Concepts in Parallel Programming

Arrows for Parallel Computation

Martin Braun

University of Bayreuth
martinbraun123@aol.com

July 31, 2018

1. What has happened until now?

2. What is part of the MA thesis?
   - Improvements
   - Futures
   - Topology Skeletons
   - Benchmarks
   - Experiment: Cloud Haskell backend

Functional Programming 101

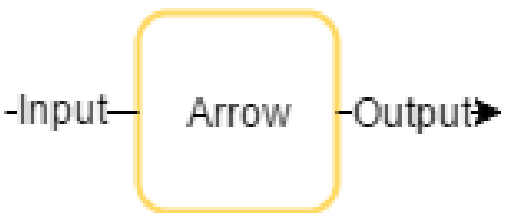
```
1 public static int fib(int x) {
2   if (x<=0)
3     return 0;
4   else if (x==1)
5     return 1;
6   else
7     return fib(x-2) + fib(x-1);
8 }
```

```
1 fib :: Int -> Int
2 fib x
3   | x <= 0 = 0
4   | x == 1 = 0
5   | otherwise =
6     (fib (x - 2))
7       + (fib (x - 1))
```

- Functional programming equally powerful as imperative programming
- focused on the "what?" instead of the "how?"
  $\Rightarrow$ more concise $\Rightarrow$ easier to reason about
- based on Lambda Calculus

Arrows (1)

Another way to think about computations:

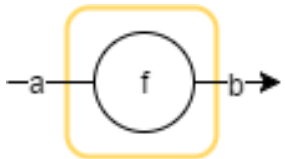


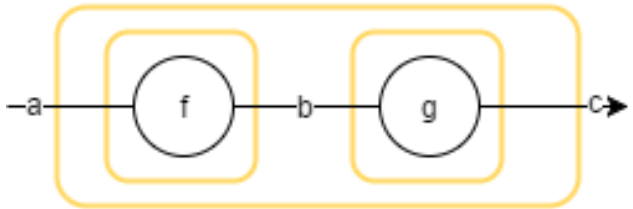
Arrows (2)

```
class Arrow arr where
  arr  ::  (a -> b) -> arr a b

  (>>>) :: arr a b -> arr b c -> arr a c

  first  ::  arr a b -> arr (a,c) (b,c)
```

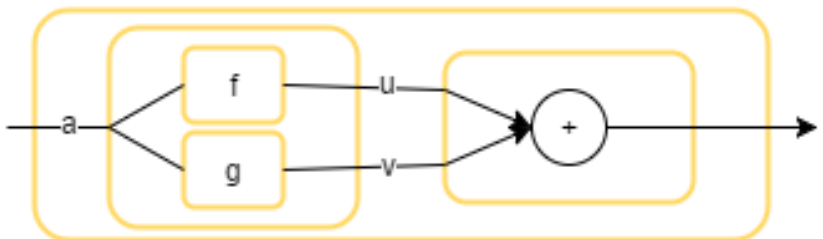




Arrow Example

Arrow usage example:

```
1 add :: Arrow arr => arr a Int -> arr a Int -> arr a Int
2 add f g = (f &&& g) >>> arr (\(u, v) -> u + v)
```



The ArrowParallel typeclass

```
1 class Arrow arr => ArrowParallel arr a b conf where
2   parEvalN :: conf -> [arr a b] -> arr [a] [b]
```

Implementation for several backends:

- GpH
- Par Monad
- Eden

More things from the MA project

More things done in the MA project:

- Syntactic Sugar

```
1 (|***|)  ::
2   arr  a  b  -> arr c d ->
3   arr  (a,  c)  (b,  d)
```

- Basic mapping skeletons

```
1 parMap ::
2   conf  -> arr a b -> arr [a]  [b]
```

- first benchmarks

Profit

So... What does this get us?

- Arrow based Haskell ⇒ **Free Parallelism** for (other) Arrows
- **Replaceable Backends** ⇒ Easier Development
- possible **common interface** for parallelism
- Arrows are quite **intuitive** for parallelism

