Heutiges Thema

Parallelität

- Implizit
- Explizit
- Skelette

Literaturhinweis

Die folgende Darstellung orientiert sich an...

- Kapitel 21
  Peter Pepper, Petra Hofstedt. *Funktionale Programmierung*, Springer, 2006.

Im Kontext relevante Literaturstellen...

- Murray Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*, The MIT Press, 1989.
- Philip W. Trinder, Kevin Hammond, Hans-Wolfgang Loidl, Simon L. Peyton Jones. *Algorithms + Strategy = Parallelism*. Journal of Functional Programming, 8(1):23-60, 1998.
- Philip W. Trinder, Hans-Wolfgang Loidl, Robert F. Pointon. *Parallel and Distributed Haskells*. Journal of Functional Programming, 12(4&5):469-510, 2002.

Parallelität in imperativen Sprachen

Insbesondere...

- Datenparallele Sprachen (z.B. High Performance Fortran)
- Bibliotheken (PVM, MPI) / *Message Passing Modell* (C, C++, Fortran)

Parallelität in funktionalen Sprachen

Insbesondere...

- Implizit/Ausdrucksparallelität
- Explizit
- Algorithmische Skelette

Implizite Parallelität

...bzw. *Ausdrucksparallelität*

Betrachte funktionalen Ausdruck der Art `f(e1,...,en)`:

Beachte:

- Argumente (und Funktion) können parallel ausgewertet werden.
- Vorteile: Parallelisierung *for free*! Kein Aufwand für den Programmierer.
- Nachteile: Ergebnisse oft unbefriedigend. Z.B. Granularität, Lastverteilung, etc. unberücksichtigt.

Mithin:

- Einfach (für den Übersetzer) zu erkennende Parallelität, schwierig umfassend auszunutzendes Potential.

Explizite Parallelität

Durch...

- Einführung von Metaanweisungen (etwa zur Steuerung von Daten- und Lastverteilung, Kommunikation)
- Vorteile: Möglicherweise überlegene Ergebnisse durch explizite Kontrolle in der Hand des Programmierers
- Nachteile: Hoher Programmieraufwand

Algorithmische Skelette

Mittelweg zwischen...

- *expliziter imperativer* Parallelprogrammierung
- *impliziter funktionaler* Parallelprogrammierung

In der Folge

- Massiv parallele Systeme
- Algorithmische Skelette (engl. algorithmic skeletons)

Massiv parallele Systeme

...charakterisiert durch

- große Anzahl von Prozessoren mit
  - lokalem Speicher
  - Kommunikation durch Nachrichtenaustausch
- MIMD-Parallelrechnerarchitektur (*Multiple Instruction Multiple Data*)
- Hier: SPMD-Programmierstil (*Single Program Multiple Data*)

Algorithmische Skelette

Algorithmische Skelette...

- repräsentieren typische Parallelisierungsmuster (*Farm*, *Map*, *Reduce*, *Branch&Bound*, *Divide&Conquer*,...)
- sind für den Programmierer einfach zu instantiieren
- erlauben parallele Programmierung auf hohem Abstraktionsniveau

Realisierung algorithmischer Skelette

...in funktionalen Sprachen

- durch spezielle Funktionale
- mit paralleler Implementierung
- eingebettet in sequentielle Sprachen

Somit

- Verbergen paralleler Implementierungsdetails im Skelett
- Eleganz und (parallele) Effizienz für spezielle Anwendungsmuster

Beispiel: Paralleles Map auf verteilter Liste

Betrachte das Funktional `map` auf Listen...

