Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät Informatik, Mathematik und Naturwissenschaften

Projektdokumentation
Parallel Sorting by Regular Sampling

Analyse der Laufzeit eines parallelisierten Sortieralgorithmus mit Hilfe von MPI

Prüfungsprojekt im Fach Cluster Computing
Prof. Dr.-Ing. Schneider

B. Sc. Michael Schmidt Mat.Nr: 52676
B. Sc. Christof Pieloth Mat.Nr: 52674
29. Januar 2010

Inhaltsverzeichnis

1 Einleitung **2**

1.1 Aufgabenstellung . . . 2
1.2 Definition PSRS . . . 2
1.3 Die Testumgebung . . . 2
1.4 Einschränkungen . . . 2

2 Implementierung von PSRS **3**

2.1 Phase 1 . . . 3
2.2 Phase 2 . . . 3
2.3 Phase 3 . . . 3
2.4 Phase 4 . . . 4

3 Auswertung der Testergebisse **5**

3.1 Zeit des Sortiervorganges . . . 5
3.2 Overhead und Sortierzeit gegenüber der Gesamtlaufzeit . . . 6
3.3 Speedup . . . 7

4 Zusammenfassung **9**

1 Einleitung

1.1 Aufgabenstellung

Setzen Sie den vorgegebenen Algorithmus in ein C-Programm unter Einsatz von MPI um. Dabei sollen folgende Punkte Berücksichtigung finden:

- die Zahlen sind von einem Prozessor zu erzeugen und an die beteiligten Prozessoren zu verteilen
- am Ende soll ein Prozessor das sortierte Feld zu Kontrollzwecken ausgeben können
- das Programm soll mit unterschiedlich großen Datensätzen getestet werden, z.B. 20000, 40000, 80000 Zahlen
- das Programm soll mit unterschiedlich vielen Prozessoren getestet werden.

Analysieren Sie das Laufzeitverhalten Ihres Programms, wobei insbesondere Aussagen zum Speedup von Interesse sind. Des weiteren sollen Aussagen darüber getroffen werden, welchen zeitlichen Anteil das anfängliche sequenzielle Sortieren an der Gesamtlaufzeit des Programms hat und wie groß der Kommunikationsoverhead ist.

1.2 Definition PSRS

Bei Parallel Sorting by Regular Sampling (PSRS) handelt es sich um einen parallelen Sortieralgorithmus der einen verträglichen Kommunikationsaufwand und gute Balance-Eigenschaften aufweist. Der Algorithmus ist für diverse MIMD-Architekturen geeignet.

Eine genaue Definition der einzelnen Phasen ist in Kapitel 2 aufgezeigt.

1.3 Die Testumgebung

Als Testumgebung für dieses Projekt diente der Glass-House Cluster (alle Knoten befinden sich im gleichen Raum) im Zusebau, Z423 der HTWK Leipzig. Dieser Cluster besteht aus 24 Knoten. Es handelt sich um einen homogenen Cluster (Hard und Software auf allen Geräten gleich).

1.4 Einschränkungen

Da der Cluster auf den Workstations der HTWK läuft, ist kein dedizierter Zugriff auf das System möglich. Dadurch ergeben sich recht große Schwankungen bei den Laufzeitmessungen. Um mögliche Messschwankungen gering zu halten, wurden alle Messungen fünf mal nacheinander ausgeführt und anschließend daraus der Durchschnitt berechnet.

Des weiteren wurden alle Tests mehrfach ausgeführt, um eine konkrete Aussage über das Verhalten des Programms treffen zu können. Durch diese Mehrfachausführung konnten Fehler in den Messreihen, welche durch die Benutzung der einzelnen Knoten als Workstation aufgetreten sind, eingeschränkt werden. Zudem wurden größere Messungen während der Nacht oder am Wochenende ausgeführt, da zu diesen Zeiten davon auszugehen ist, dass die einzelnen Knoten nicht als Workstations benutzt werden.

2 Implementierung von PSRS

2.1 Phase 1

In der Phase 1 des Algorithmus erhält jeder Knoten ein Teilfeld des gesamten Feldes. Dieses Teilfeld wird lokal in jedem Knoten sortiert, um anschließend die Elemente an den Positionen

$$1, w + 1, 2 * w + 1, ..., (N - 1 * w + 1) \quad \text{mit} \quad w = n/N^2$$

zu selektieren. Die selektierten Elemente jedes Knoten bilden die reguläre Auswahl des gesamten Feldes.