1. What has happened until now?

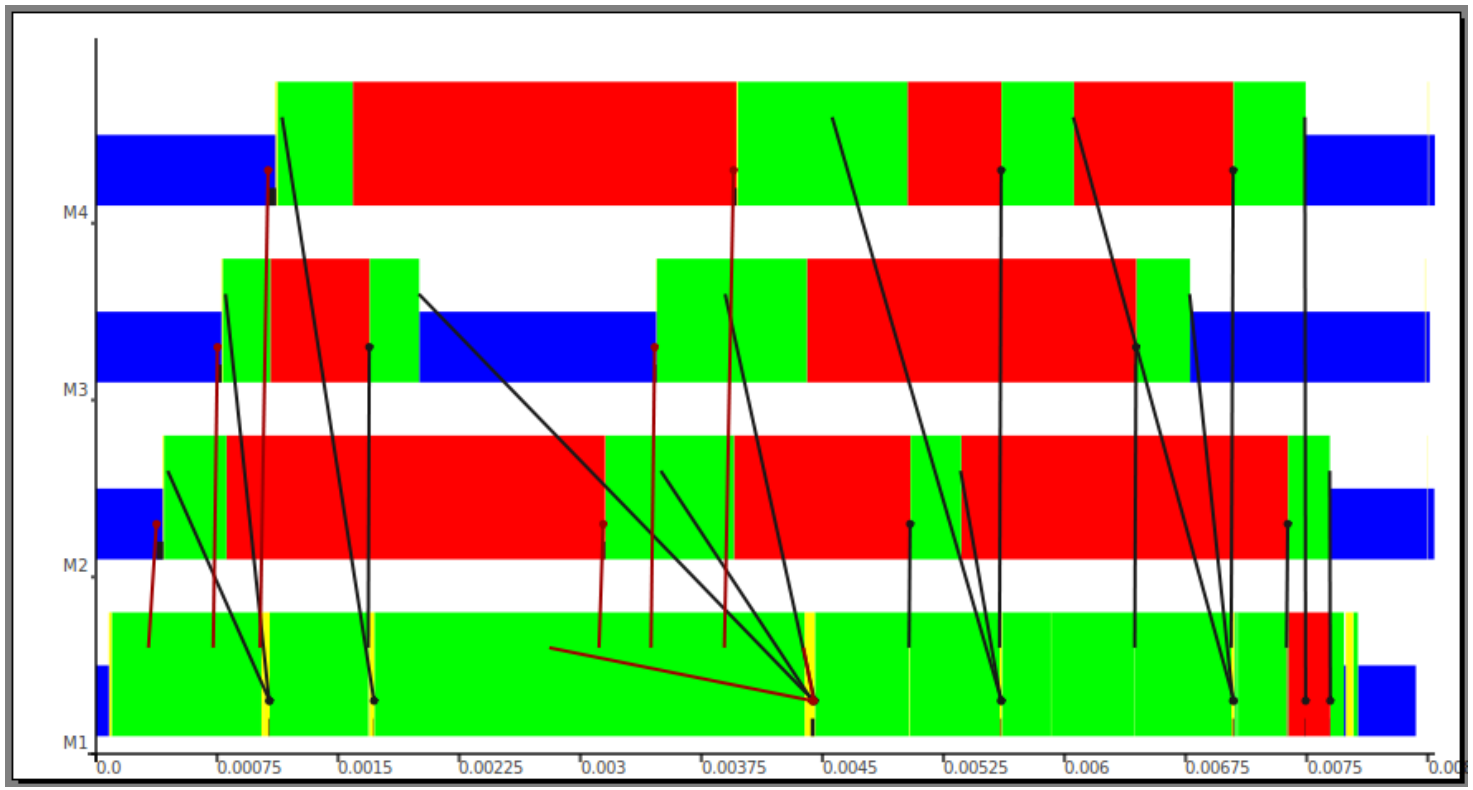
2. What is part of the MA thesis?
   - Improvements
   - Futures
   - Topology Skeletons
   - Benchmarks
   - Experiment: Cloud Haskell backend

The thesis includes everything from the project, but with rewritten/refactored:

