Prof. Dr. R. Loogen, M. Dieterle
Fachbereich Mathematik und Informatik
Hans-Meerwein-Straße
D-35032 Marburg

26. Oktober 2010

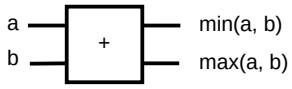
Philipps Universität Marburg

Übungen zu „Parallelität in funktionalen Programmiersprachen“, WS 2010

Nr. 2, Abgabe der Aufgaben: 2. November 2010 vor der Vorlesung

---

Odd-Even-Merge-Sort ... ist ein paralleles Sortierverfahren bei dem sogenannte *Komparatoren* zu einem *Sortiernetz* verbunden werden. Ein Komparator erhält zwei Eingaben und produziert zwei Ausgaben, das Minimum und das Maximum der beiden Eingaben:



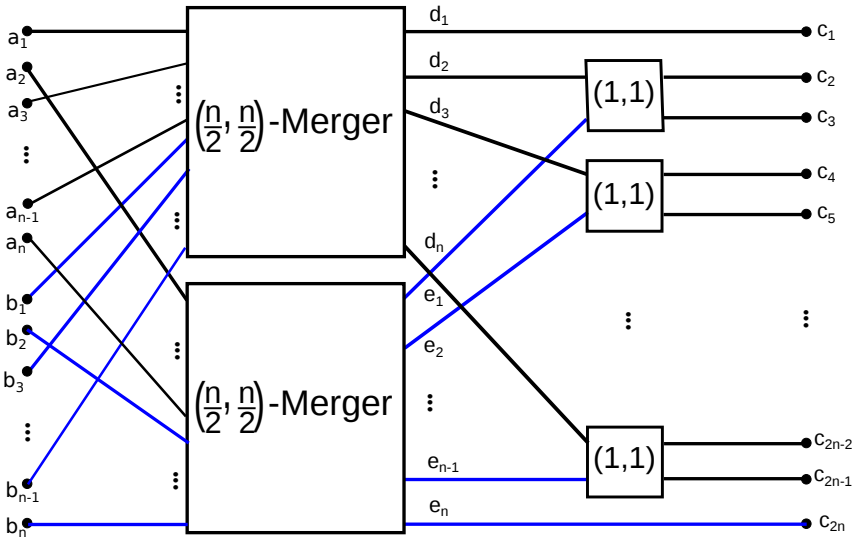
Induktive Konstruktion eines (n,n)-Mergers $\left(n = 2^k\right)$ ($\longrightarrow$ odd-even-merger)
Zunächst werden (n,n)-Merger gebildet, die je zwei sortierte Folgen

$$\begin{aligned} A &= (a_1, \ldots, a_n) \\ B &= (b_1, \ldots, b_n) \end{aligned}$$

in eine sortierte Folge mischen.

Fall $n = 1$: Ein Komparator ist ein $(1,1)$-Merger.

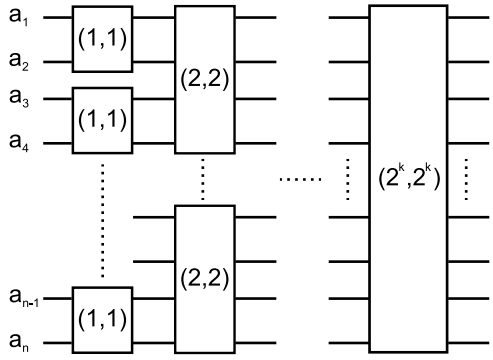
Fall $n > 1$: Durch den induktiven Aufbau stehen bereits $\left(\frac{n}{2}, \frac{n}{2}\right)$-Merger zur Verfügung. Ein $\left(\frac{n}{2}, \frac{n}{2}\right)$-Merger wird mit den ungeraden Elementen der Liste A und den ungeraden Elementen der Liste B verbunden, ein weiteren mit den geraden Elementen der Liste A und den geraden Elementen der Liste B. Der erste Output des ersten $\left(\frac{n}{2}, \frac{n}{2}\right)$-Mergers d_1 ist das erste Element der Outputliste c_1, der letzte Output des zweiten $\left(\frac{n}{2}, \frac{n}{2}\right)$-Mergers e_n ist das letzte Element der Outputliste c_{2n}. Alle weiteren Outputs der d_{i+1} und e_i der $\left(\frac{n}{2}, \frac{n}{2}\right)$-Merger mit $i \in \{1, ..., n-1\}$ werden noch paarweise mit Komparatoren verglichen.



