Hume and Multicore Architectures

Kevin Hammond, Roy Dyckhoff,
Pedro Vasconcelos, Meng Sun, Leonid Timochouk,
Edwin Brady, Steffen Jost, Armelle Bonenfant
University of St Andrews, Scotland

Greg Michaelson, Andy Wallace,
Robert Pointon, Graeme McHale, Chunxiu Liu, Gudmund Grov, Zenzi Chen
Heriot-Watt University, Scotland

Jocelyn Sérot, Norman Scaife
LASMEA, Clermont-Ferrand, France

Martin Hofmann, Hans-Wolfgang Loidl
Ludwig-Maximilians Universität, München, Germany

Christian Ferdinand, Reinhold Heckmann
AbsInt GmbH, Saarbrücken, Germany

`http://www.hume-lang.org`
`http://www.embounded.org`

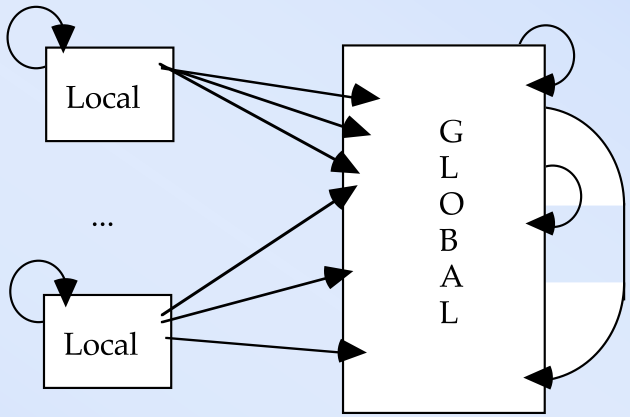
Background: Glasgow Parallel Haskell

- **Glasgow Parallel Haskell (GpH)**
  - Parallel Functional Programming Language
  - built on good sequential compiler (GHC - Glasgow Haskell Compiler)
  - Semi-explicit parallelism - minimal modification (par introduces threads)
  - purely functional = no artificial limits on thread introduction
  - Message passing implementation (mapped to cache on SMP)
  - low parallel overheads

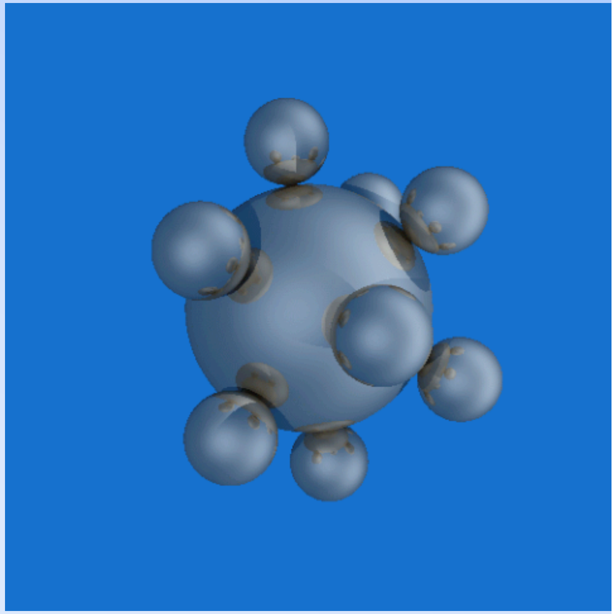
with Simon Peyton Jones, Jim Mattson, Phil Trinder etc

- **Large-scale multithreading**
  - Evaluation strategies to structure parallelism and control threads
    - » good for irregular parallelism (control parallel apps.)
  - Implicit threading
  - Automatic throttling where needed (evaluate-and-die)
  - task stealing approach