- mapping skeletons
- GpH backend
- Par Monad backend

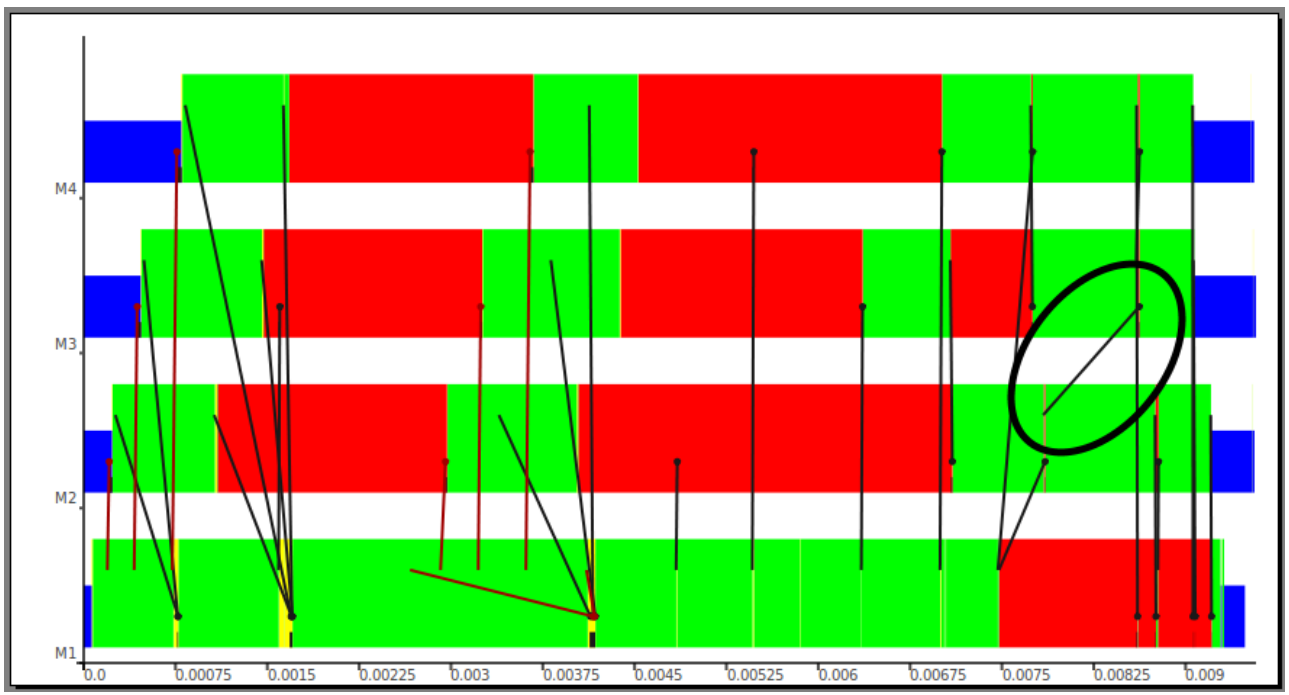
without Futures

```
1 someCombinator :: [arr a b] -> [arr b c] -> arr [a] [c]
2 someCombinator fs1 fs2 =
3   parEvalN () fs1 >>> rightRotate >>> parEvalN () fs2
```



with Futures

```
1 someCombinator :: [arr a b] -> [arr b c] -> arr [a] [c]
2 someCombinator fs1 fs2 =
3     parEvalN () (map (>>> put ()) fs1) >>>
4     rightRotate >>>
5     parEvalN () (map (get () >>>) fs2)
```

