Programmiersprachen im Multicore-Zeitalter

Haskell

Colin Benner

6. 2. 2012

Einführung

- Warum Parallelverarbeitung?

Einführung

- Warum Parallelverarbeitung?
- Warum Haskell?

Haskell-Grundlagen

- Syntax

```
filter :: (a -> Bool) -> [a] -> [a]
filter p [] = []
filter p (x:xs) | p x       = x : filter p xs
                | otherwise = filter p xs
```

- Keine Nebeneffekte
- lazy evaluation

Parallele Programmierung mit Haskell

- Glasgow parallel Haskell
- Concurrent Haskell
- Data Parallel Haskell

Glasgow parallel Haskell

Funktionen

```
par  :: a -> b -> b
pseq :: a -> b -> b
```

Beispiel

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = a + b
  where a = fib (n-1)
        b = fib (n-2)
```

Glasgow parallel Haskell

Funktionen

```
par  :: a -> b -> b
pseq :: a -> b -> b
```

Beispiel

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = a `par` a + b
  where a = fib (n-1)
        b = fib (n-2)
```

Glasgow parallel Haskell

Funktionen

```
par :: a -> b -> b
pseq :: a -> b -> b
```

Beispiel

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = a `par` (b `pseq` a + b)
  where a = fib (n-1)
        b = fib (n-2)
```

Concurrent Haskell

- Explizites Thread-Management
- Kommunikation mit
  1. MVars
  2. Kanälen
  3. Software Transactional Memory

Data Parallel Haskell

Berechne Produkt von dünn besetzter Matrix und Vektor

```
type Vector = [ Float ]
type SparseVector = [ (Int, Float) ]
type SparseMatrix = [ SparseVector ]

sparseVecMul :: SparseVector -> Vector -> Float
sparseVecMul sv v = sum [ x * v!!i | (i,x) <- sv ]

sparseMatMul :: SparseMatrix -> Vector -> Vector
sparseMatMul sm v = [ sparseVecMul r v | r <- sm ]
```

Data Parallel Haskell

Berechne Produkt von dünn besetzter Matrix und Vektor

```
type Vector = [: Float :]
type SparseVector = [: (Int, Float) :]
type SparseMatrix = [: SparseVector :]

sparseVecMul :: SparseVector -> Vector -> Float
sparseVecMul sv v = sumP [: x * v!:i | (i,x) <- sv :]

sparseMatMul :: SparseMatrix -> Vector -> Vector
sparseMatMul sm v = [: sparseVecMul r v | r <- sm :]
```

Erweiterungen

- Eden:
  www.mathematik.uni-marburg.de/~eden/
- Glasgow Distributed Haskell:
  www.macs.hw.ac.uk/~dsg/gdh/
- Parallel Haskell:
  csg.csail.mit.edu/projects/languages/ph.shtml
- Eager Haskell:
  csg.csail.mit.edu/pubs/haskell.html

Literatur

- Haskell Homepage: haskell.org
- GHC: haskell.org/ghc/
- Real World Haskell: book.realworldhaskell.org (Kapitel 24)
- Concurrent and Parallel Programming: haskell.org/haskellwiki/Applications_and_libraries/Concurrency_and_parallelism
- A Tutorial on Parallel and Concurrent Programming in Haskell: research.microsoft.com/en-us/um/people/simonpj/papers/parallel/AFP08-notes.pdf
- Data Parallel Haskell: haskell.org/haskellwiki/GHC/Data_Parallel_Haskell
- Vortrag über Data Parallel Haskell: youtube.com/watch?v=NWSZ4c9yqW8