- **2-level heap structure**
  - independent memory
  - parallel GC



Example: Ray Tracer



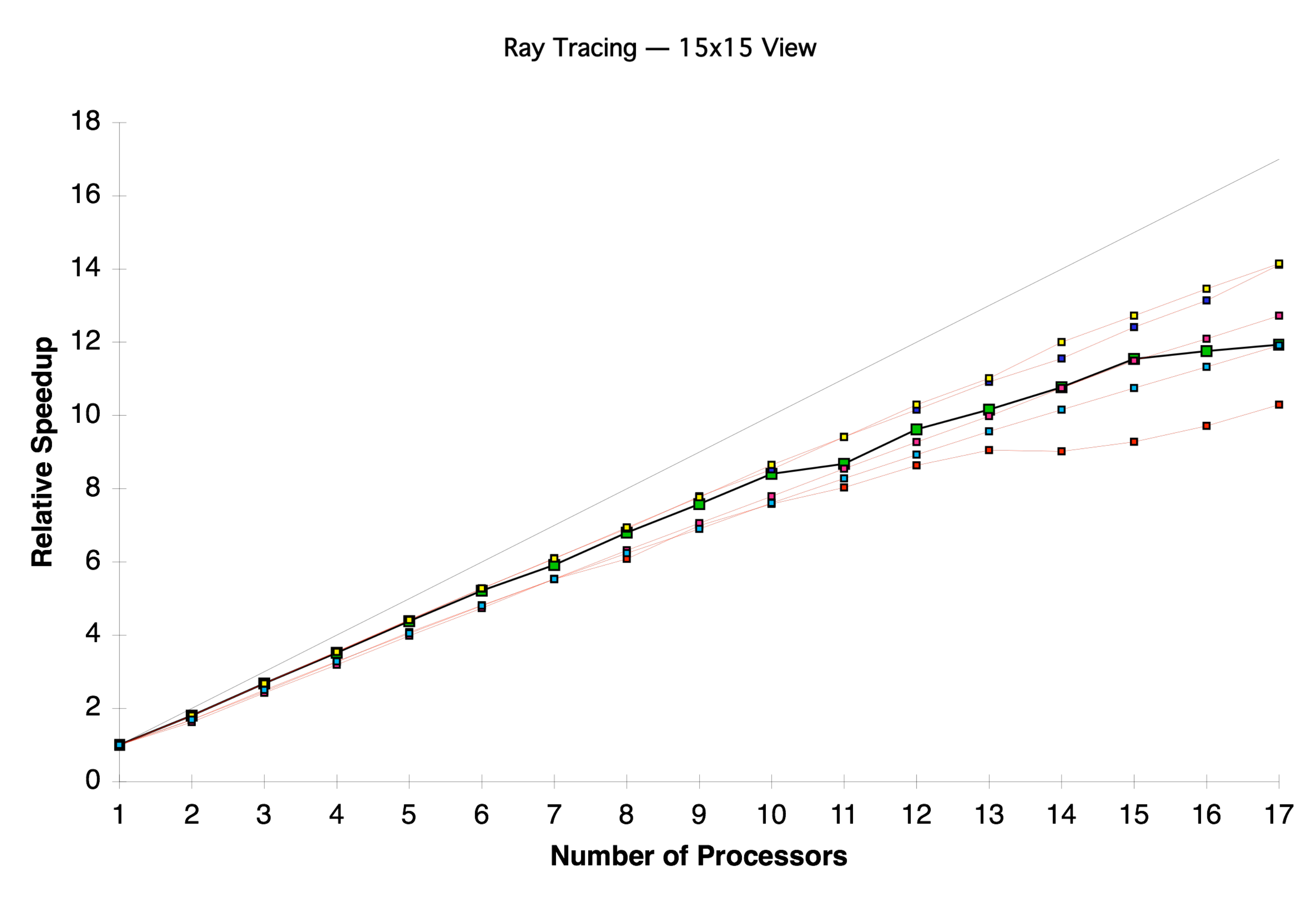
- Maps individual ray tracing function `(trace)` over all pixels in the view.

```
ray :: Int -> [[(Int,Int), Vector]]
ray size = map f1 coords
 where f1 i = map (f2 i) coords
       f2 i j = ((i,j), trace i j)
       trace = ...; coords = [1..size]
```

