wir in Abbildung 3 gezeigt.

Es ist zu sehen, wie man die Schlüssel zwischen den Prozessoren verteilt, um wieder in jedem Prozessor die gleiche Anzahl von Schlüsseln zu bekommen. Also in Teil a) ist das Ergebnis der Abbildung 2 noch Mal vorgestellt. Anzahl der Schlüssel pro Segment ist unterschiedlich. Um es zu ändern, werden die Schlüsseln so verteilt, dass zu jedem Segment das gleiche Anzahl der Schlüsseln gehört, also nehmen wir im Beispiel alle Schlüsseln der erste Segment, wie sie in erstem Segment waren, und nehmen dazu noch die Schlüsseln der zweite Segment, um (im Beispiel) 10 Schlüsseln für den ersten Segment zu bekommen. Dafür müssen wir 6 Schlüsseln, die früher im Prozessor 1 Segment 1 waren, zwischen zwei Segmenten verteilen. 2 Schlüssel bekommt dann B0 und weitere 4 bleiben für B1 (Sehe Teil b) der Abbildung 3). Also nach dieser Verteilung bekommt jeder Prozessor von jedem Segment der gleichen Anzahl der Schlüssel.

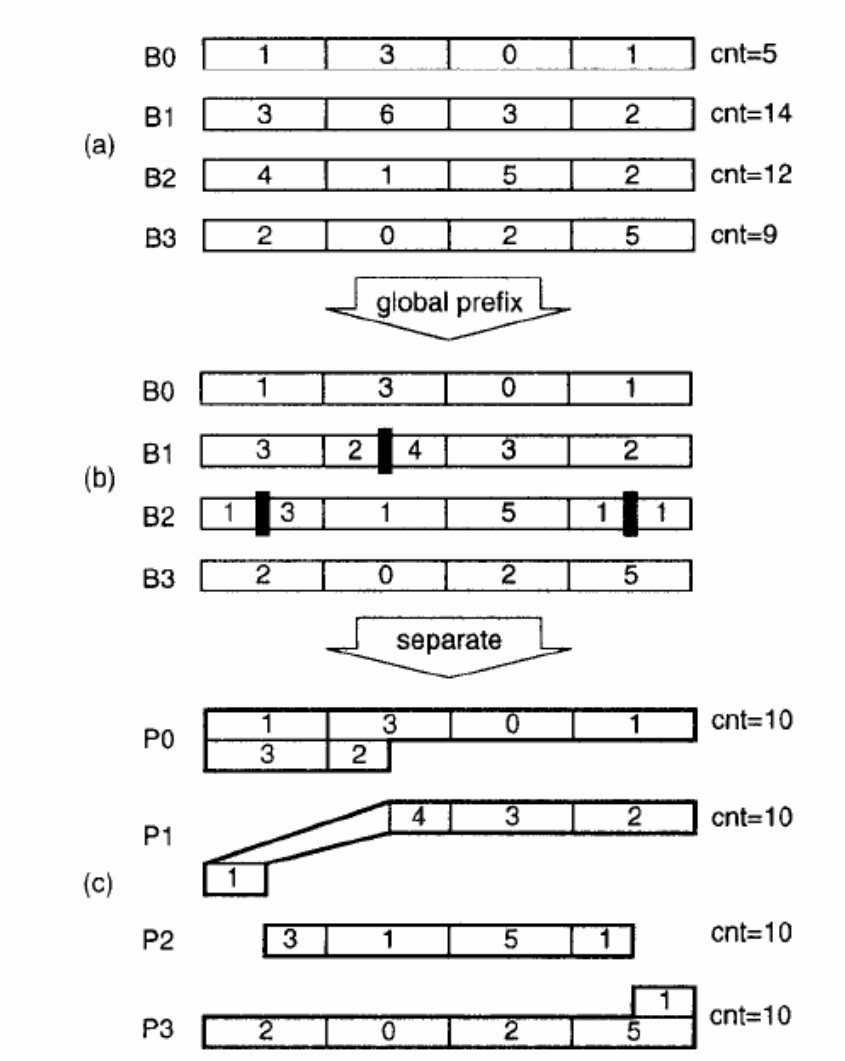

Abbildung 3 Load Balanced Radix Sort

Vorteile

Nach dem Ausführen der Algorithmus ist es ständig in jedem Prozessor die gleiche Anzahl der Schlüsseln, was die Kommunikation zwischen den Prozessoren erleichtert, weil man nicht mehr abwarten muss, wann der Prozessor, der die größte Anzahl der Schlüsseln bekommen hat, fertig wird. In der Abbildung 4 wird dieser Algorithmus noch Mal kurz vorgestellt.

