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

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

7. Januar 2003

Übungen zu „Parallelität in funktionalen Sprachen“

Nr. 10, Abgabe: 14. 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: Workpool und Divide & Conquer

10.1 Inkrementelle Funktionen für das Workpool-Schema — 5 Punkte

a) Definieren Sie für das in der Vorlesung vorgestellte `workpool`-Prozessschema die noch fehlenden Funktionen.
   - `tagWithPIDs :: [[a]] -> [[(Int,a)]]` (Nummerieren der Ausgaben)
   - `distribute :: Int -> [Int] -> [a] -> [[a]]` (Arbeitsverteilung)

b) Beachten Sie, dass die Funktion `distribute` im Schema bereits Resultate liefern muss, wenn ihre Eingabe `[Int]` noch nicht vollständig verfügbar ist. Ebenso muss `tagWithPIDs` alle Listen *kontinuierlich* um verfügbare Daten ergänzen, die nach und nach eintreffen. Beide Funktionen müssen also “inkrementell” arbeiten. Prüfen Sie, ob Ihre Funktionen diese Eigenschaft haben und geben Sie ggf. inkrementell arbeitende Varianten an.

10.2 Workpool-Implementierung für `map` [1] — 7 Punkte

a) Modifizieren Sie das `workpool`-Schema der Vorlesung, so dass die Reihenfolge der Elemente in der Taskliste erhalten bleibt.

b) Definieren Sie ein Implementierungsskelett `map_pool` mit Ihrem `workpool`-Schema und testen Sie es mit dem Mandelbrot-Programm.

10.3 divide&conquer-Schema — 8 Punkte

a) Mit einer Funktion höherer Ordnung kann ein allgemeines Schema für Algorithmen mit divide & conquer ausgedrückt werden. Definieren Sie eine solche Funktion `divCon` mit dem angegebenen Typ in Eden:

```
divCon :: (Trans a, Trans b) =>
          (a->Bool) -> (a->b) -> (a->[a]) -> (a->[b]->b)
           -> Process a b
divCon trivial solve split combine = ...
```

b) Definieren Sie mittels `divCon` eine Funktion `parquicksort`, die ein paralleles Quick-Sort in Eden realisiert. Bauen Sie dabei auch ein “Umschalten” auf ein sequenzielles Sortierverfahren ein und vergleichen Sie die Laufzeiten mit verschiedenen Abbruchkriterien.

[1] Beachten Sie bitte bei Ihren Lösungen, dass `merge` aktuell nicht als Prozess, sondern als Funktion implementiert ist: `merge :: [[a]] -> [a]`