# Automatic Parallelisation in Mercury

Paul Bone pbone@csse.unimelb.edu.au

January 19, 2010

NICTA

THE UNIVERSITY OF MELBOURNE

# Mercury

- Mercury is a declarative, *pure* language.
- Purity makes programming more reliable.
- Purity also makes it easier for the compiler to optimise code, including automatic parallel evaluation.

- Over 15 years old, and has been self-hosted for most of this time.
- The compiler has 425,674 LoC, excluding the standard library and runtime, yet our daily snapshots are usually stable!
- Can compile to C, Java, Erlang and MS IL.
- Named after the Roman god of speed.

# The problem

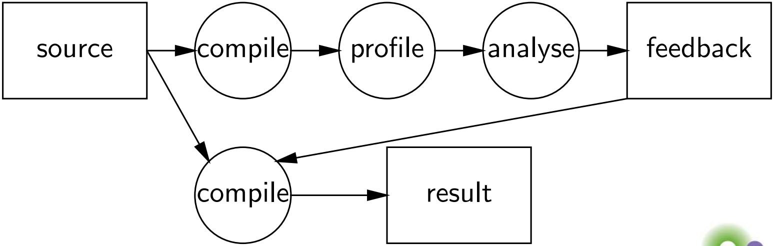
Parallel programming is hard, but multicore systems are ubiquitous.

- Thread synchronisation is very hard, but *purity* makes this a non-issue.
- Working out *how* to parallelise a program can be difficult.
- What if the program changes in the future? The programmer may have to re-parallelise it.

This makes parallel programming time consuming and expensive. Yet in a multicore era it is desirable to parallelise most programs.

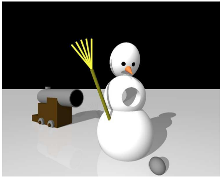
# Automatically Parallelising a program

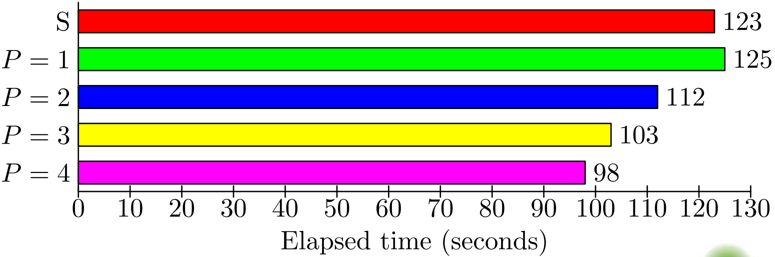
- Profile the program to find the expensive parts.
- Analyse the program to determine what can be run in parallel.
- Determine if it is profitable to introduce parallel evaluation. This may involve trial and error.
- Repeat until the program runs fast enough or there is nothing left to parallelise.



# Benchmarks — ICFP 2000 Raytracer

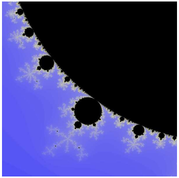
- Heavy garbage collector usage
- 6,199 LoC.
- Code was altered to make it less stateful.

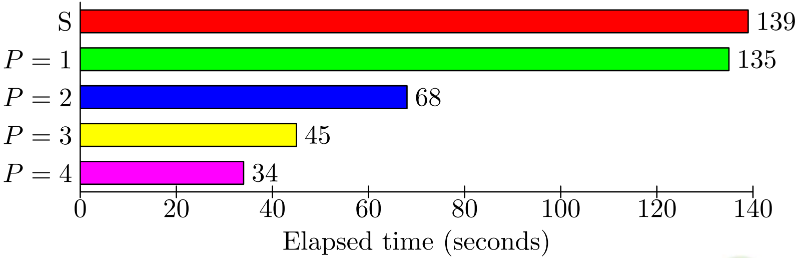




# Benchmarks — Mandelbrot image generator

- Light garbage collector usage
- 280 LoC.
- Written for this test.



