Parallel Programming Support in Haskell

Jingun Hong

Haskell?

- Purely functional programming language
  - with non-strict semantics (lazy evaluation)
  - and strong static typing
- Lots of new programming language resear ch is done with (and on) **Haskell**
  - **“Powered** by Ph.D”

Examples in Haskell

- Hello World

```
main :: IO ()
main = putStrLn "Hello, World!"
```

- Factorial

```
Factorial 0 = 1
Factorial n = n * factorial (n-1)
```

Haskell and Parallel Programming

- No side effects in every subprograms
  - They don't
    - modify any state
    - interact with calling functions or the outside world
- All functions can be executed in parallel
  - Too fine grained

PP Supports in Haskell (1)

- Implicit Parallelism
  - Users write sequential codes
  - Compiler detects parallelism automatically
  - Hard to predict how program will be parallelized

- Feedback Directed Implicit Parallelism (ICFP'07)
  - Profile whole program and detect source of parallelism and recompile program

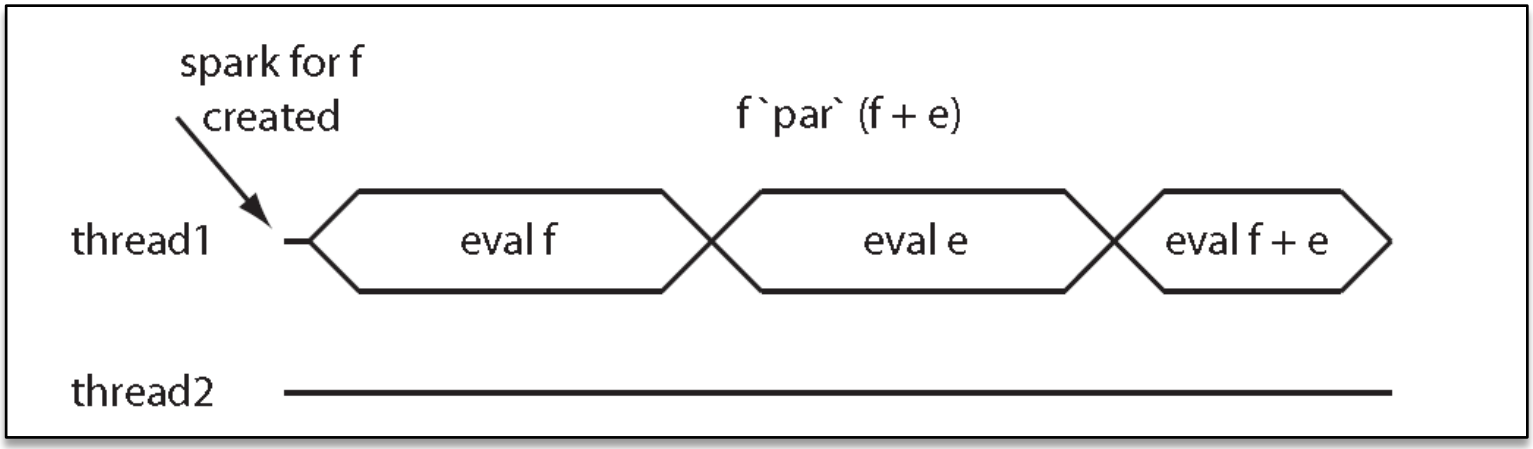
PP Supports in Haskell (2)

- Semi-Explicit Parallelism
  - User provides a hint about the appropriate level of granularity for parallel operations
- **`Control.Parallel`** module supports
  - **`par :: a -> b -> b`**
  - The function **`par`** indicates to the Haskell run-time system that it may be beneficial to evaluate the first argument in parallel with the second argument

PP Supports in Haskell (2)

- Example

```
foo m n = f `par` (f + e)
    where f = bar1 m
          e = bar2 n
```