In der vorliegenden Implementierung überprüft der Root-Knoten die Größe des Feldes. Falls die Anzahl der Knoten kein ganzzahliger Teiler der Feldgröße ist, wird das Feld so erweitert, dass die Feldgröße ein Vielfaches der Knotenanzahl ist. Anschließend verteilt der Root-Knoten die Teilfelder durch

```
MPI_Scatter(array, nLocal, MPI_INT, aLocal, nLocal, MPI_INT, ROOT,
  MPI_COMM_WORLD);
```

Nachdem jeder Knoten sein Teilfeld erhalten hat, wird dieses mit Mergesort sortiert. Danach wird die reguläre Auswahl berechnet und in einem Feld gespeichert.

2.2 Phase 2

Ein festgelegter Knoten sammelt in der Phase 2 die reguläre Auswahl der Knoten ein und sortiert diese. Nun werden $N - 1$ Pivots an den Position

$$N + 1, 2 * N + t, ..., (N - 1) * N + t \quad \text{mit} \quad t = N/t \quad \text{(ganzzahliger Teil)}$$

aus der sortierten regulären Auswahl selektiert. Jeder Knoten erhält eine Kopie der Pivots und zeugt N Blöcke aus seiner lokalen sortierten Liste.

In der praktischen Umsetzung sammelt der Root-Knoten die reguläre Auswahl mit

```
MPI_Gather(aRegLocal, N, MPI_INT, aRegular, N, MPI_INT, ROOT,
  MPI_COMM_WORLD);
```

ein. Anschließend wird die Auswahl mit Mergesort sortiert und die Pivots selektiert. Nun verteilt der Root-Knoten durch

```
MPI_Bcast(aPivots, N - 1, MPI_INT, ROOT, MPI_COMM_WORLD);
```

die selektierten Pivots.

2.3 Phase 3

Der Knoten i erhält jeweils den Block i von allen Knoten ($i = 1, ..., N$).

Da die Blockgrößen variablen sind, müssen in der Implementierung zwei MPI-Funktionen genutzt werden. Zuerst berechnet jeder Knoten die Größe seiner Blöcke und teilt diese den anderen Knoten mit.

```
calculateCount(aLocal, nLocal, aPivots, N - 1, countsSnd, N);
MPI_Alltoall(countsSnd, 1, MPI_INT, countsRecv, 1, MPI_INT,
  MPI_COMM_WORLD);
```

Nachdem jeder Knoten die zugehörigen Blockgrößen gesendet bzw. empfangen hat, können anhand dessen die benötigte Feldgröße und die Displacements errechnet werden. Erst jetzt können die eigentlichen Blöcke ausgetauscht werden.

```
calculateDisplsFromCount(countsSnd, N, displsSnd, N);
calculateDisplsFromCount(countsRecv, N, displsRecv, N);
// ...
MPI_Alltoallv(aLocal, countsSnd, displsSnd, MPI_INT, aBlocks,
  countsRecv, displsRecv, MPI_INT, MPI_COMM_WORLD);
```

2.4 Phase 4

In der Phase 4 mischen die Knoten ihre N Blöcke parallel. Durch Zusammensetzen der erzeugten Listen entsteht das sortierte Feld.

Mit Hilfe der Methode

```
mergeBlocks(aBlocks, displsRecv, countsRecv, N);
```

werden die N vorsortierten Blöcke gemischt. Damit das sortierte Feld im Root-Knoten zusammengesetzt werden kann, müssen wiederholt zuerst die Größen der zusammengesetzten Teilfelder versendet werden. Anschließend können daraus die Displacements berechnet werden und um nun die Teilfelder zu dem sortierten Feld zusammensetzen zu können.

```
MPI_Gather(&nBlocks, 1, MPI_INT, countsRecv, 1, MPI_INT,
  ROOT, MPI_COMM_WORLD);
// ...
calculateDisplsFromCount(countsRecv, N, displsRecv, N);
MPI_Gatherv(aBlocks, nBlocks, MPI_INT, array, countsRecv,
  displsRecv, MPI_INT, ROOT, MPI_COMM_WORLD);
```