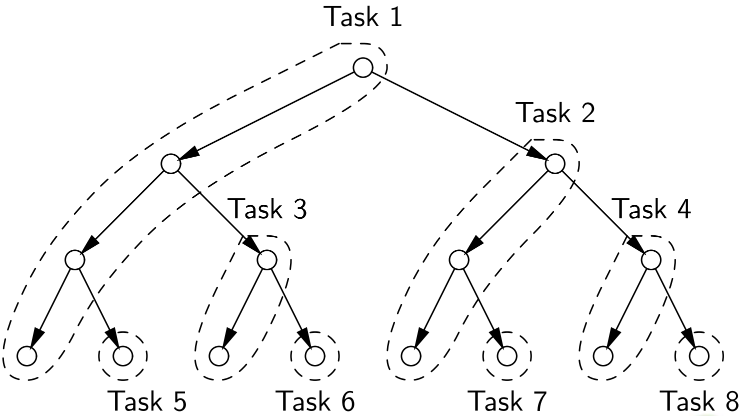


# Trickier cases — Divide and Conquer

```
quicksort([]) = [].
quicksort([ P | Unsorted ]) = Sorted :-
  (Bigs, Littles) = partition(P, Unsorted),
  (
    SortedBigs = quicksort(Bigs) &
    SortedLittles = quicksort(Littles)
  ),
  Sorted = SortedLittles ++ [ P | SortedBigs ].
```

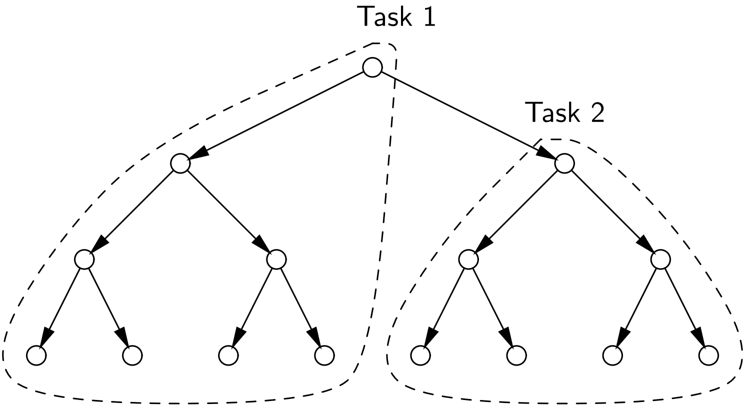
# Trickier cases — Divide and Conquer

On average, this creates $O(N)$ small parallel tasks. This is far too many since most systems have far fewer than $N$ cores.



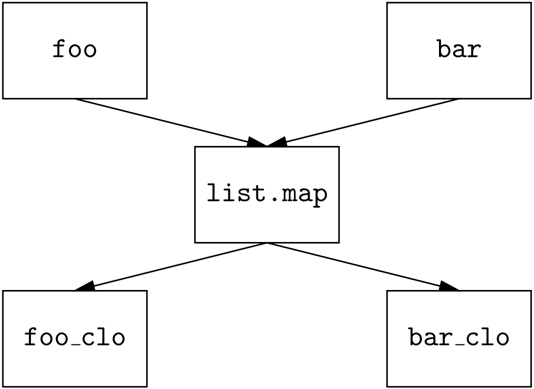
# Trickier cases — Divide and Conquer

It is much better to parallelise the first $O(log_2 P)$ levels of the tree.



# Tricker cases — Specialisation

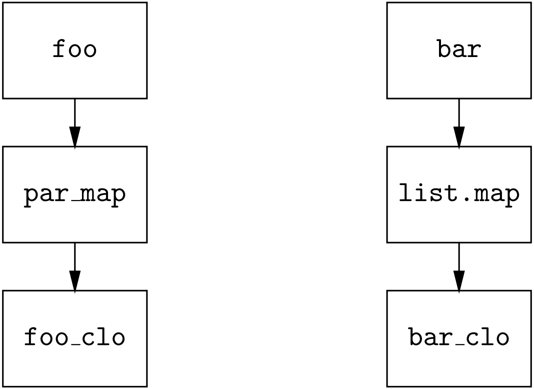
`foo_clo` is expensive and we can parallelise `list.map` to speed up `foo`. But `bar_clo` is simple and fast, parallelising `list.map` would slow it down.



# Tricker cases — Specialisation

Make a copy of `list.map` and parallelise that, re-write `foo` so it calls the new copy of `list.map`.

Our profiler can collect the necessary information to make these decisions.



# Conclusion

- Parallel garbage collection is an active research area.
- Many other optimisations are being developed to make automatic parallelisation useful for a wider range of programs.
- Pure, declarative languages make parallelism easier.
- Automatic parallelisation will make it easy for developers to take advantage of multicore systems.

# Questions?

Questions?