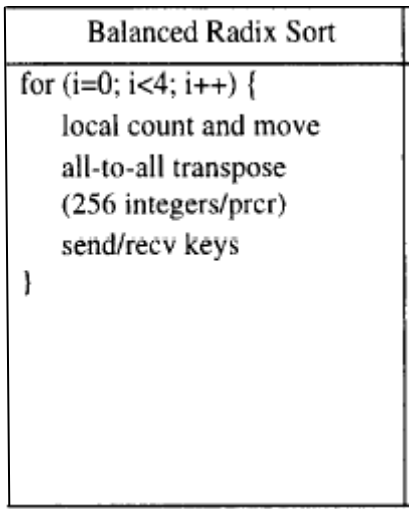

Abbildung 4 Schritten der Algorithmus Balanced Radix Sort

Nachteile

Obwohl dieser Algorithmus im Jahr 1996 der schnellste war, braucht er viele Kommunikationsschritten, einige von denen werden aber überflüssig sein. Als Folge ist die Sortierung langsamer, als es sein konnte.

*C*ommunication und *C*ache *C*onscious Radix Sort (C^3-Radix-Sort)

Dieser Algorithmus löst die oben genannten drei Probleme der Parallele Radix-Sort.

1. Um Anzahl der Kommunikationsschritten zu reduzieren, wird Reverse – Sorting benutzt, der besteht aus am Anfang b_{m-1} höchstwertigsten Bits pro Schlüssel, Zahl m-1. Dabei ist jeder Prozessor in $2^{b_{m-1}}$ lokale Segmenten verteilt. Diese Segmente sind sortiert, obwohl man die Schlüssel in dieser Segmenten noch sortieren muss. Danach, wenn alles balanciert bleibt, fängt die Kommunikationsschritt an, in dem man in jedem Prozessor die lokalen Segmente in globalen Segmenten umformt. Jeder von diesen globalen Segmenten wird lokal in einem Prozessor sortiert.
2. Wenn die Daten nach dem Schritt der Reverse – Sorting unbalanciert sind, wird der Reverse - Sorting so lange wiederholt, bis die Daten balanciert werden. Nur danach werden die globalen Segmente sortiert.
3. Wenn ein lokales Segment eines Prozessors zu viele Schlüssel hat, wird C^3-Radix-Sort dynamisch entscheiden, wie viele Schlüssel soll dieser Prozessor sortieren. Alle restlichen Schlüssel werden in dem nächsten Prozessor sortiert.

Nachteile

Die C^3- Radix - Sort hat das Problem, die von Reverse – Sorting stammt, dass die Anzahl der zu sortierende Zahlen mit der $\log_2$ der Anzahl der Eingängen der Cache beschränkt ist. Also dieser Sortieralgorithmus ist für die 64-Bit Zahlen uneffektiv.

Sample Sort

Einen typischen Sample Sort kann als Ausführung von acht Schritten vorgestellt werden:

1. Jeder Prozessor sortiert lokal n/P Schlüssel und fügt die bereits sortierten Schlüssel in P Behälter ein.
2. Jeder Prozessor schickt Behälter j der Größe 0…n/P zu dem Prozessor j. Die Größe jeder Behälter kann dabei von 0 bis n/P sein, häng von der Eigenschaften der Schlüssel ab. (*send*)
3. Jeder Prozessor empfängt P-1 Behälter von Schlüsseln und wählt davon s aus. Diese s Schlüsseln werden zu dem Prozessor 0 geschickt. (*receives*)
4. Prozessor 0 sammelt s Schlüssel von jedem von P Prozessoren und bekommt eine Reihe von s*P Schlüsseln. P Werten von dieser Reihe heißen ***Splitter*** (engl. *splitter*)
5. Prozessor 0 sendet P Splitter. (*broadcast*)
6. Jeder Prozessor empfängt P Splitter. Jede sortierte Unterreihe, die in Schritt 2 empfangen ist, wird basiert auf P Splitter umarrangiert. Als Ergebnis bekommt man dann neue Unterreihe. Jeder Prozessor also generiert insgesamt P^2 Unterreihen, wobei die Reihe von P Unterreihen entspricht jedem Splitter.
7. Jeder Prozessor schickt dem Prozessor j die Reihe von P Schlüsselreihen, entsprechend dem Splitter j.
8. Jeder Prozessor empfängt P^2 Unterreihen von Schlüsseln und fügt die in die Sortierte Liste hinzu.