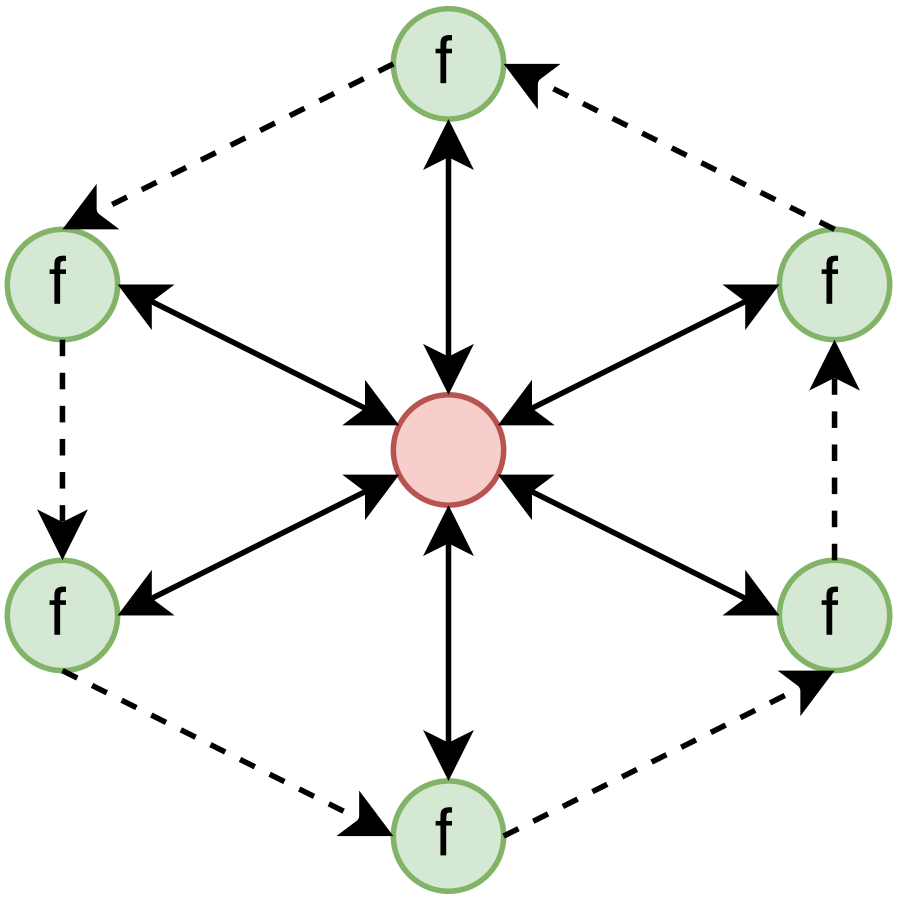


More Skeletons

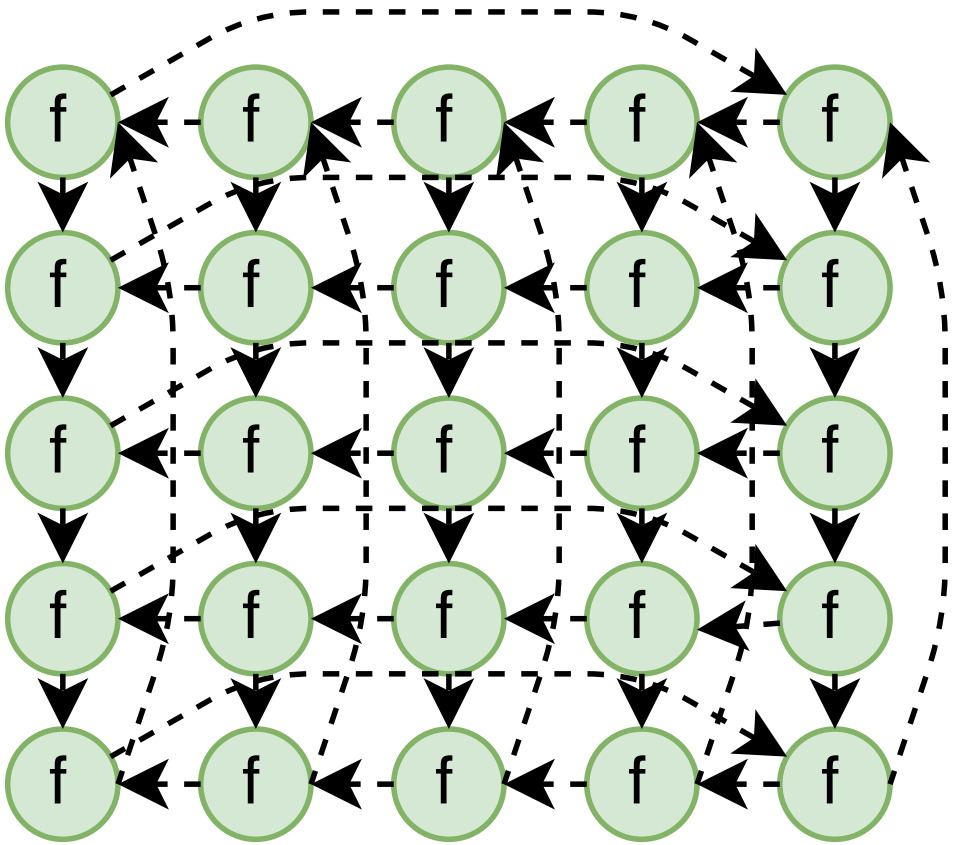
With Futures, we can implement more skeletons:

- pipe
- parallel composition |>>>|
- ring
- torus

Ring



Torus



Better Benchmarks

The thesis will also include better benchmarks for GpH, the Par Monad and Eden:

- Sudoku solver
- parallel Matrix Multiplication (Gentleman)
- Rabin Miller prime test
- Jacobi prime test

Properly executed on a 16 node Beowulf Cluster and statistically analyzed.

Cloud Haskell = Haskell in the Cloud with dynamic node discovery, hosting on AWS/Azure etc. In the thesis we will:

- explore basic parallelism with Cloud Haskell
- implement its `ArrowParallel` instance
- layout further steps of development (Futures, etc.)