3 Auswertung der Testergebisse

Bei den Testergebnissen handelt es sich um Zeitmessungen, welche an verschiedenen Punkten des Sortieralgorithmus vorgenommen wurden. Gemessen wurde die Gesamtzeit, um daraus Aussagen über den Speedup und andere Aspekte des Algorithmus treffen zu können. Da in einem Cluster Aufgaben und Ergebnisse ausgetauscht werden müssen, beeinflusst diese Kommunikation zwischen den Knoten die Laufzeit. Indem die benötigte Zeit der MPI-Methoden gemessen wird, kann dieser Anteil bestimmt werden. Die dritte Zeitmessung diente zur Ermittlung der durchschnittlichen Sortierzeit auf den einzelnen Knoten.

Als Werkzeug zum Messen der Zeit diente die Funktion `MPI_Wtime` . Bei einem Aufruf der Funktion wird ein Timestamp zurückgeliefert. Um eine Zeitspanne zu ermitteln, wurde jeweils am Anfang und am Ende der Prozedur ein solcher Timestamp erzeugt und die Differenz als Zeit ausgegeben.

3.1 Zeit des Sortiervorganges

Dieser Abschnitt betrachtet das Verhältnis der Gesamtlaufzeit zur Anzahl der zu sortierenden Elemente. Eine theoretische Vorüberlegung zu diesem Verhältnis ergibt einen Anstieg der Gesamtlaufzeit bei steigender Anzahl der zu sortierenden Elemente und gleichbleibender Knotenanzahl. Dieses Ereignis ist in Abbildung 1 ist zu erkennen. Die Gesamtlaufzeit steigt bei steigender Anzahl von Elementen nahezu linear. Der genaue Anstieg ist logarithmisch, da der zugrundeliegende Sortieralgorithmus, Mergesort, eine asymptotische Laufzeit von $\mathcal{O}(nlogn)$ und PSRS von $\mathcal{O}(\frac{n}{N}logn)$ hat. Zudem wird anhand dieser Abbildung deutlich, dass dieser Algorithmus parallelisiert ist. Dies ist daran zu erkennen, dass mit steigender Anzahl von Knoten die Gesamtlaufzeit des Algorithmus bei gleichbleibender Elementanzahl abnimmt.