Also alle diese Schritte sind in der Abbildung 5 noch Mal gezeigt.

Sample Sort
local sort
send/recv keys
local sample selection
1-to-all send/recv samples (n/P^2 samples/processor)
splitter selection on a single processor
broadcasting splitters
send/recv keys
local merge

Abbildung 5 Schritten der Algorithmus Sample Sort

Vorteile

Dieser Sortieralgorithmus kann auf den Schlüsseln beliebiger Größe benutzt werden. Dieser Algorithmus **ist** oft als Sortieralgorithmus für das Sortieren von Plattenspeichern benutzt. Deswegen ist dieser Algorithmus sehr beliebt.

Nachteile

Die Nachteile dieser Algorithmus sind aber Unmöglichkeit unnötiges Sortieren zu vermeiden was von der Eigenschaft der Unberücksichtigung der empfangenen Daten folgt.

Also dieser Sortieralgorithmus ist immer noch nicht die beste Lösung für Sortierung.

Zusammenfassung: PCS-Radix-Sort

Die Ziele der Algorithmus ist es Anzahl der Kommunikationsschritte bis zu einem zu reduzieren, bei dem sollen die Daten in der balancierte Form sein und unnötiges Sortieren zu vermeiden. Zuerst zeigen wir am Beispiel allgemeine Variante der Sortierung, und danach zeigen wir auch jeder Schritt der Algorithmus in Detail.

Beispiel der PCS-Radix-Sort

In der Abbildung 6 werden fünf Schritten der PCS-Radix-Sort für den Prozessor 0 vorgestellt. Insgesamt sind 3 Prozessoren gegeben.

1. Jeder Prozessor i verteilt seine lokale Schlüssel in s lokale Behälter: $B^i_0,\ldots,B^i_j,\ldots,B^i_{s-1}$. (Sehe Abbildung 6.1) Dabei ist s größer als die Anzahl der Prozessoren und nicht unbedingt durch 2 Teilbar ist. ***s ist der erste von drei Parametern des Algorithmus***. Behälter in jedem Prozessor werden gleich sortiert, aber die Schlüssel werden erst im Schritt 5 des Algorithmus sortiert. Die Behälter mit dem Index j in jedem Prozessor werden dann im gleichen Bereich liegen. (Die Vereinigung der Behälter mit dem Index j, also $\bigcup_{k=0}^{P-1} B_j^k$, schafft die globale Behälter).