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = (f x) : (map f xs)
```

Beobachtung

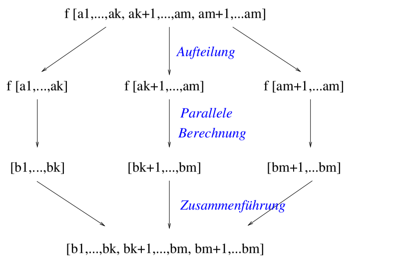
- Anwendung von `f` auf ein Listenelement unabhängig von anderen Listenelementen

Naheliegend

- Teilung der Liste in Teillisten und *parallele* Anwendung von Map auf die Teillisten (Parallelisierungsmuster *Farm*)

Paralleles Map auf verteilter Liste

Zur Illustration...



Peter Pepper, Petra Hofstedt. Funktionale Programmierung. Springer, 2006, S. 445.

Zur Realisierung

Implementierung der parallelen Map-Funktion erfordert...

- spezielle Datenstrukturen, die den Verteilungsaspekt berücksichtigen (herkömmliche Listen nicht effizient)

Skelette auf verteilten Datenstrukturen

- sog. *datenparallele Skelette*

Unterscheidung

- *Datenparallelität*: Verteilung der Daten a priori auf verschiedenen Prozessoren vorausgesetzt
- *Taskparallelität*: Zu verteilende Prozesse und Daten nicht a priori bekannt, sondern dynamisch erzeugt

Programmierung paralleler Anwendung

...mittels algorithmischer Skelette

- Erkennung probleminhärenter Parallelität
- Auswahl angemessener Datenverteilungen (Granularität)
- Auswahl passender Skelette aus Bibliothek
- Problemspezifische Skelettinstanziierung

Bemerkung:

- Einige Sprachen (etwa Eden) unterstützen auch die Implementierung von Skeletten

Datenverteilung auf Prozessoren

...ist

- entscheidend für
  - Struktur des Gesamtalgorithmus
  - Effizienz

Schwierigkeit abhängig von

- Unabhängigkeit aller Datenelemente (wie im Map-Beispiel): Verteilung einfach
- Unabhängigkeit von Teilmengen von Datenelementen
- Komplexe Abhängigkeiten der Datenelemente: Angemessene Verteilung herausfordernd

Ein Hilfsmittel

- Sog. *Covers* (von verschiedenen Autoren untersucht)

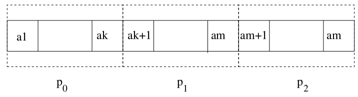
Cover

...beschreiben

- Zerlegung und Kommunikationsmuster einer Datenstruktur

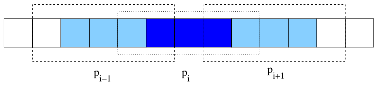
Beispiel: Einfaches Listen-Cover

Aufteilung einer Liste auf 3 Prozessoren p_1, p_2 und p_3:



Peter Pepper, Petra Hofstedt. Funktionale Programmierung. Springer, 2006, S. 446.

Beispiel: Listen-Cover mit überlappenden Elementen



Peter Pepper, Petra Hofstedt. Funktionale Programmierung. Springer, 2006, S. 446.

Allgemeine Cover-Struktur

```
Cover = {
  Type S a      -- Gesamtobjekt
       C b      -- Cover
       U c      -- Lokale Subobjekte

split :: S a -> C (U a)   -- Zerlegen des Originalobjekts
glue :: C (U a) -> S a    -- Zusammenfuegen des Originalobjekts
}
```

Dabei solle gelten:

```
glue . split = id
```

Beachte: Kein Haskell

Programmiersprachliche Umsetzung

...von Covern erfordert Unterstützung

- zur Spezifikation von Covern
- zur Programmierung von algorithmischen Skeletten über Covern
- zur Verfügungstellung häufig benutzter Skelette in Bibliotheken

...ist

- aktuelles Forschungsthema funktionaler Programmierung

Weiterführender Literaturhinweis

- Hans-Werner Loidl et al. *Comparing Parallel Functional Languages: Programming and Performance*. Higher-Order and Symbolic Computation, 16(3):203-251, 2003.

Nächster Vorlesungstermin

- Mi, 28.06.2006, 16 - 18 Uhr, CompLang-Bibliothek