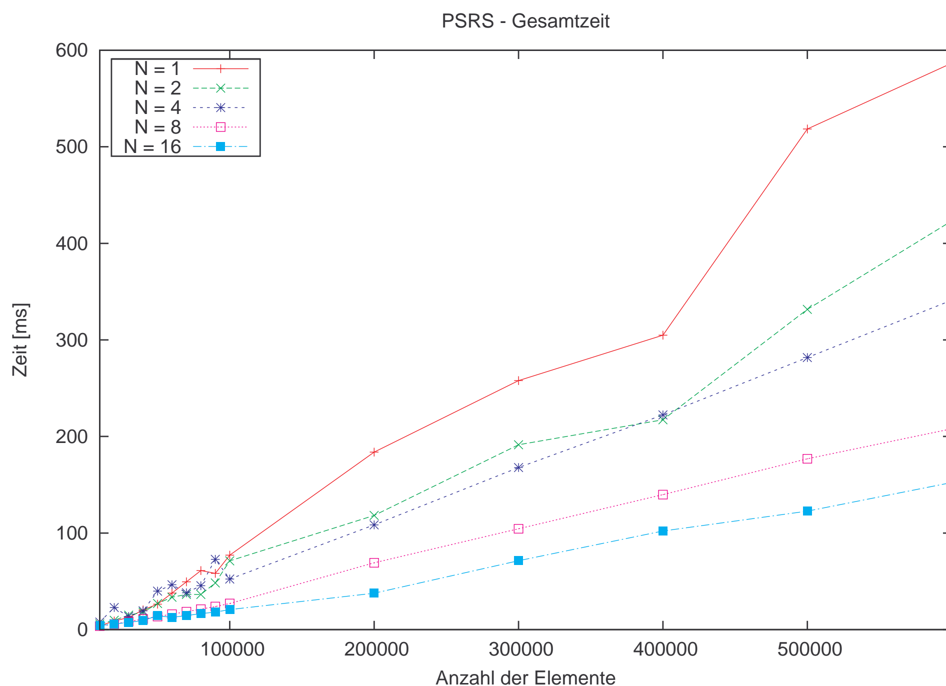

Abb. 1: Gesamtlaufzeit

3.2 Overhead und Sortierzeit gegenüber der Gesamtlaufzeit

Abbildung 2 zeigt die Gesamtlaufzeit, die durchschnittliche Sortierzeit und den Overhead des PSRS bei einer konstanten Anzahl von acht Knoten. Der Graph der durchschnittlichen Sortierzeit hat den geringsten Anstieg. Ab 100 000 Elementen ist zu erkennen, dass der Overhead mit steigender Anzahl von Elementen einen deutlich größeren Anstieg als die sequenzielle Sortierzeit hat. Die Gesamtlaufzeit hat den größten Anstieg, da sich diese unter anderem aus der Summe von Overhead und der Sortierzeit berechnet. Aus weiteren Messungen ist zudem ersichtlich, dass der Overhead mit steigender Anzahl der Knoten zunimmt.

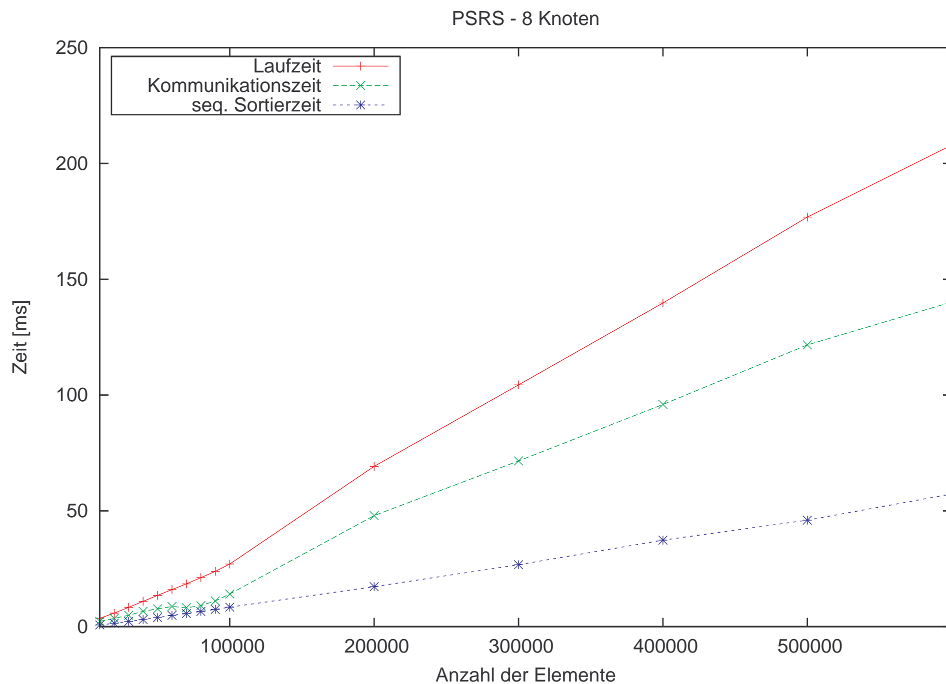

Abb. 2: Gesamtlaufzeit, Overhead und Sortierzeit bei 8 Knoten

Aus diesem Graphen lässt sich das Verhältnis der durchschnittlichen Sortierzeit der einzelnen Knoten zur Gesamtlaufzeit des Programmes ablesen. In Abbildung 3 ist dieses Verhältnis genauer dargestellt. Die durchschnittlichen Sortierzeit der einzelnen Knoten ist zur Gesamtlaufzeit gesehen konstant. Dieses Phänomen ergibt sich daraus, dass die Differenz aus der Gesamtlaufzeit und der Sortierzeit in Abhängigkeit von der Anzahl der Elemente und der Anzahl der aktiven Knoten linear ansteigt.

