Thinking Parallel: Generating Parallel Erlang Programs from High-Level Patterns

Kevin Hammond

University of St Andrews, Scotland

Invited Talk at goto; Conference, Zurich, April 2013

T: ***@paraphrase_fp7***

E: ***kh@cs.st-andrews.ac.uk***

W: http://www.paraphrase-ict.eu

The Present

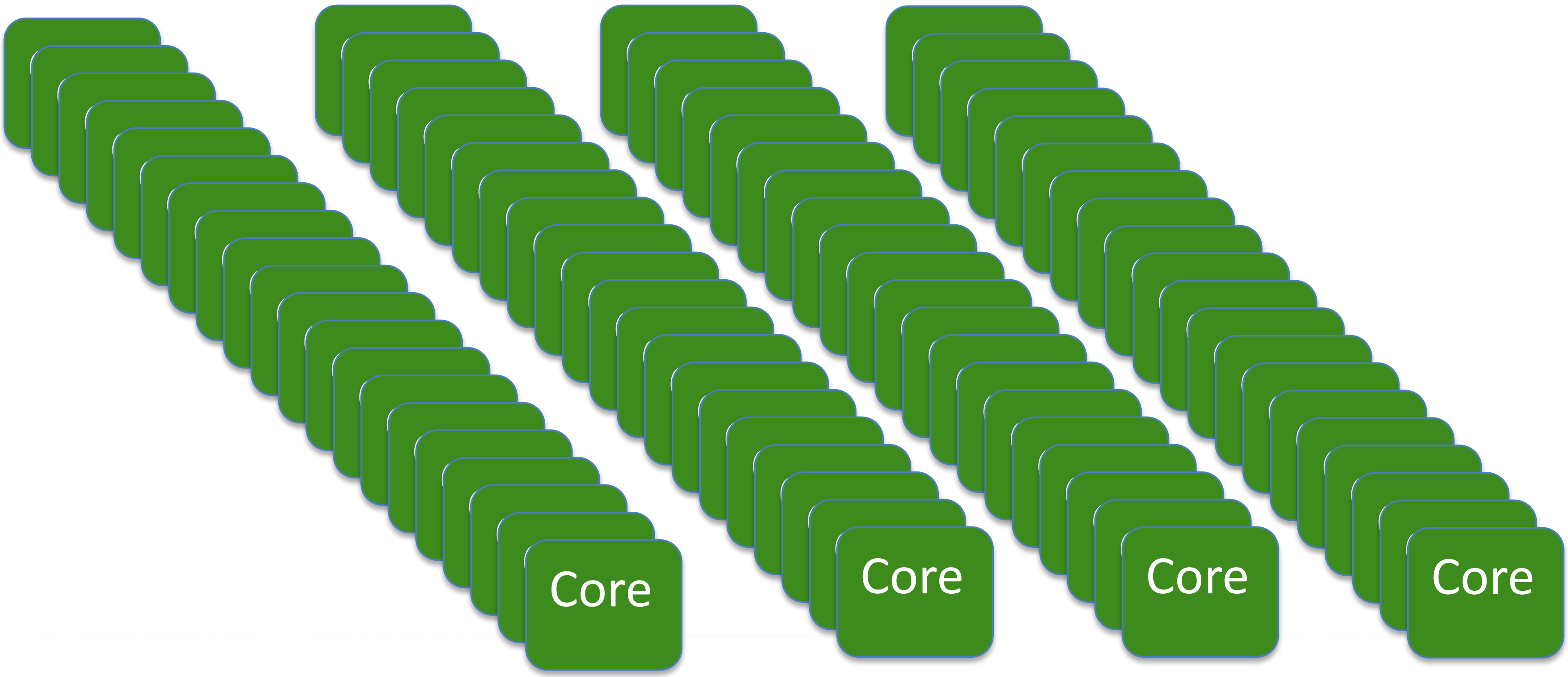
Pound versus Dollar

2013: a ManyCore Odyssey

The Future: “megacore” computers?

University of St Andrews

- ***Hundreds of thousands, or millions, of (small) cores***



Laki (NEC Nehalem Cluster) and hermit (XE6)

Laki

- 700 dual socket Xeon 5560 2,8GHz ("Gainestown")
- 12 GB DDR3 RAM / node
- Infiniband (QDR)
- 32 nodes with additional Nvidia Tesla S1070
- Scientific Linux 6.0

hermit (phase 1 step 1)

- 38 racks with 96 nodes each
- 96 service nodes and 3552 compute nodes
- Each compute node will have 2 sockets AMD Interlagos @ 2.3GHz 16 Cores each leading to 113.664 cores
- Nodes with 32GB and 64GB memory reflecting different user needs
- 2.7PB storage capacity @ 150GB/s IO bandwidth
- External Access Nodes, Pre- & Postprocessing Nodes, Remote Visualization Nodes

The Manycore Challenge

University of St Andrews

"Ultimately, developers should start thinking about *tens, hundreds, and thousands* of cores *now* in their algorithmic development and deployment pipeline."

