University of St Andrews

SEVENTH FRAMEWORK PROGRAMME

Skel: A Streaming Process-based Skeleton Library for Erlang

Archibald Elliott[1] Christopher Brown[1] Marco Danelutto[2]
Kevin Hammond[1]

Email: ashe@st-andrews.ac.uk

[1]School of Computer Science, University of St Andrews, Scotland, UK.

[2]Dept. Computer Science, University of Pisa, Pisa, Italy.

IFL 2012 - Oxford

PARAPHRASE

This Talk

- Why we need Parallelism
- Skeletons are good Abstractions
- Skeletons in Erlang
- `skel`'s good Speedups

The Vision

1. The single-core processor is almost completely obsolete
2. Hardware systems are rapidly moving towards many- and mega-core

   > *By 2019 there will be millions of cores in home desktop machines* – Joe Armstrong

3. Software systems are still not ready:
   - Programming languages have not caught up
   - Software practices have not caught up
   - Programmers have not caught up
4. We need to make programming parallel systems *easy*

Race Conditions

What happens when you use Pthreads

Current Approaches

- Most Programmers are taught to program sequentially
- Modifying sequential code will not scale
- Typical concurrency techniques will not scale
- Fundamentally, current approaches are too low-level:
  - You can't program effectively while thinking about deadlocks, race conditions, synchronisation, non-determinism etc.
  - You can't program effectively directly with threads, message passing, mutexes or shared memory.
  - You can only program effectively with a different mindset

Our Approach

- We need to provide a set of abstractions for the programmer;
- They need to become second-nature;
- They need to make it easy to introduce parallelism;
- They need to make it easy to tune parallelism to gain maximum speedup;

Functional Programming

Functional programming can provide the correct abstractions

However, languages take fundamentally different approaches

- Haskell (GpH) is too implicit:
  - `par :: a -> b -> b`
  - `pseq :: a -> b -> b`
- Erlang is too explicit:
  - `spawn`
  - `!` and `receive`

Our Approach

Parallel Pattern

A reusable way of parallelising a computation.

Algorithmic Skeleton

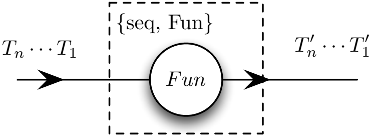
An implementation of a Parallel Pattern.

API

```
skel:run(Skeleton, InputItems).
% -> OutputItems
```

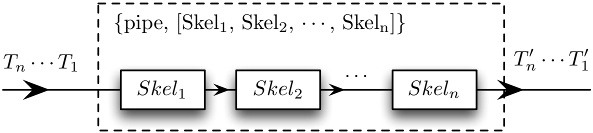
- `Skeleton` – a skeleton
- `InputItems` – items to be processed
- `OutputItems` – items that have been processed

Seq



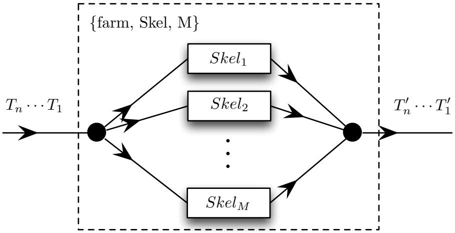
```
skel:run({seq, fun (X) -> X+1 end},
         [1,2,3,4,5,6]).
% -> [2,3,4,5,6,7]
```

Pipe



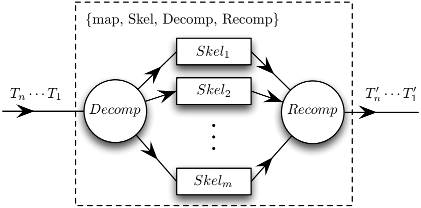
```
Inc    = {seq, fun (X) -> X+1 end},
Double = {seq, fun (X) -> X*2 end},
skel:run({pipe, [Inc, Double]},
         [1,2,3,4,5,6]).
% -> [4,6,8,10,12,14]
```

Farm



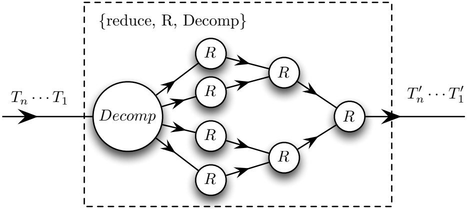
```
Inc = {seq, fun(X)-> X+1 end},
skel:run({farm, Inc, 3},
              [1,2,3,4,5,6]).
% -> [2,5,3,6,4,7]
```

Map



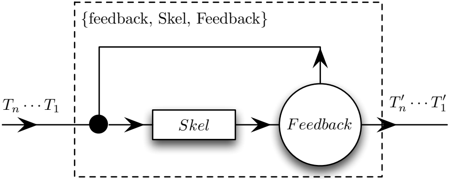
```
Inc = {seq, fun(X)-> X+1 end},
skel:run({map, Inc,
               fun erlang:tuple_to_list/1,
               fun erlang:list_to_tuple/1},
         [{1,2},{3,4}]).
% -> [{2,3},{4,5}]
```

Reduce



```
skel:run({reduce, fun(X,Y) -> X + Y end,
                  fun erlang:tuple_to_list/1},
         [{1,2,3,4,5,6},{7,8,9,10,11,12}]).
% -> [21,57]
```