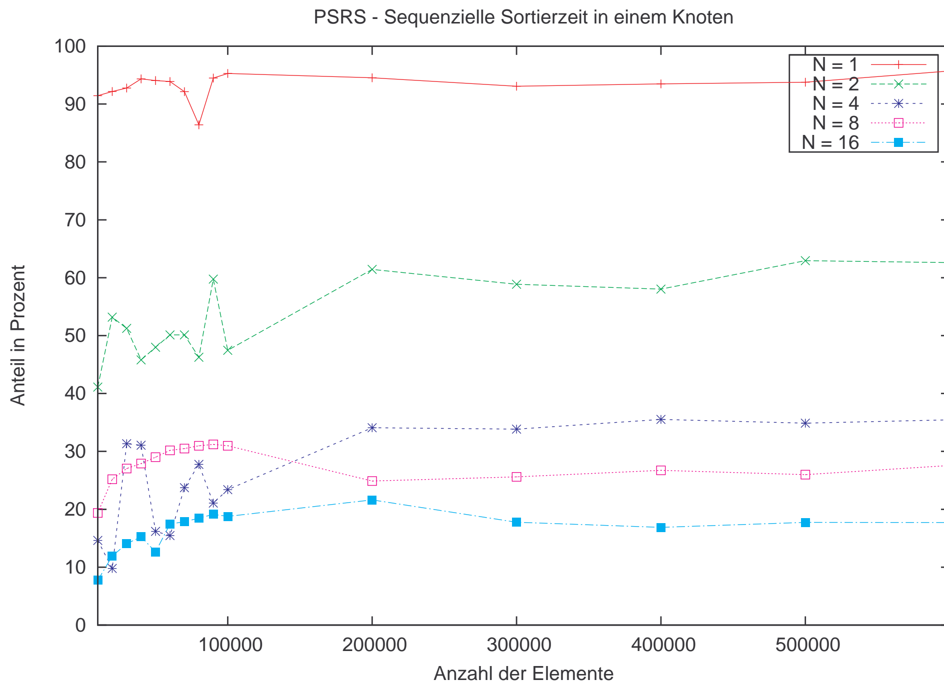

Abb. 3: Sortierzeit in Abhängigkeit der Gesamtlaufzeit

3.3 Speedup

Der Speedup gibt an um welchen Faktor die Ausführung eines Programms beschleunigt werden kann, wenn dieses anstatt auf einem Einprozessorsystem, auf einem Mehrprozessorsystem ausgeführt wird. Im vorliegenden Fall berechnet sich dieser aus dem Quotienten der Gesamtlaufzeit des Algorithmus auf einem Knoten und der Gesamtlaufzeit auf mehreren Knoten.

$$S(N) = \frac{T(1)}{T(N)} \text{ mit } \begin{array}{l} \text{S(N)} = \text{Speedup auf N Knoten} \\ \text{T(1)} = \text{Laufzeit auf einem Knoten} \\ \text{T(N)} = \text{Laufzeit auf N Knoten} \end{array}$$

Weiterhin gilt für den Speedup $1 \leq S(N) \leq N$. Die Zeitkomplexität von PSRS beträgt für den Fall $n >= N3$:

$$O[(n/N)logn]$$

Theoretisch kann der maximale Speedup für Parallel Sorting by Regular Sampling mit Hilfe dieser asymptotischen Laufzeit gezeigt werden:

$$S(N) = \frac{\frac{n}{1}\log n}{\frac{n}{N}\log n} = \frac{n\log n}{\frac{n}{N}\log n} = \frac{N * n\log n}{n\log n} = N$$

Der theoretisch maximale Speedup von N ist in der Praxis jedoch kaum zu erreichen, da dieser durch den sequentiellen Anteil des Problems beschränkt ist (Amdahlsches Gesetz). Des weiteren wird der Speedup in einem Cluster stark durch die anfallende Kommunikation beeinflusst. Laut Abschätzung nach Giloi ist ein linearer Anstieg der Speedup-Funktion der günstigste Fall, da so die Ausführungszeit unter Zunahme weiterer Knoten immer merklich