Anwar Ghuloum, Principal Engineer, Intel Microprocessor Technology Lab

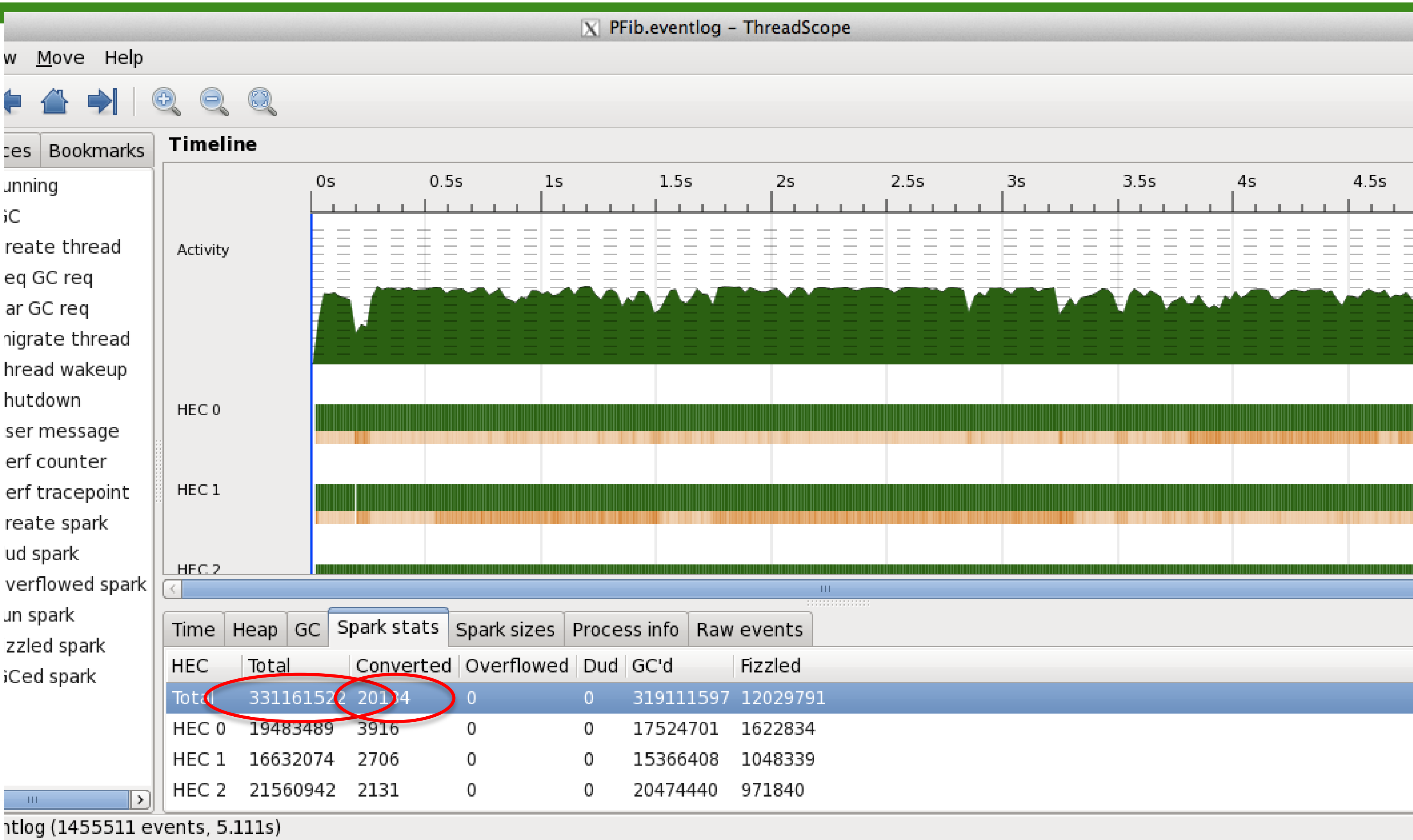
The **ONLY** important challenge in Computer Science

Intel

"The dilemma is that a *large percentage* of mission-critical enterprise applications will not ``automagically'' run faster on multi-core servers. *In fact, many will actually run slower.* We must make it as easy as possible for applications programmers to exploit the latest developments in multi-core/many-core architectures, while still making it easy to target future (and perhaps unanticipated) hardware developments."

Patrick Leonard, Vice President for Product Development
Rogue Wave Software

Doesn't that mean millions of threads on a megacore machine??



