

Programmierübungen 2

Sommersemester 2008

10. Übungsblatt

23. Juni 2008

Abgabe bis Montag, 30. Juni 14:00 Uhr

Auf den Vortragsfolien der Programmierübungen oder im Skript zur Einführung in die Informatik abgedruckte Quelltexte können verwendet werden, müssen aber der Programmierrichtlinie entsprechend formatiert und kommentiert werden.

Bitte bearbeiten Sie die Übungsaufgaben in Kleingruppen mit bis zu 3 Teilnehmern. Alle Teilnehmer einer Kleingruppe müssen in der selben Übungsgruppe eingetragen sein. Ihre Abgabe muss eine Datei `contributions.txt` enthalten, in der aufgelistet ist welche Anteile der Bearbeitung von jedem einzelnen Teilnehmer erstellt wurden. Jeder Teilnehmer muss die Funktionsweise aller Teile der Bearbeitung erläutern können. Beachten Sie die Programmierrichtlinie und kommentieren Sie Ihren Quelltext. Dokumentieren Sie unbedingt Ihre Lösungsidee in den Quelltext-Kommentaren.

<http://www.iste.uni-stuttgart.de/ps/Lehre/SS2008/inf-prokurs>
Bitte beachten Sie, dass zum Scheinerwerb unter anderem auf den letzten 4 Aufgabenblättern 50 % der Punkte erreicht werden müssen. Das letzte Aufgabenblatt wird Nummer 12 sein.

Zu diesem Aufgabenblatt findet am Dienstag, 24.06. um 8 Uhr eine Vortragsübung statt.

Aufgabe 10.1: Mischen

(6 Punkte)

Zu diesem Aufgabenblatt erhalten Sie wiederum das generische Paket `Sorting`, das bereits von dem letzten Aufgabenblatt bekannt ist. Fügen Sie dem Paket ein Unterprogramm hinzu, das zwei sortierte Arrays mischt und die resultierende Elementfolge in ein drittes Array speichert. Stellen Sie sicher, dass auch Slices (zusammenhängende Ausschnitte von Arrays) als Parameter für alle Parameter übergeben werden können.

Hinweis Damit Slices korrekt behandelt werden können, darf Ihre Implementierung keine Annahmen über den Indexbereich (also über den Wert von `A'First` oder `A'Last`, falls `A` das Array ist) verwenden.

Aufgabe 10.2: Paralleles Sortieren

(14 Punkte)

In dieser Aufgabe soll eine im folgenden genau beschriebene Strategie implementiert werden, um ein Feld parallel auf mehreren Prozessoren zu sortieren. Moderne Betriebssysteme erlauben es einem Programm weitere Abläufe (Threads) parallel zu dem Hauptprogramm zu starten. Alle Threads dürfen dabei auf den gemeinsamen Hauptspeicher zugreifen (shared memory). Das Betriebssystem benötigt recht viel Laufzeit dafür, um einen neuen Thread (in Ada „Task“ genannt) zu starten und zu verwalten, so dass sich dieser Aufwand nur lohnt, falls jeder Thread eine genügend große Teilaufgabe des gesamten Problems zu lösen hat.

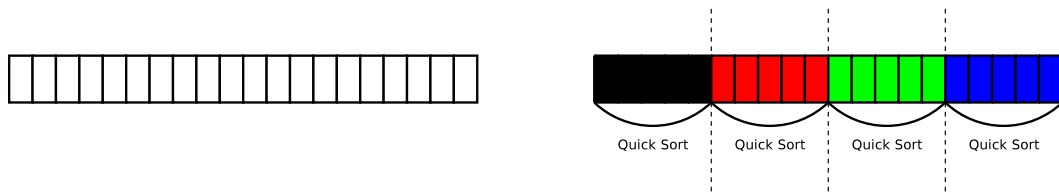


Abbildung 1: Paralleles sortieren, links der unsortierte Array, rechts Anwendung mehrerer Quick Sort-Tasks. Verschiedene Farben repräsentieren unabhängig voneinander sortierte Slices