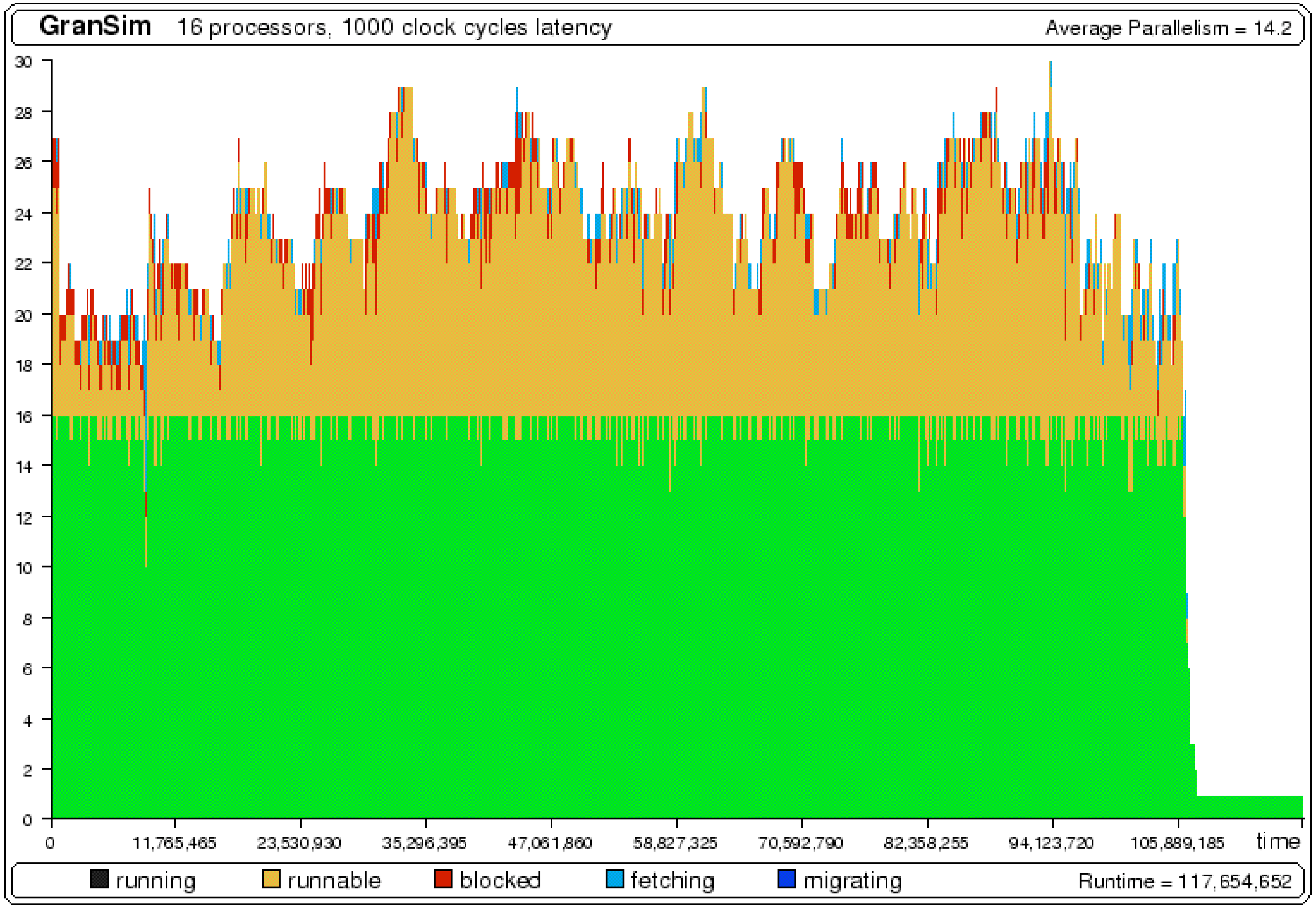
- Parallelism is introduced by adding the `parallel all` strategy on lists.

```
ray size = map f1 coords                 `using` parallel all
 where f1 i = map (f2 i) coords          `using` parallel all
       f2 i j = ((i,j), trace i j)
```