Feedback

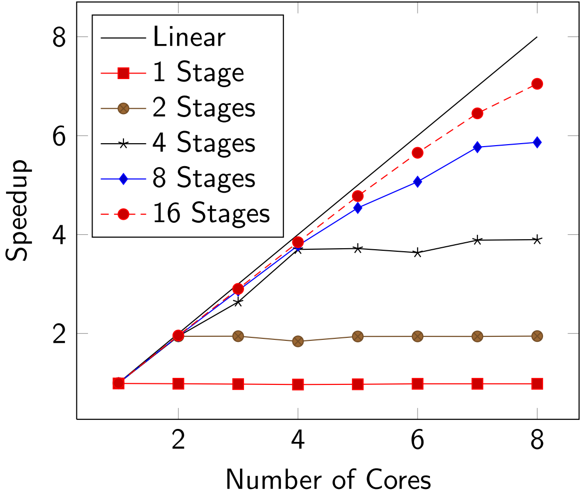


```
Inc = {seq, fun(X) -> X+1 end},
skel:run({feedback, Inc,
                    fun(X) -> X < 5 end},
         [1,2,3,4,5,6,7,8,9,10]).
% -> [5,6,7,8,9,10,11,5,5,5]
```

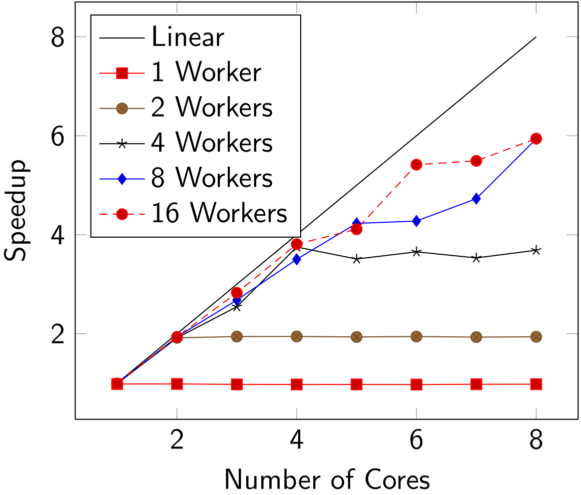
Experiments

- Regular task cost: 1ms
- Task cost > communication cost
- Constant input number: 100,000
- 8 Cores: 2x Intel Xeon 4-Core X5355 2.66GHz
- Erlang R15B01

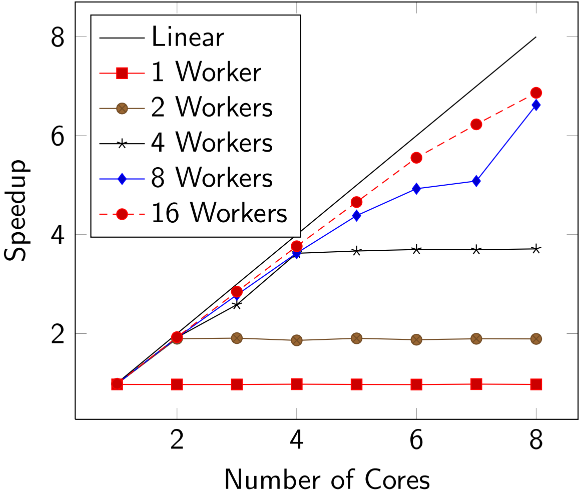
Results: Pipe



Results: Farm

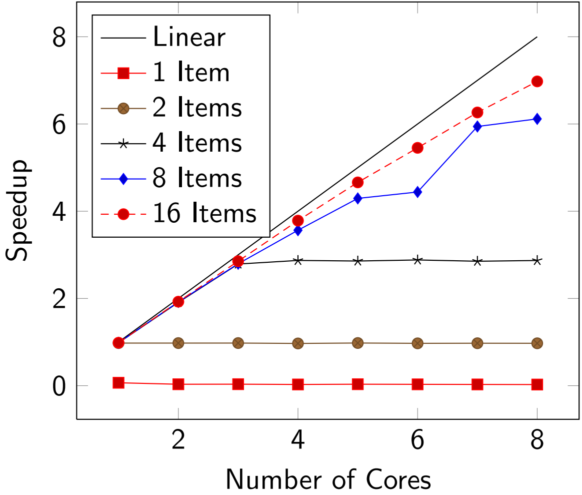


Results: Map



Linear
1 Worker
2 Workers
4 Workers
8 Workers
16 Workers
Speedup
8
6
4
2
2
4
6
8
Number of Cores

Results: Reduce



Conclusions

- Adopting a Parallel Mindset
  - Functional Programming
  - Algorithmic Skeletons
- Erlang is a great fit.
  - Low-level enough for control
  - High-level enough to allow abstraction
- `skel`'s speedups prove our implementation is good

Future Work

- More Benchmarks
- Better Speedups
- Higher-order Skeletons
  - Divide and Conquer
  - MapReduce
  - Genetic Algorithms
  - ...
  - Domain-specific Algorithms

THANK YOU

http://www.paraphrase-ict.eu

@paraphrase_fp7