# Haskell parallel

- Haskell unterscheidet auf Ebene des Typsystems zwischen reinen und unreinen Berechnungen (Seiteneffekte)
- Sehr viel Umstrukturierung durch den Compiler möglich
- Reine Berechnungen könnten fast ohne weiteres parallelisiert werden, insbesondere z.B. *map* (vgl. Google™ MapReduce™)
- Funktionen als First Class Citizens - könnte helfen

## Definition

Fazit Parallelismus beinahe implizit verfügbar

# Eden

- Paralleler Haskell-Dialekt
- Übersetzt von gewissen Haskell-Funktionen nach z.B. MPI

# Trans & Process

**Definition**

Trans Typklasse für “übertragbare” Daten

**Definition**

Process Typklasse für “übertragbare” Funktionen

# Spawn

Spawn u.A. dienen dazu, Berechnungen “auszulagern” an einen anderen Rechner. Vergleichbar mit MPI *spawn*.
Expliziter Parallelismus in Haskell.

# Monaden

- Haskell-Konstrukt für “imperative Programmierung”
- vgl. “Monoid”
- Typ höherer Ordnung (“Kind” $* \rightarrow *$')
- Beispiel: $print : String -> IO()$
- *do*-Syntax: “imperativer Programmblock”

## Definition

IO IO a: “Skript mit Rückgabewert a” - Achtung, IO ist atypisch!

Listen als Monade: “Nicht-Deterministische” Evaluation ohne Aufpreis.

# ST-Monade

- mittels *runST* umzuwandeln in eine “reine” Berechnung: $runST :: (forall s.STsa) -> a$
- ermöglicht monadische Formulierung von imperativen Programmen ohne Abhängigkeit von “unreinen” Berechnungen
- u.A. Arrays, veränderbare Referenzen etc.

# ST-Monade und Eden

- Problem: $Trans(STx)$ instanziieren?
- Rein technisch wohl möglich
- besser erstmal nicht...

- Lieber “Array” (reiner Typ) übertragen
- Noch besser vielleicht als “Remote Data”

# Aber...

```
Som/Parallel.hs:62:12:
    Could not deduce (Trans d, Trans i) arising from a use of ‘spawn’
    from the context (Show d,
                      Show i,
                      Ix i,
                      DataPoint d,
                      Coordinate i,
                      Inf d,
                      Trans (Array i d))
      bound by the type signature for
                 learndataparallel :: (Show d, Show i, Ix i, DataPoint d,
                                       Coordinate i, Inf d, Trans (Array i d)) =>
                                      Float -> [Array i d] -> d -> [(i, d)]
      at Som/Parallel.hs:(61,1)-(64,9)
    Possible fix:
      add (Trans d, Trans i) to the context of
        the type signature for
          learndataparallel :: (Show d, Show i, Ix i, DataPoint d,
                                Coordinate i, Inf d, Trans (Array i d)) =>
                               Float -> [Array i d] -> d -> [(i, d)]
    In the expression:
      spawn
        (repeat
         $ process
           $ \ s
               -> runST
                  $ do { som_ <- freeze s;
                         findclosest som_ datapoint })
```

Haskell-Typisches Problem: Zusammentreffen von

- Mehreren Constraints
- Typen höherer Ordnung
- Unspezifizierte Typen

Verlass Dich nie auf Hindley-Milner!

Lösung:
In mehrere Unterfunktionen mit klaren Typen aufgliedern
Hilfreiche Funktion:

$$thawST :: (Ix\ i,\ IArray\ a\ e) => a\ i\ e -> ST\ s\ (STArray\ s\ i\ e)$$
$$thawST = thaw$$

(Typ eingeschränkt ggü. thaw!)
2 Zeilen klären eine ganze Menge Typ-Unklarheiten

# Remote Data

- Verwaltung von Daten als “Handle”
- Vermeidet unnötige Übertragung
-