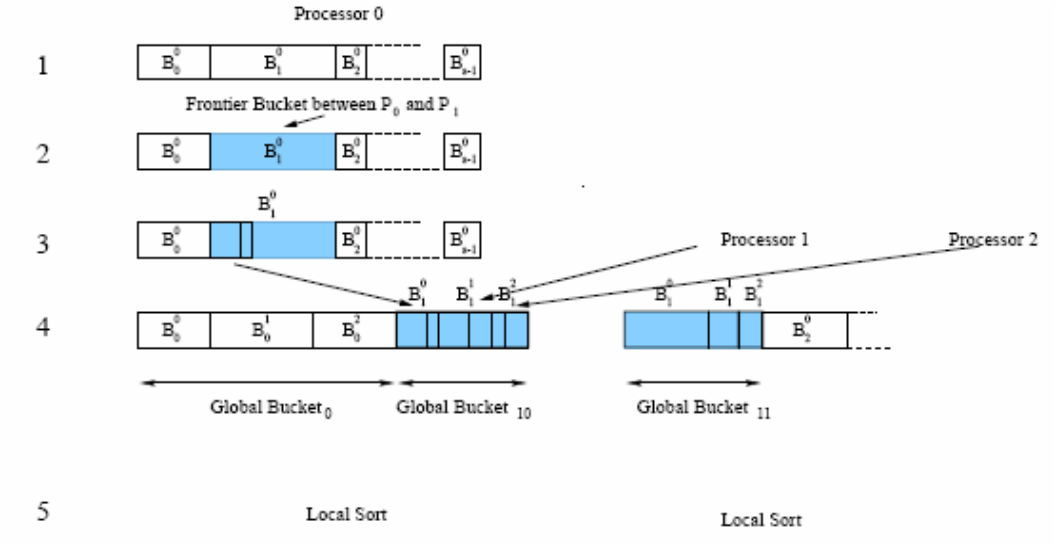

Abbildung 6 Beispiel der PCS-Radix-Sort[5]

2. Übertragung der Anzahl der Schlüssel pro Behälter ist so organisiert, so dass alle Prozessoren wissen die Anzahl der Schlüssel pro globale Behälter. Mit dieser Information werden alle Prozessoren entscheiden, wie viele globale Behälter muss jeder Prozessor nehmen, um die Anzahl der Schlüssel in jedem Prozessor ungefähr N/P wäre. Einige dieser Behälter werden dann in zwei benachbarten Prozessoren verteilt. Allerdings nicht mehr als in zwei. Im Beispiel (Abbildung 6.2) ist es zu sehen, dass der lokale Behälter B_1^0 auf der Grenze zwischen den Prozessoren 0 und 1 liegt. Die Schlüssel werden aber in diesem Schritt noch nicht verteilt.
3. In diesem Schritt werden alle auf der Grenze liegende Behälter zwischen den benachbarten Prozessoren auf s' Unterbehälter verteilt. ***s' ist der zweite von drei Parametern des Algorithmus***. Wieder ist s' nicht unbedingt durch 2 Teilbar. Im Abbildung 6.3 ist es zu sehen, dass der Behälter B_1^0 ist auf s' = 3 Unterbehälter verteilt. Diese feine Kernhaftigkeit ermöglicht die gute Balancierung.
4. In diesem Schritt wird die Kommunikation all-to-all zwischen den Prozessoren realisiert. Die Verteilung wird ähnlich wie bei C^3-Radix-Sort entschieden. Nach dieser Entscheidung sendet jeder Prozessor die Behälter, die anderem Prozessor gehören, zu diesen Prozessoren. Gleichzeitig werden die globalen Behälter gebildet. Zu einem globalen Behälter gehören auch die auf der Grenze liegende Unterbehälter. Also alle Behälter, die zu einem Prozessor zugeordnet werden, gehören zu seinem globalen Behälter.
5. Um die Sortierung zu beenden, werden die Schlüssel in jedem globalem Behälter unabhängig von den anderen Prozessoren sortiert.

Algorithmus in Detail

Jetzt schauen wir alle 5 Schritte in Detail:

