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

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

14. Januar 2003

Übungen zu „Parallelität in funktionalen Sprachen“

Nr. 11, Abgabe: 21. 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

11.1 Workpool-Schema mit dynamischen Kanälen — 7 Punkte

Wir betrachten die workpool-Version der Vorlesung, bei der die Arbeitsaufträge über dynamische Kanäle verteilt werden:

```
wp_dyn :: (Trans a, Trans b) => Int -> (a -> b) -> [a] -> [b]
wp_dyn np f tasks = getResults (merge workerlist) restlist
    where
       worker = process (\t -> compute f t)
          where compute f (Just t) = new (\chan new_task ->
                                            (chan, f t):(compute f new_task))
                compute f Nothing  = []
       workerlist = [worker # t | t <- (map Just init)]
       -- bzw. workerlist = eagerInstList (repeat worker) ( map Just init)
       getResults ((c,r):fws) []     = parfill c Nothing  (r:(getResults fws []))
       getResults ((c,r):fws) (t:ts) = parfill c (Just t) (r:(getResults fws ts))
       getResults [] ts = []
       (init, restlist) = splitAt np tasks
```

a) Bestimmen Sie die Typen aller lokalen Definitionen im `where`-Ausdruck.

b) Die vorgestellte Version arbeitet aufgrund ihrer Definition immer mit je einem Arbeitsauftrag pro Prozess (prefetch von 1). Die Prozesse haben also nach der Bearbeitung vorübergehend keinen Arbeitsauftrag.
Modifizieren Sie das Prozessschema, so dass die Prozesse eine Liste der Länge `prefetch` von Aufträgen erhalten und somit ohne Wartezeit arbeiten. Welche Typen aus Teil a) ändern sich dabei?

11.2 Paralleles map-reduce — 7 Punkte

a) Definieren Sie parallele Implementierungsskelette mit `farm` und `workpool` für die wie folgt spezifizierte Funktion (Reduktion einer transformierten Liste):

```
mapReduce :: (a -> b) -> (c -> b -> c) -> c -> [a] -> c
mapReduce f g e tasks = foldl g e (map f tasks)
```

b) Legen Sie dar, welche Voraussetzungen die Funktionen f und g in Ihren Implementierungsskeletten erfüllen müssen.

11.3 Kostenmodell für map_pool — 6 Punkte

Entwerfen Sie analog zu den in der Vorlesung vorgestellten Kostenmodellen ein Kostenmodell für das Implementierungsskelett `map_pool` (sortierende Version ohne dynamische Kanäle). Gehen Sie dabei von einer perfekten Lastverteilung aus, d.h. arbeiten Sie mit Durchschnittswerten für die Zahl der Arbeitsaufträge und ihren Aufwand.