Übungsblatt 7

Ausgabe: 10.06.2008
Abgabe: 25.06.2008, 16:00 Uhr

Aufgabe 1 (8 P)

Problemlösungen im Stil des Teile-und-Herrsche (divide-and-conquer) sind wegen ihrer regulären Struktur beliebt. Teile-und-Herrsche kann problemunabhängig wie folgt präzisiert werden:

```
class Problem {
.....
        Solution dc() {
                if(small()) return solve();
                Pair pp = divide();
                Solution s = pp.fst.dc();
                s.combine(pp.snd.dc());
                return s;
        }
}
```

Solche Problemlösungen lassen sich häufig sehr gut parallelisieren; auf einem Parallelrechner wird dann entsprechend weniger Zeit benötigt. Beliebt ist eine als Master/Worker bekannte Strukturierung: eine feste Anzahl von Workers (Prozesse oder Threads) ist für die Lösung von Teilproblemen zuständig (wobei typischerweise neue Teilprobleme generiert werden - siehe oben); ein Master sorgt für die Initialisierung, sammelt die Teilergebnisse ein und stellt sie zum Gesamtergebnis zusammen.

Es gibt einen Auftragspool, in dem sich unerledigte Aufträge (= Beschreibungen von Teilproblemen) befinden, und einen Ergebnispool, in dem sich Teillösungen befinden. Jeder Worker-Prozess entnimmt wiederholt einen Auftrag aus dem Auftragspool. Der Auftrag kann dann entweder direkt erledigt werden, so dass die entsprechende Lösung im Ergebnispool abgelegt werden kann; alternativ kann es erforderlich sein, den Auftrag in kleinere Teilaufträge zu zerlegen und diese wieder im Auftragspool zu deponieren. Zu Beginn deponiert der Master die Beschreibung des Gesamtproblems im Auftragspool. Anschließend übernimmt er die Teillösungen aus dem Ergebnispool und fügt sie zu einer Gesamtlösung zusammen:

```
Master: lege Problem in Auftragspool;
        wiederhole: entnimm nächste Teillösung aus Ergebnispool und kombiniere sie
                    mit bereits entnommenen Teillösungen,bis Gesamtlösung komplett(combine).
Worker: wiederhole: entnimm nächstes Problem aus Auftragspool;
                    falls einfaches Problem(simple), löse es (solve) und deponiere Lösung im Ergebnispool;
                    andernfalls zerlege es in Teilprobleme(divide) und deponiere diese im Auftragspool.
```

Formulieren Sie das Master- bzw. Worker-Verhalten in Java, wobei geeignete Objekte (raytracing.jar -> ThreadBag.java) für die Pools zum Einsatz kommen sollen!

Aufgabe 2 (8 P)

Mit dem Ansatz aus Aufgabe 1 soll ein konkretes Problem gelöst werden: Rendering eines Bildes mittels Raytracing. Ein fertiges Raytracing-Programm finden Sie unter raytracing.jar. Sie können das Programm mit dem ant-Werkzeug ausprobieren. (Es dauert etwas, bis das Bild erscheint.) Dieses Programm ist aber zu simpel gestrickt: es arbeitet nur mit zwei Fäden und ist kein Master/Worker-Programm. Übernehmen Sie die problemtypischen Teile dieses Programms in Ihre Master/ Worker-Lösung! Richten Sie eine feste Anzahl von Worker-Fäden ein, die der Anzahl der verfügbaren Prozessoren entspricht; diese Zahl erfahren Sie mit:

```
Runtime.getRuntime().availableProcessors()
```

Erläutern Sie die Arbeitsweise Ihres Programms unter Bezugnahme auf die jeweiligen Programmteile!

Aufgabe 3 (4 P)

Der Rechner human (alternativ: vader oder paste (nur 2 Prozessoren) [Achtung! Java 1.5 in /import/java1.5/bin/java]) hat vier Prozessoren. Testen Sie ihr Programm auf human unter Verwendung von 1, 2, 3 bzw. 4 Prozessoren (Parameter von main) und messen Sie die Zeiten! [Hinweis: Ein Aufsplitten des Problems in mehr als 4 Teilprobleme macht offenbar keinen Sinn.]

Durch Einsatz mehrerer Prozessoren erzielt man eine gewisse Beschleunigung (speedup), d.i. das Verhältnis der Rechenzeit auf einem Prozessor zur Rechenzeit auf n Prozessoren. Beim Einsatz von zwei Prozessoren würde man etwa eine Beschleunigung von nahezu 2 erwarten. Das Verhältnis von Beschleunigung zu n bezeichnet man als Effizienz, und man hofft, dass die Effizienz nahe bei 1 liegt.

Stellen Sie die erzielten Beschleunigungen und Effizienzen graphisch dar und diskutieren Sie die Ergebnisse!

Hinweise zur Abgabe:

Die Abgabe der Lösungen erfolgt auf zwei Wegen:

- Abgabe auf Papier:

  Bitte schreiben Sie zur jeder Aufgabe eine Dokumentation, in der Sie anhand von *relevanten* Codefragmenten Ihre Implementierung erläutern. Die Codefragmente sollten möglichst kurz gehalten werden und nur der Orientierung für den Leser dienen; entscheidend sind Ihre Erläuterungen, was diese Fragmente machen und wieso Sie sich für diese Art der Implementierung entschieden haben. Die Dokumentation ist *bis Mittwoch 16:00 Uhr in den Tutorenfächern* abzugeben .

- Abgabe per E-Mail:

  Schicken Sie ein ZIP-Archiv an Ihren Tutor, welches den vollständigen Quellcode sowie ausführbare JAR-Dateien für die jeweiligen Aufgabenteile enthält. Der Betreff der E-Mail sollte wie folgt aussehen: "[ALP4] Blatt X - Namen Y".

Zum Bestehen des Übungsblattes müssen 50% der Punkte erreicht werden.