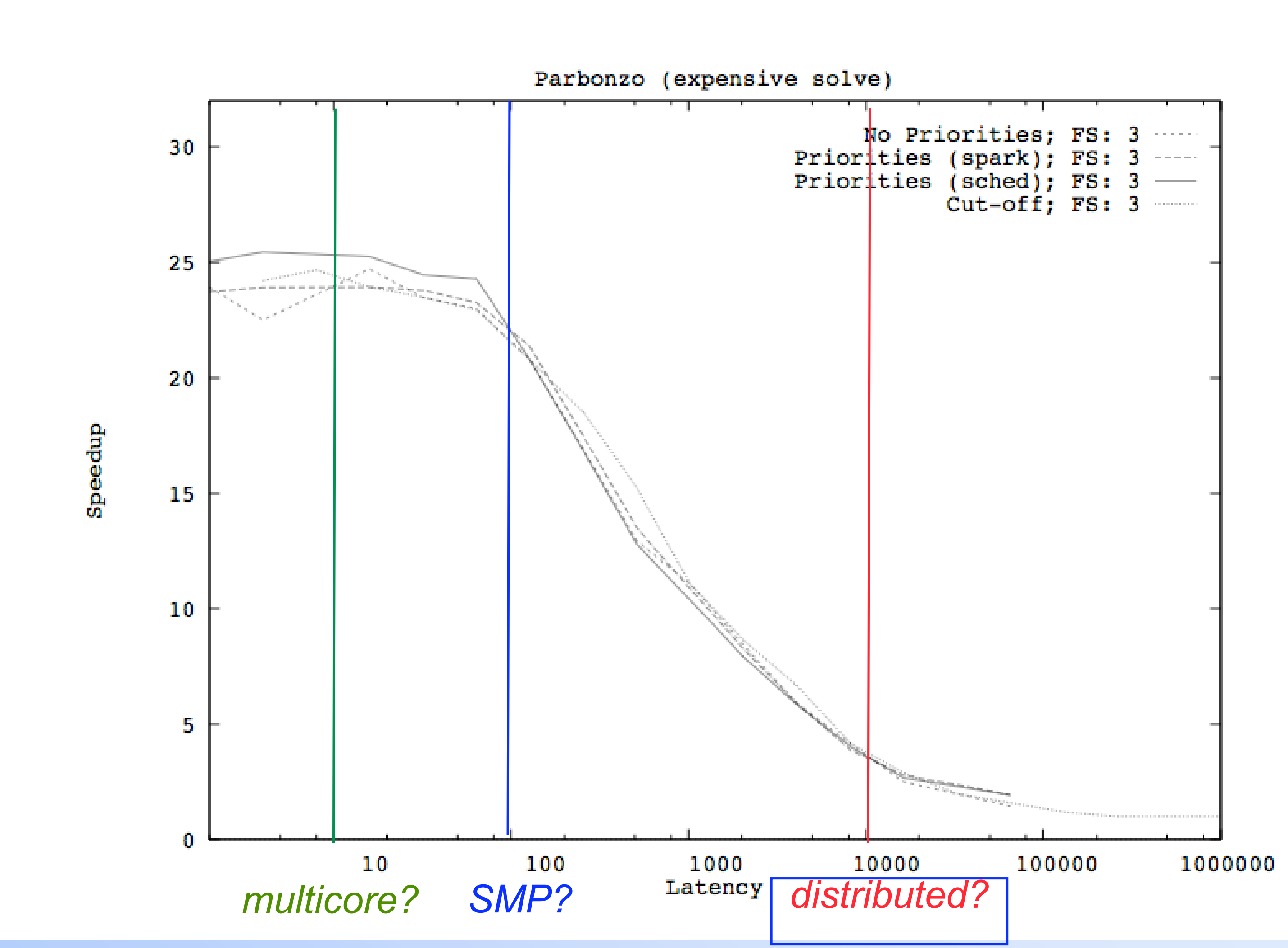
Example Speedup Graph



Simulation Activity Profile



Speedup v Comms. Latency (Simulated)



