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

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

17. Dezember 2002

# Übungen zu „Parallelität in funktionalen Sprachen“

## Nr. 9, Abgabe: 7. Januar 2003 in der Vorlesung

---

**Abgabe:** Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an `eden@mathematik.uni-marburg.de` abgegeben werden.

Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

---

### Eden-Programmierung

9.1 Implementierungsskelette und Prozessschemata in Eden **[5 Punkte]**

In der Vorlesung wurden verschiedene Prozessschemata und Implementierungsskelette vorgestellt.

a) Definieren Sie eine `map_dm`-Funktion, die das Prozessschema `direct-mapping` der Vorlesung benutzt. Wie die Beispiele der Vorlesung soll sie exakt den Typ von `map` haben und das jeweilige Prozessschema intern benutzen.

b) Erläutern Sie an diesem Beispiel den Unterschied zwischen Prozessschemata und Implementierungsskeletten sowie deren Benutzung in Programmen.

9.2 Mandelbrot in Eden **[10 Punkte]**

Wir betrachten erneut das Programm zur Berechnung von Mandelbrot-Mengen (“Apfelmännchen”) von Übung 8 und vergleichen die Parallelisierung mit GpH mit einigen Parallelisierungsmöglichkeiten, die Eden bietet. Wie Sie bereits feststellen konnten, ist die Verteilung der Arbeit entscheidend für das Laufzeitverhalten. Achten Sie daher auch in dieser Aufgabe auf die Aufteilung.

a) Erstellen Sie eine Version des Mandelbrot-Programms für Eden, die ein vordefiniertes Prozessschema oder ein Implementierungsskelett für `map` benutzt.

b) Erstellen Sie eine weitere Version des Programms, bei der die Arbeitsverteilung explizit erfolgt und nur (evtl.) `parMap` benutzt wird (Hinweise beachten!).

c) Messen Sie die Laufzeit Ihrer beiden Programme und des GpH-Programms mit Hilfe des `time` Befehls. Erstellen Sie dazu von jedem Bildausschnitt jeweils ein 300-Pixel-Bild. Leiten Sie die Ausgabe des Programms in die “Datei” `/dev/null` um und benutzen Sie evtl. den `at` Befehl, um Störfaktoren (Last durch andere Benutzer) auszuschalten.

9.3 Workpool-Schema **[5 Punkte]**

Definieren Sie für das in der Vorlesung vorgestellte `workpool`-Prozessschema die noch fehlenden Funktionen.

a) `tagWithPIDs :: [[a]] -> [[(Int,a)]]` (Nummerieren der Ausgaben)

b) `distribute :: Int -> [Int] -> [a] -> [[a]]` (Arbeitsverteilung)

Beachten Sie dabei, dass die Funktion `distribute` bereits Resultate liefern muss, wenn ihre Eingabe noch nicht vollständig verfügbar ist.

**Hinweise:**

**Compiler für Eden** Der Compiler in `/app/lang/functional/ghc-5.00.2-eden`, den wir bereits für GpH verwendet haben, besitzt eine Erweiterung für Eden. Bei Problemen sollten Sie aber den neu installierten Compiler in `/app/lang/functional/ghc-5.02.3-eden` benutzen (jeweils im Verzeichnis `bin`). Eden-Programme müssen mit der Option `-eden` übersetzt werden und das Modul *Eden* importieren.

**Skelett-Modul** Zur Lösung der Aufgaben sollten Sie das Modul `EdenSkel.lhs` verwenden, das auf der Vorlesungsseite zum Download bereitsteht. Es enthält bereits die Prozessschemata `parMap`, `farm` und `direct_mapping` in exakt den Versionen, die in der Vorlesung vorgestellt wurden.

**Workaround** Wo Sie kein Implementierungsskelett verwenden, benutzen Sie zur Prozesserzeugung bitte die Funktion `eagerInstList` anstelle des Operators `(#)`. Ansonsten werden die gewünschten Prozesse u.U. verteilt sequenziell erzeugt, was mit Unzulänglichkeiten des Compilers zusammenhängt.

`eagerInstList` erhält eine Liste von Prozessabstraktionen und eine Liste passender Inputs dafür. Ergebnis ist eine Liste der Ausgaben der erzeugten Prozesse.

```
eagerInstList :: (Trans a, Trans b) =>
                 [Process a b ] -> [a] -> [b]
```

**Frohe Weihnachten**
**und einen**
**“Guten Rutsch”**
**ins Neue Jahr 2003**