Die Resultatfolge C ist sortiert.

Bitte wenden!

Aufbau eines Sortiernetzes aus (n, n)-Mergern



Aufgaben

2.1 Odd-Even-Merge-Sort in Haskell

12 Punkte

Implementieren Sie das beschriebene Verfahren mit folgender Signatur in Haskell:

```
comparator          :: Ord a ⇒ a → a → [a]
nToNMerge           :: Ord a ⇒ [a] → [a] → [a]
sortNet, sortNetPar :: Ord a ⇒ [a] → [a]
sortNetParD         :: Ord a ⇒ Int → [a] → [a]
```

(a) Programmieren Sie einen Komparator. / 1

(b) Definieren Sie nun rekursiv einen (n, n)-Merger, der dem oben beschriebenen induktiven Aufbau folgt. / 3

(c) Setzen Sie nun die (n, n)-Merger zu einem rekursiv definierten Sortiernetz zusammen. / 2

(d) Schreiben Sie eine erste parallele Version `sortNetPar` des Sortiernetzes, die die Funktion `sortNet` mit Hilfe von `par` und `pseq` parallelisiert. Parallelisieren Sie dabei alle rekursiven Aufrufe von `sortNetPar`. / 2

Achtung: Da `par` und `pseq` nur die Auswertung zur Kopfnormalform anstoßen bzw. erzwingen, führt deren Anwendung auf Listen nur zur Auswertung des ersten Listenkonstruktors. Man kann sogenannte *evaluation transformer* anwenden um Listen strikter auswerten zu lassen. Der Aufruf `(length xs) `par` ...` bzw. `(length xs) `pseq` ...` für eine Liste `xs` führt z.B. zur Auswertung der kompletten Listenstruktur und somit auch zur Auswertung der Anordnung der Element.

(e) Schreiben Sie eine weitere parallele Version `sortNetParD`, die zusätzlich einen Parameter `d` zur Tiefenkontrolle der parallelen Rekursion benutzt. Nach `d` rekursiven Aufrufen soll mit der sequentiellen Version aus Aufgabenteil (c) weitergerechnet werden. / 2

(f) Vergleichen Sie die verschiedenen Versionen auf dem 8-Kern Rechner `sakania` in einer ausführlichen Messreihe (unterschiedliche Eingabegrößen, unterschiedliche Prozessorzahlen, ...) und interpretieren Sie die gemessenen Zeiten. / 2

Kompilieren: Zur Zeit können Sie Ihre Programme leider nicht auf `sakania` kompilieren, benutzen Sie z.B `tanga`. Übersetzen Sie Ihr Programm mit dem Flag `-threaded`, damit es mit dem Threaded Laufzeitsystem gelinkt wird.

Beispiel: `ghc --make -O2 -threaded -fforce-recomp sortnet.hs`

Ausführen: Wechseln Sie mit `ssh` auf den Rechner sakania. Verwenden Sie die RTS-Option `-N<n>` zur Benutzung von `<n>` Prozessen. Eventuell ist es nötig die Stack-Größe mit der Option `-K<size>` zu erhöhen, mit z.B. `size = 1M`. Eine Erhöhung der Heap Größe (`-H<size>`) kann zur Beschleunigung der parallelen Laufzeit führen. Zur Messung der parallelen Laufzeit können Sie das Programm `time` auf der Konsole verwenden.

Beispiel: `time ./sortnet 3 65536 +RTS -K1M -N2 -H50M`