Irregular Applications

- Lolita Natural Language Parser — 47,000 lines
- Naira Compiler — 6,000 lines
- Ray Tracing — 1,500 lines
- Accident Blackspots — 1,000 lines
- Particle Simulation — 800 lines
- Linear Equation Solver — 800 lines

plus many smaller examples

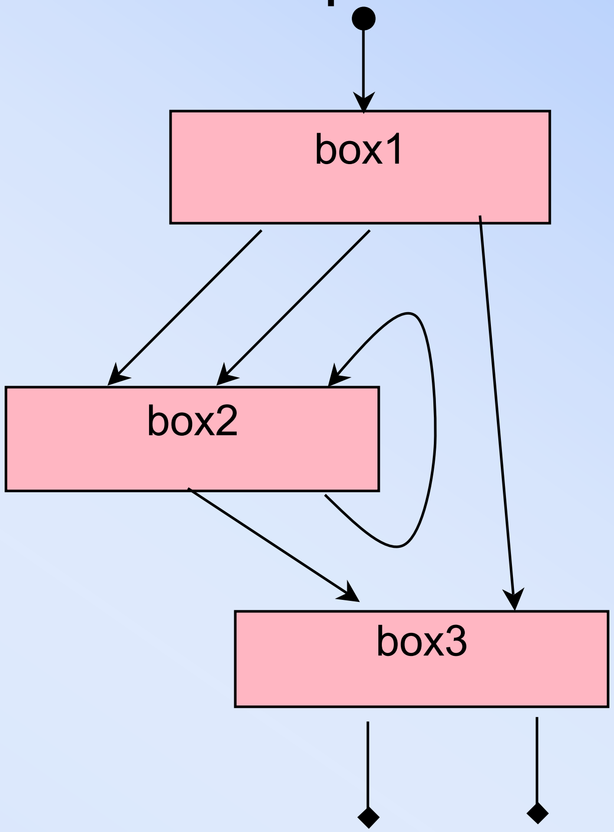
Hume Research Objectives

- **Virtual Testbed for Space/Time/Power Cost Modelling**
  - targetting Embedded Systems
- **Real-Time, Hard Space High-Level Programming**
  - Based on Functional Programming and Finite Automata
- **Concurrent Multithreaded Design**
  - Asynchronous threading

Hume Language Structure

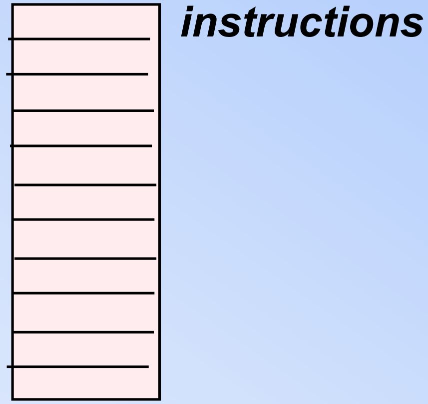
- ***Boxes*** **structure processes**
  - **Implicitly parallel, but clearly identification of tasks**
  - ***Asynchronous*** **communication**
  - ***Stateless*** **automata**