[5] [1] s. 118

1. Am Anfang wird der Algorithmus alle Schlüssel zwischen s Behältern verteilen. Das geschäht ähnlich der Verteilungsphase der Sample Sort, aber mit folgende Modifikation:
    a) Zufällige Auswahl von q Schlüssel von den lokalen Daten und Übertragung deren. ***q ist dabei der dritte von drei Parametern des Algorithmus***. Aber q ist trotzdem begrenzt, um unnötiges Kommunikation in nächstem Schritt zu vermeiden.
    b) Weiter wird der Vektor von P*q Schlüsseln hergestellt, indem werden (P-1)*q Schlüsseln von den anderen Prozessoren empfangen und q Schlüsseln von sich selbst.
    c) Jetzt werden diese Vektoren sortiert.
    d) Danach wird der Splitter von s-1 Schlüsseln hergestellt. Das wird gleichmäßig von den Vektoren gewählt, gestartet mit (P*q/s).tem Schlüssel.
    e) Nun wird Counting - Algorithmus benutzt, um diese N/P Schlüssel in s lokale Behälter zu verteilen. Dabei wird der Schlüssel k zu dem Behälter B_j zugeteilt, wenn Splitt[j] <= k < Split[j+1] ist. Die benötige Anzahl der Schlüsseln ist im Parameter s gespeichert.
2. Jeder Prozessor sendet Index s, der er in vorherigem Schritt bekommen hat. P*s Zähler werden benötigt, um die Anzahl der Schlüssel zu berechnen, die werden in globale Behälter benötigt. Wenn s wird so gewählt, dass s > P ist, dann werden sicher nicht mehr als zwei Prozessoren die Grenzwertenbehälter benutzen. Hier wird festgelegt, dass Behälter j als Grenzbehälter behandelt wird, wenn einen von seiner Schlüsseln ist auf Position N/P gespeichert.
3. Jeder Prozessor teilt jede von seine lokale Behälter mit dem Parameter s' auf Unterbehälter (benutze Counting - Splitt - Algorithmus entsprechend 1e). Für den Fall, dass wurde s'-1 Schlüssel in der Vector verteilt, benutze die Teil 1b.
4. Wieder werden die Indexen s' gesendet. Das ist der Kommunikationsschritt. Hier werden alle Behälter zwischen den Prozessoren endgültig verteilt. Alle Unterbehälter werden zu den entsprechenden Prozessoren gesendet. Am Ende dieser Schritt bekommt jeder Prozessor einen oder mehrere globale Behälter.
5. Nach der Kommunikation wird jeder globale Behälter in seinem Prozessor lokal sortiert mit dem Sortieralgorithmus Sequential Counting Split Radix Sort (SCS – Radix - Sort). Bei diesem Schritt sparen wir die unnötiges sortieren der schon sortierte Bits. Das wird wie folgt erreicht:
    a) Man berechnet die Anzahl der größten Bits, die wurden schon sortiert für jeden Behälter. Diese Zahl ist gleich der Anzahl der aufeinander folgenden größten Bits in Splitt[i] und Splitt[i+1].
    b) Sortiere die Schlüssel in jedem globalen Behälter, die man noch sortieren muss, mit dem Sortieralgorithmus SCS-Radix-Sort. Dieser Sortieralgorithmus ermöglicht die Berücksichtigung der Größe von Cache, was die totale Ausführungszeit verbessert.

Vergleich den Komponenten mit den vorgezeigten Algorithmen

Also wie man noch im Algorithmus sieht, wird es gleich im Schritt 1 im PCS der Verfahren aus dem Sample Sort benutzt. Dieser Algorithmus wird aber nicht eins – zu – eins übernommen, sondern modifiziert, in dem wird die Anzahl der empfangener Daten berücksichtigt, sowie die unnötige Sortieren wird angewendet.

Obwohl der C^3- Radix – Sort der Algorithmus Reverse-Sorting benutzt, wird dieser Verfahren nicht in PCS-Radix-Sort übernommen. Aber die Idee und der Algorithmus der Kommunikation in einem Schritt sind genau von diesem Algorithmus genommen, um den PCS-Radix-Sort zu verbessern.

Der Load Balanciert Radix Sort ist aber sehr wichtig für PCS-Radix-Sort, weil der Algorithmus kann nur dann effektiv sein, wenn er balanciert ist. Obwohl dieser Verfahren zu vieler Kommunikation zwischen den Prozessoren braucht, ist dieser Algorithmus im Vergleich mit allen anderen Algorithmen viel schneller, deswegen wird er auch für den neuen verbesserten Algorithmus PCS-Radix-Sort genommen. Aber der Verfahren wird wiederum geändert, um diese unnötige Kommunikation zu vermeiden. Der Hauptunterschied ist es also: Der PCS-Radix – Sort verteilt die Grenzbehälter nicht in zwei Teilen, wie beim LB, sondern verfeinert sie noch weiter und senden mehrere Teil-Behälter zu einem der zwei Prozessoren.

In der Abbildung 7 ist es zu sehen, wie daraus die Beschleunigung des Verfahrens sich ändert.