verkürzt werden kann. In Abbildung 4 ist der Speedup für drei verschiedene Problemgrößen dargestellt. Erkennbar ist, dass der Speedup mit steigender Anzahl der Knoten ansteigt und somit die Laufzeit verkürzt wird. Da der Speedup z.B. von vier auf acht Knoten jedoch nicht verdoppelt wird und dieser Faktor mit steigender Knotenanzahl sinkt, hat dieser Algorithmus eine logarithmische Speedup-Funktion. Im Gegensatz zum günstigsten Fall laut Giloi nimmt der Geschwindigkeitsgewinn pro weiteren Knoten ab und somit sinkt die Effizienz. Des weiteren ist erkennbar, dass eine Abhängigkeit zwischen Speedup und der Anzahl der Elemente besteht. Da die Übertragungsrate im Bezug zur eigentlichen Rechenleistung sehr gering ist, beansprucht das Verteilen des Problems und das Sammeln der Ergebnisse bei kleinerer Anzahl der Elemente viel mehr Zeit als die Berechnung auf den einzelnen Knoten.

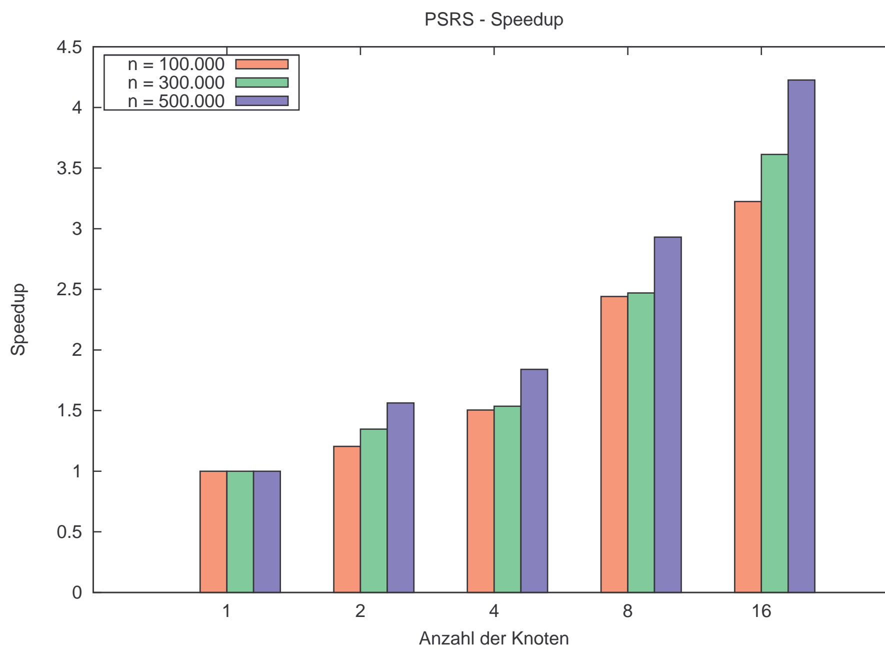

Abb. 4: Speedup mit verschiedenen Problemgrößen

4 Zusammenfassung

Das parallele Sortieren durch Parallel Sorting by Regular Sampling (PSRS) ist für Problemgrößen, bei denen $n >= N^3$ gilt, sehr gut geeignet. In diesem Projekt wurden Messungen mit einer variierenden Anzahl von Prozessoren (Knoten) und verschiedenen Größen der zu sortierenden Zahlen durchgeführt. Ideale Messwerte konnten dabei durch den Einfluß verschiedener Faktoren nicht erreicht werden. Einen großen Einfluss dabei hatte zum Beispiel, daß das Clustersystem während Durch die Durchführung mehrerer Messreihen, welche hauptsächlich in den Nachtstunden beziehungsweise am Wochenende durchgeführt wurden, konnte dennoch ein repräsentatives Ergebnis erziehlt werden. Dieses spiegelt den typischen Speedup von PSRS unter Verwendung eines Sortieralgorithmus mit einer Laufzeit, welche sich in $\mathcal{O}(nlogn)$ befindet, wieder. Faktoren, welche Einfluss auf einen suboptimale Speedup haben sind unter anderem:

- Eine Parallelisierung ist nur bedingt möglich (Amdahlsches Gesetz)
- Kommunikationsoverhead

Dennoch konnte mit Hilfe dieses Projektes gezeigt werden, daß eine Parallelisierung durchaus sinnvoll sein kann. Nicht zuletzt deswegen sind die schnellsten Supercomputer der Welt Clustersysteme.