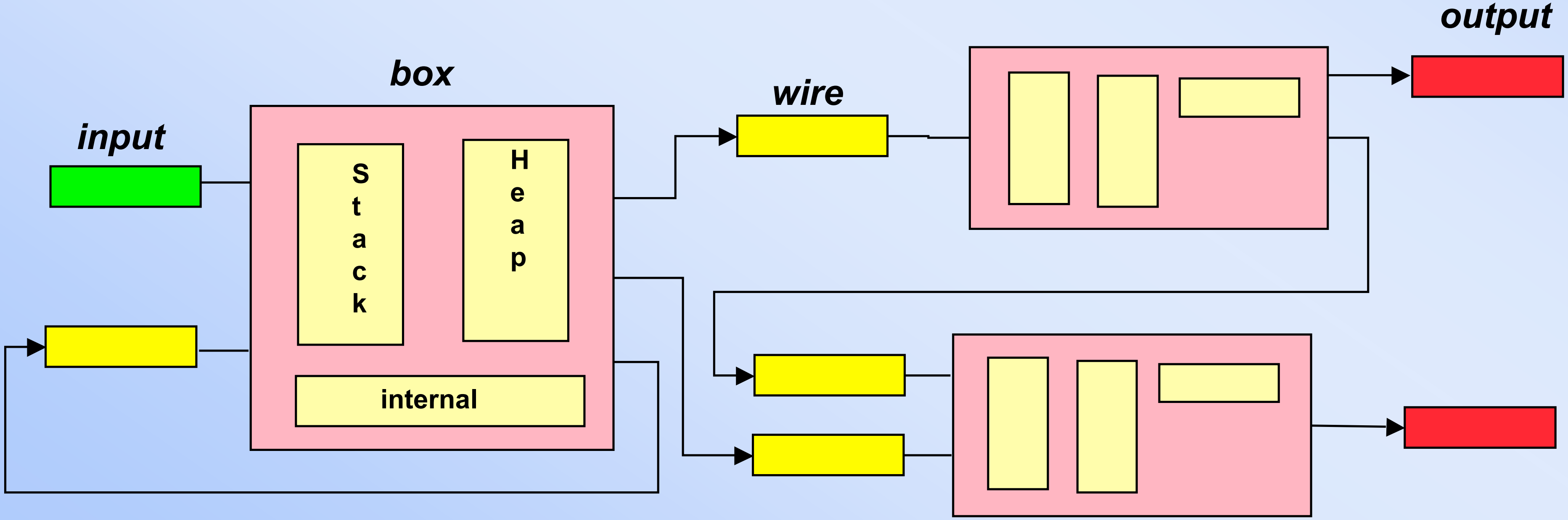
- ***Functions*** **structure computations**
  - **Purely functional notation (based on Haskell)**
  - **Pattern-matching relates inputs to outputs through functional expressions**
  - ***No communication during thread execution***
    - » **fire, match, execute, write**
  - **Strict evaluation**



Hume Implementation

- One thread per box
- Independent thread stack/heap
  - No GC necesary for short threads
- Fixed-Size Wire Buffers (shared mem.)
- Shared Instruction Stream (multi possible)





Hume and Multicore

- Boxes can be mapped to different cores
  - Concurrency model supports multithread scheduling
  - Asynchronous threading model
- Each box runs as a thread up to communication
  - Efficient execution by one core
  - No inter-core interaction except at communication points
  - Thread interaction can be predicted => more efficient scheduling
  - Threads can be further decomposed to microthreads
- Exceptions happen at box level
  - Single handler for all thread exceptions
  - Efficient handling
- Handles real-time, real-space restrictions
  - Highly accurate space cost estimates

Hume and Multicore (2)

- **Hardware/software co-design notation?**
  - Different computation levels can be used
    - » HW-Hume - close match to hardware
    - » FSM-Hume - more programming power
    - » Template-Hume - Higher-order patterns to structure computations
    - » Full-Hume - fully featured language

- **Time, Space and ?Power? consumption can be predicted**
  - Source based approach
    - » Loop bounds, conditionals, worst-case or probabilistic
  - Combined with static analysis of computer architecture
    - » (accurate low-level, worst-case behaviour - AbsInt GmbH, Germany)

Conclusions

- High-Level Notation for Concurrent Programming
  - lightweight threading: high degree of parallelism
  - minimise communication/synchronisation
  - locking points explicitly identified (and minimal)
  - *independent* memory
  - good sequential code within thread
  - fast scheduling (based on available inputs)
  - 2-level structure allows focus on different properties
- Research focus on hard real-time, but this can help with multicore
  - natural concurrency
  - boxes can be arbitrarily replicated
  - controlled communication
  - per-thread cache requirements easily identified

Wish List for Multithreading

- **What hardware support would be useful**
  - several (fast) cores
  - non-uniform caches
  - lock support on part of the memory (cache coherence)
  - *but* most cache needs to be fast, coherence isn't an issue
  - memory allocation support (allocation hints, in-cache allocation)?
  - thread creation support
  - thread placement hints (to improve spatial locality)
  - scheduling support (thread pools)

Current Projects

28 person years in total

- **EmBounded: €1.3M (5 EU sites)**
  - Develop and robustify Hume
  - Enhance cost models and analyses
  - Provide resource certification
  - Develop substantial real-time applications (control and computer vision)

