

### Aufgabe 1 (8 Punkte)

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<Problem> 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 ( <code>combine</code> ).
Worker:	wiederhole: entnimm nächstes Problem aus Auftragspool;
	falls einfaches Problem ( <code>simple</code> ), löse es ( <code>solve</code> )

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 für die Pools zum Einsatz kommen sollen! (Zur Erinnerung: Aufgabe 7.2.)

## **Aufgabe 2** (8 Punkte)

Mit dem Ansatz aus Aufgabe 1 soll ein konkretes Problem gelöst werden: *Rendern eines Bildes mittels Raytracing*. Ein fertiges Raytracing-Programm finden Sie unter `raytracing.zip`. 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 Punkte)

Der Rechner *vader* hat vier Prozessoren. Testen Sie ihr Programm auf *vader* 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!