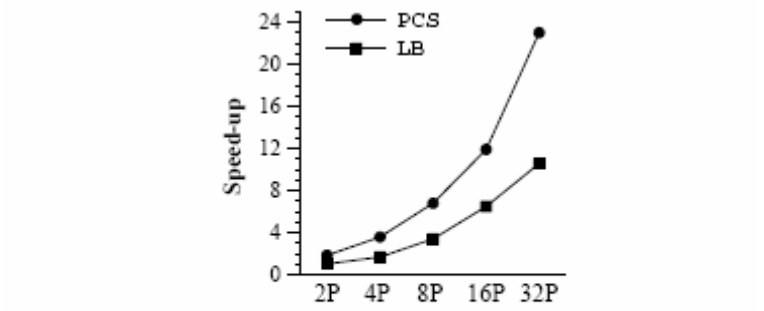

Figure 6: Speed-up comparison between PCS-Radix sort (PCS) and Load Balanced Radix sort(LB).

Abbildung 7 Vergleich von PCS- und LB- Beschleunigkeit[6]

Also alle starken Stellen dieser Algorithmen sind für das neue Verfahren benutzt und alle schwachen Stellen wurden in neuem Verfahren korrigiert. Deswegen können wir feststellen, dass der Algorithmus PCS-Radix-Sort der beste von allen parallelen Algorithmen ist.

[6] [1] s. 122

Fazit

Es gibt viele parallele Sortieralgorithmen, deswegen man hat die Auswahl und kann für jeden Zweck der beste Algorithmus wählen. Aber die Anzahl der Daten wird immer größer, also man braucht einen guten Algorithmus, der bei der große Menge der zu sortierenden Daten am schnellsten ist und leicht zu realisieren ist. Diese Lösung heute kann PCS-Radix-Sort sein. Dass dieser Algorithmus heutzutage der schnellste ist, ist in [1] bewiesen. Außerdem ist er nicht schwer zu realisieren, weil der Sortieralgorithmus PCS-Radix-Sort mit der Hilfe der MPI-Bibliothek realisieren wird, was die Größe der Code erniedrigt. Der PCS-Radix-Sort hat sich von guten Seiten gezeigt und ist für den Praxis Empfehlungswert.

Literatur

1. Fast Parallel In-Memory 64-bit Sorting
   Daniel Jimenez-Gonzalez, Juan J. Navarro, Josep-L.Laribba-Pey Computer Architecture Dept. Universitat Politecnica de Catalunya Jordi Girona1-3, Campus Nord-UPC, Modul D6, E-08034 Barcelona s. 114-122

2. Load Balanced Parallel Radix Sort
   Andrew Sohn Computer Information Science Dept. New Jersey Institute of Technologie Newark, NJ 07102-1982 Yuetsu Kodama Parallel Computer Architecture Laboratory Electrotechnical Laboratory 1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan s. 305 – 312

3. Radix Sort und Couniting Sort. Robert Hilbrich, Humboldt-Univeristät zu Berlin http://www.informatik.hu-berlin.de/top/mitarbeiter/schmidt/lehre/progveri/radixsort.pdf

4. **Algorithmen** Vorlesung im WS 2004/2005 Oliver Vornberger Ralf Kunze Olaf Müller Institut für Informatik Fachbereich Mathematik/Informatik Universität Osnabrück Skript als PDF-Version Radix Sort http://www-lehre.inf.uos.de/~ainf/2004/skript/node65.html

5. **Zeitoptimale Sortierverfahren Von Matthias Jauernig und Tobias Hammerschmidt Kapitel 5 Radix Sort** http://www.linux-related.de/coding/sort/sort_radix.htm

6. Peter Weigel, **Sortieralgorithmen**
   http://www.sortieralgorithmen.de/radixsort/index.html

Abbildungsverzeichnis:

Abbildung 1 Beispiel sequentieller Radix-Sort a) Zahlen b) Buchstabenfolgen___ 4
Abbildung 2 unbalancierte Radix-Sort _____ 5
Abbildung 3 Load Balanced Radix Sort _____ 7
Abbildung 4 Schritten der Algorithmus Balanced Radix Sort _____ 8
Abbildung 5 Schritten der Algorithmus Sample Sort _____ 9
Abbildung 6 Beispiel der PCS-Radix-Sort _____ 11
Abbildung 7 Vergleich von PCS- und LB- Beschleunigkeit _____ 13