EMBOUNDED

- **Defence technology Consortium: £297K (part of a £4M overall project)**
  - Apply Hume to Control Systems for Autonomous Vehicles
  - ?Extend to mobile sensor networks?
  - Industrial project coordinated by BAe Systems

BAE SYSTEMS

- **Generative Programming for Embedded Systems: £145K**
  - Allow reasoning about resource usage in multi-stage compilers

EPSRC

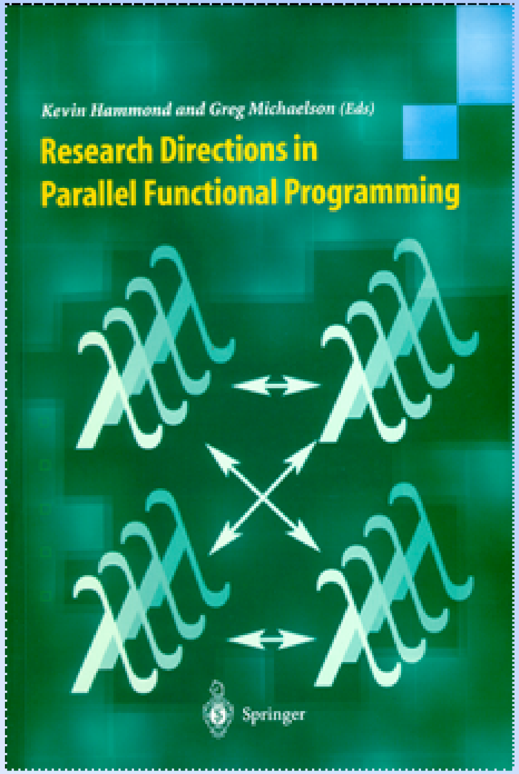
- **Symbolic Computing for Commodity Parallel Machines: £153K**
  - Adapt symbolic computing algs. to stock architectures (e.g. multicore)

EPSRC

K. Hammond
and G. Michaelson (eds.)

Research Directions in Parallel Functional Programming

Springer, October 1999,
ISBN 1-85233-092-9, 520pp, £60



Chapters on:

Design, Implementation, Parallel Paradigms, Proof, Cost Modelling, Performance Evaluation, Applications

`http://www-fp.dcs.st-and.ac.uk/pfpbook`

http://www.hume-lang.org

Some Recent Papers

Is it Time for Real-Time Functional Programming?

Kevin Hammond

Trends in Functional Programming 4, 2005, pp. 1-12.

Inferring Costs for Recursive, Polymorphic and Higher-Order Functional Programs

Pedro Vasconcelos and Kevin Hammond

Proc. 2003 Intl. Workshop on Implementation of Functional Languages (IFL '03), Edinburgh,

Springer-Verlag LNCS, 2004. *Winner of the Peter Landin Prize for best paper*

Hume: A Domain-Specific Language for Real-Time Embedded Systems

Kevin Hammond and Greg Michaelson

Proc. 2003 Conf. on Generative Programming and Component Engineering (GPCE 2003), Erfurt, Germany,

Springer-Verlag LNCS, Sept. 2003. *Proposed for ACM TOSEM Fast Track Submission*

FSM-Hume: Programming Resource-Limited Systems using Bounded Automata

Greg Michaelson, Kevin Hammond and Jocelyn Sérot

Proc. 2004 ACM Symp. on Applied Computing (SAC '04), Nicosia, Cyprus, March 2004

The Design of Hume

Kevin Hammond

Invited chapter in *Domain-Specific Program Generation*,

Springer-Verlag LNCS State-of-the-art Survey, C. Lengauer (ed.), 2004

Predictable Space Behaviour in FSM-Hume,

Kevin Hammond and Greg Michaelson,
Proc. 2002 Intl. Workshop on Implementation of Functional Languages (IFL '02), Madrid, Spain, Sept. 2002,

Springer-Verlag LNCS 2670, ISBN 3-540-40190-3,, 2003, pp. 1-16