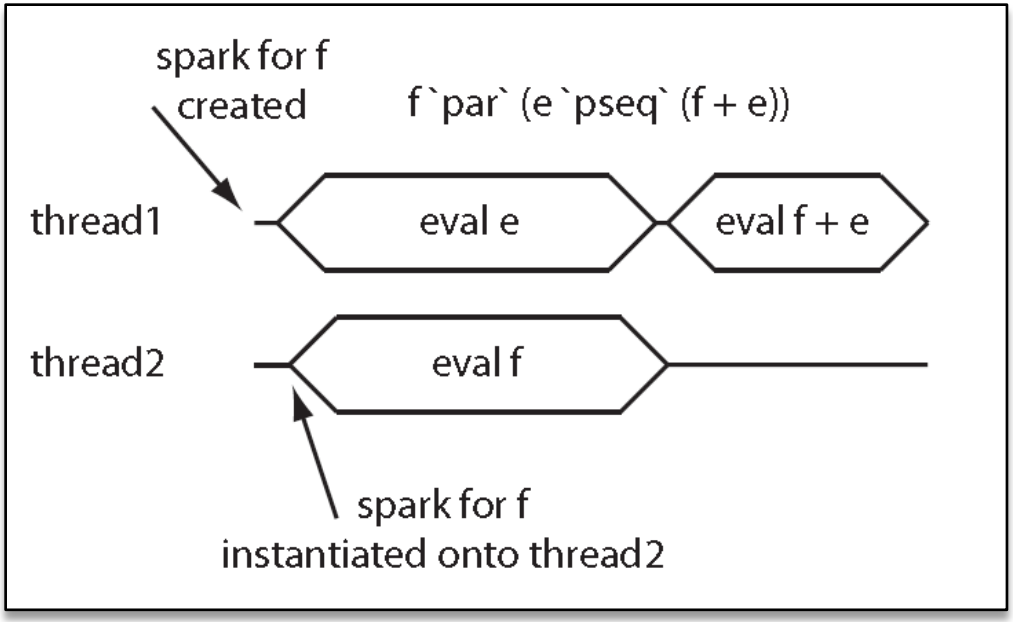


A spark that does not get instantiated onto a thread

PP Supports in Haskell (2)

- Example 1

```
foo m n = f `par` (e `pseq` f + e)
    where f = bar1 m
          e = bar2 n
```



PP Supports in Haskell (3)

- Explicit Parallelism
  - Semi-explicitly parallel program doesn't work with the IO monad
  - User writes explicitly threaded programs
  - Rendezvous among threads : Lock and STM

```
atomically (do v <- readTVar bal
               writeTVar bal (v+1)
           )
```

STM example
Atomically increments a shared Tvar value bal

Parallel Programming Support with Extension of Haskell

Data Parallel Haskell

- Supports new list type for data parallelism
  - [: Double :], [: Int :]
  - [: x * y | x <- xs | y <- ys :]

- Performs code vectorisation and parallel execution on multicore systems

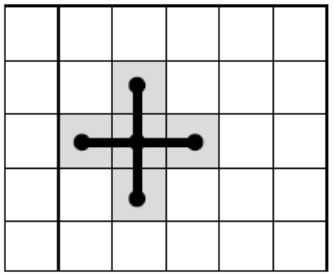
http://haskell.org/haskellwiki/GHC/Data_Parallel_Haskell

Ypnos

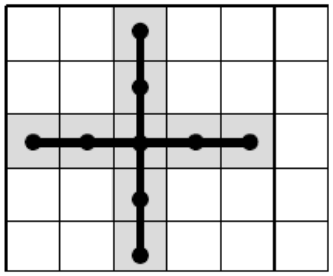
- Domain specific language(DSL) for structured-grid programming
  - DSL offer problem or implementation specific
    - Expressivity
    - Optimisations
- Embedded into Haskell
  - Embedded DSLs
    - Inexpensive DSL
    - Reuse host language syntax, semantics, libraries

Structured Grids

- Arrays representing discretised environments
- Apply a stencil function over an array
- Computes a new value for each array element from neighbours



(a) 5-point stencil



(b) 9-point stencil

- Typical in scientific computing, graphics, games, and so on
- Usually solve approximations to differential equations over discrete space: fluid dynamics

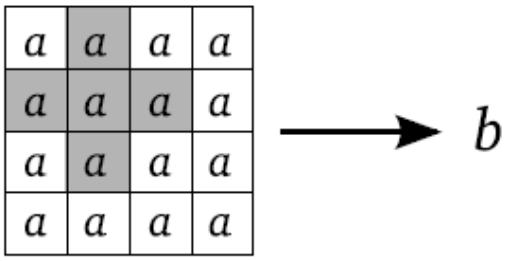
Ypnos Supports

- Grid data structure
- User-defined stencil functions
- Various primitive operations
- Haskell-like syntax

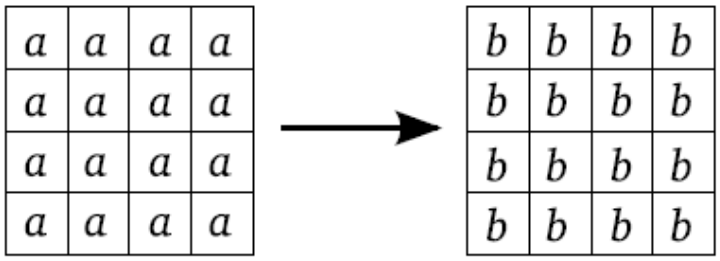
Ypnos Example

```
laplace :: Grid (X * Y) Double -> Double
laplace (X * Y): | _ t  _ | = (t+l+r+b)/4.0
                 | l @_ r |
                 | _ b  _ |
```

```
a = grid <X = 4, Y = 4> data
b = run laplace (defaults 0.0 a)
```



$f :: \text{Grid}\ a \longrightarrow b$



$\text{run}\ f :: \text{Grid}\ a \longrightarrow \text{Grid}\ b$

Summary

- Parallel programming support in Haskell
  - Implicit parallelism
  - Semi-Explicit parallelism
  - Explicit parallelism
- Type extensions for PP with Haskell
  - Data parallel Haskell
  - Ypnos