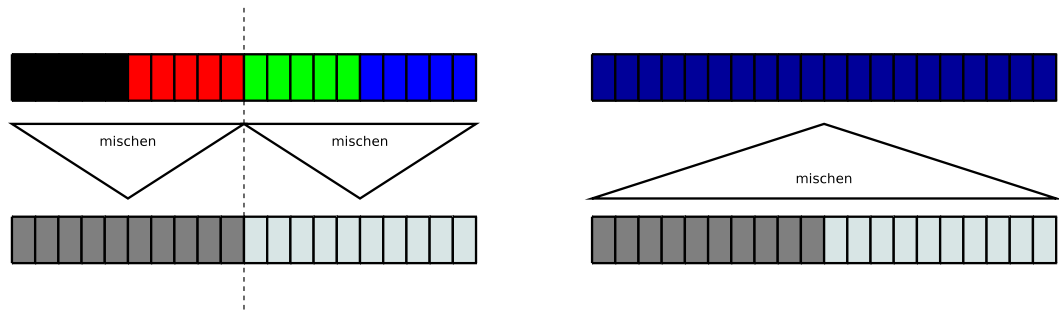


Abbildung 2: Zweimaliges Mischen, verschiedene Farben repräsentieren unabhängig voneinander sortierte Slices

Zu diesem Aufgabenblatt erhalten Sie auf der Webseite das bereits bekannte generische Paket `Sorting`, ein neues Paket `Parallel_Sorting` sowie ein Hauptprogramm `Test_Parallel_Sorting`. Das Hauptprogramm akzeptiert als ersten Kommandozeilen-Parameter die Länge eines Arrays, als zweiten Parameter eine Anzahl Tasks. Derzeit verwendet es diese Anzahl an Tasks, um den Array mit Zufallszahlen zu belegen. Zum Schluss führt das Programm eine Prüfung durch, ob der Array sortiert ist und beendet sich.

Fügen Sie Quelltext ein, der die Sortierung des Arrays mit `Number_Of_Tasks` (Variable im Hauptprogramm) vielen parallelen Tasks durchführt. Implementieren Sie diesen Algorithmus:

- Der Index-Bereich des Array wird in gleich große Teile unterteilt,
- Es werden Tasks erzeugt, von denen jeder Task jeweils einen dieser Slices separat sortiert, vgl. Abbildung 1 (hierzu kann `Float_Sorting.Quick_Sort` verwendet werden),
- Es wird abgewartet bis alle Tasks beendet sind.
- Nun werden die sortierten Slices gemischt mit der Implementierung aus Aufgabe 1. Das Mischen sollte ebenfalls parallel durchgeführt werden. Dazu wird ein zweites Array benötigt. In mehreren Phasen kann dann zwischen diesen Array „hin- und hergemischt“ werden, wie in Abbildung 2 skizziert. Die einfachste Implementierung wartet jeweils ab, bis alle Tasks einer Mischphase beendet sind, bevor die nächste Phase begonnen wird. Diese Implementierung wird hier angestrebt.

Hinweis Lassen Sie Ihr Programm auf einem Rechner mit mehreren CPUs laufen, z.B. auf einem der Rechner im Grundstudiumspool. Der Linux-Befehl

```
time ./my_prog [params]
```

führt `my_prog` aus und zeigt nach Programmende die Ausführungszeit an. Das Ausgabeformat variiert von System zu System, enthält jedoch mindestens die Anzahl Sekunden Usertime (Zeit während der das Programm ausgeführt wurde), Systemtime (Zeit während der Funktionen des Betriebssystems zur Verwaltung des Programms ausgeführt wurden) und Realtime (Kalenderzeit). Die Zeiten sind dabei Ausführungszeiten für einen Prozessor (d. h. in parallelen Programmen ist die Summe aus User und System größer als die real vergangene Zeit).