PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen
J.Berthold

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

16.Mai 2007

5. Übung zur Vorlesung "Parallele Algorithmen", Sommer 07

Abgabe: 31.Mai 2007 vor der Vorlesung

Die Bearbeitungszeit für dieses Übungsblatt ist auf zwei Wochen verlängert, es sind 18 Punkte erreichbar. Am Pfingstmontag findet kein Tutorium statt.

Aufgaben

5.1 **Aufwand von Hyperquicksort** — 4 Punkte

Bekanntlich hängt der Aufwand eines sequenziellen Quicksort-Verfahrens entscheidend von der Datenaufteilung (bzw. Wahl des Pivots) ab. Auch das parallele Verfahren Hyperquicksort hat diese Eigenschaft.

Beschreiben Sie einen *best case* und einen *worst case* für die Datenaufteilung und den resultierenden Aufwand des Verfahrens.

5.2 **Hyperquicksort in MPI** — 7 Punkte

Das in der Vorlesung vorgestellte Verfahren *Hyperquicksort* verwendet die Hypercube-Struktur für sukzessive Broadcasts und zur logischen Strukturierung der parallelen Prozesse. MPI bietet eine effiziente Broadcast-Implementierung `MPI_Bcast` (als kollektive Operation in einem Kommunikator). Eine Aufteilung der Prozesse analog zur Teilung eines Hypercubes in zwei Teil-Hypercubes kann mit Kommunikatoren erreicht werden:

```
MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm);
```

> Die kollektive Operation `MPI_Comm_split` erzeugt neue Kommunikatoren und ordnet ihnen die Prozesse im übergebenen Kommunikator zu. Prozesse werden mittels ihrer Färbung `color` zugeordnet, alle Prozesse gleicher Farbe werden Mitglieder des gleichen Kommunikators. Die Angabe `key` dient zur Zuweisung der Ränge im neuen Kommunikator; bei Konflikten (also bei gleichem `key`) entscheidet der bisherige Rang.

Implementieren Sie Hyperquicksort mit den genannten kollektiven Operationen.

Weitere Hinweise:

Die zu sortierenden Daten sollen bereits auf die Prozessoren verteilt sein. Im Programm können die Daten lokal von den einzelnen MPI-Prozessen erzeugt werden.
Implementieren Sie mehrere Varianten: Zufallswerte im Bereich [0..1024], zufällige Null-Eins-Folgen und bereits sortierte Daten. Zufällige Daten erzeugen Sie in C mit `rand()` nach Initialisierung des Zufallsgenerators mit `srand(unsigned seed)`.

Um Speicher zu sparen, sollten Sie bei Datenversand zunächst die Größe der Daten, danach die Daten selbst senden. Auf diese Weise kann der empfangende Prozess nach Empfang der Größe exakt den nötigen Speicher reservieren. Verwenden Sie die Funktion `realloc`, um bereits reservierten Speicher ggf. zu vergrößern, oder auch zurückzugeben.

Bitte wenden!

5.3 **Zeitmessungen mit MPI** 4 Punkte

Die Funktion `MPI_Wtime` liefert einen `double`-Wert zurück, welcher als Zeit in Sekunden zu interpretieren ist. Durch Aufruf an mehreren Stellen und Vergleich der Werte kann während der Ausführung eines parallelen Programms die Zeit für bestimmte Schritte gemessen werden.

(a) Versehen Sie Ihr Programm[1] aus Aufgabe 5.2 mit entsprechenden Aufrufen, um die Zeit zu messen, die das Programm zur eigentlichen Sortierung benötigt.

(b) Messen Sie diese Zeit mit verschiedenen Eingaben und Prozessorzahlen $2^k | k \in \{1, 2..\}$. Analysieren und bewerten Sie die Messdaten mit der Karp-Flatt-Metrik.

Beachten Sie die unterschiedliche Leistung der für Ihre Messung verwendeten Maschinen. Welchen Einfluss hat die Verwendung von Dual-Core- und Hyperthreaded-Rechnern?

5.4 **Listenmischung im PSRS-Algorithmus** 3 Punkte

In Phase 4 des PSRS-Algorithmus muss jeder Prozess p sortierte Teillisten mischen, deren Größe in etwa $\frac{n}{p^2}$ beträgt. Die Gesamtanzahl der zu mischenden Elemente ist ca. $\frac{n}{p}$ und höchstens $\frac{2n}{p}$.

Beschreiben Sie ein Verfahren, das den Mischvorgang in $\Theta(\frac{n}{p} \log p)$ durchführt.

[1] Falls Sie Aufgabe 2 nicht bearbeiten, verwenden Sie statt dessen das auf der Webseite bereitgestellte Programm OET-Sort. Es implementiert das Verfahren Odd-Even-Transposition Sort.