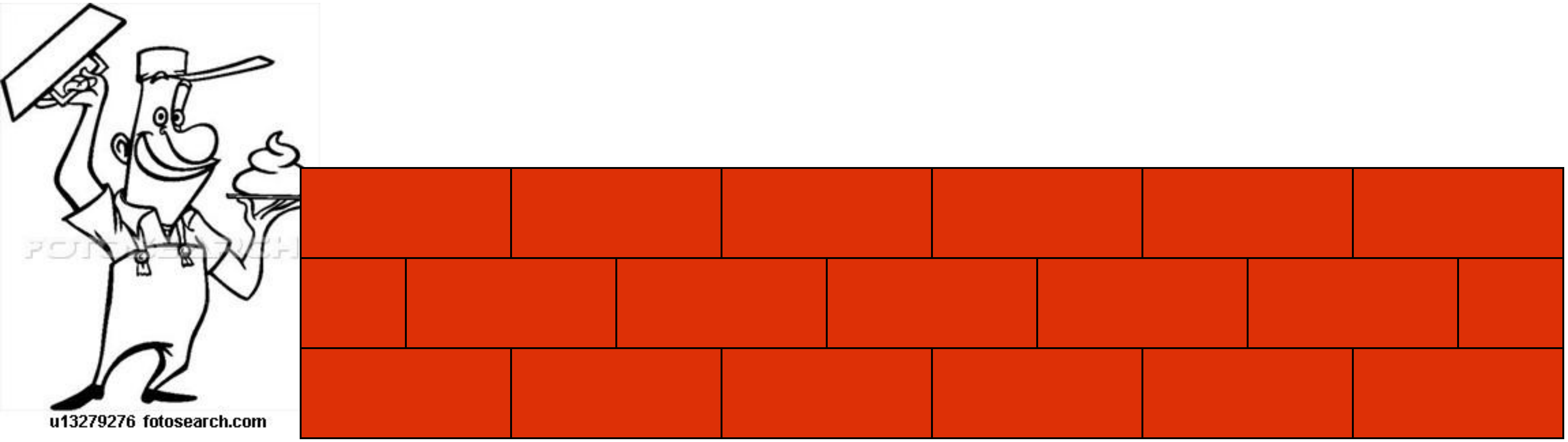
HEC	Total	Converted	Overflowed	Dud	GC'd	Fizzled
Total	331161522	201[illegible]4	0	0	319111597	12029791
HEC 0	19483489	3916	0	0	17524701	1622834
HEC 1	16632074	2706	0	0	15366408	1048339
HEC 2	21560942	2131	0	0	20474440	971840

All future programming will be parallel

University of St Andrews

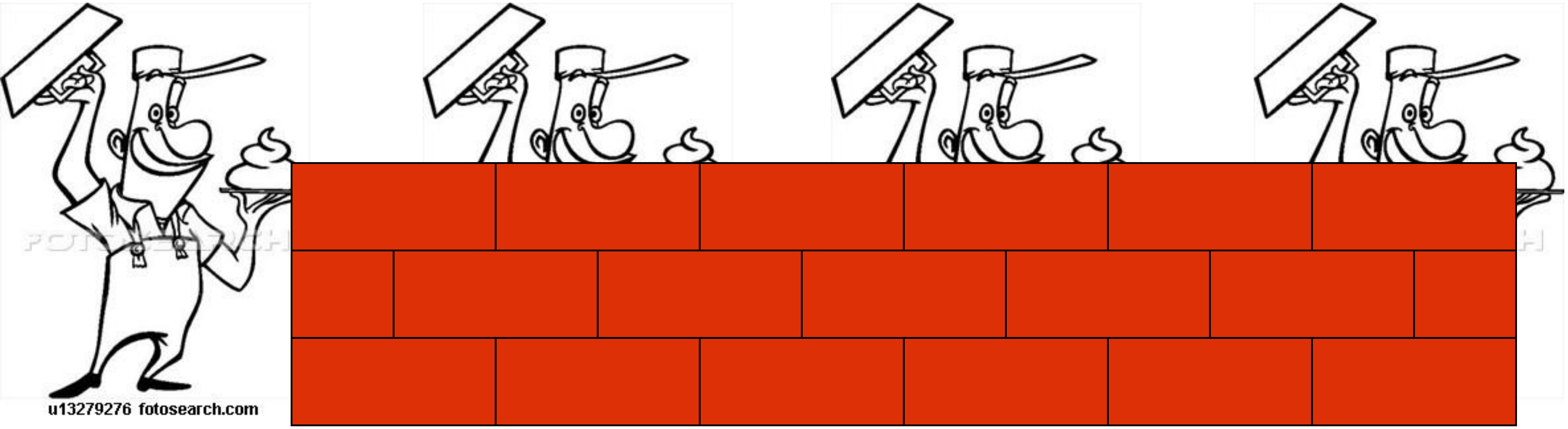
- No future system will be single-core
  - parallel programming will be essential
- It’s not just about performance
  - it’s also about energy usage
- If we don’t solve the multicore challenge, then all other CS advances won’t matter!
  - user interfaces
  - cyber-physical systems
  - robotics
  - games
  - ...

How to build a wall

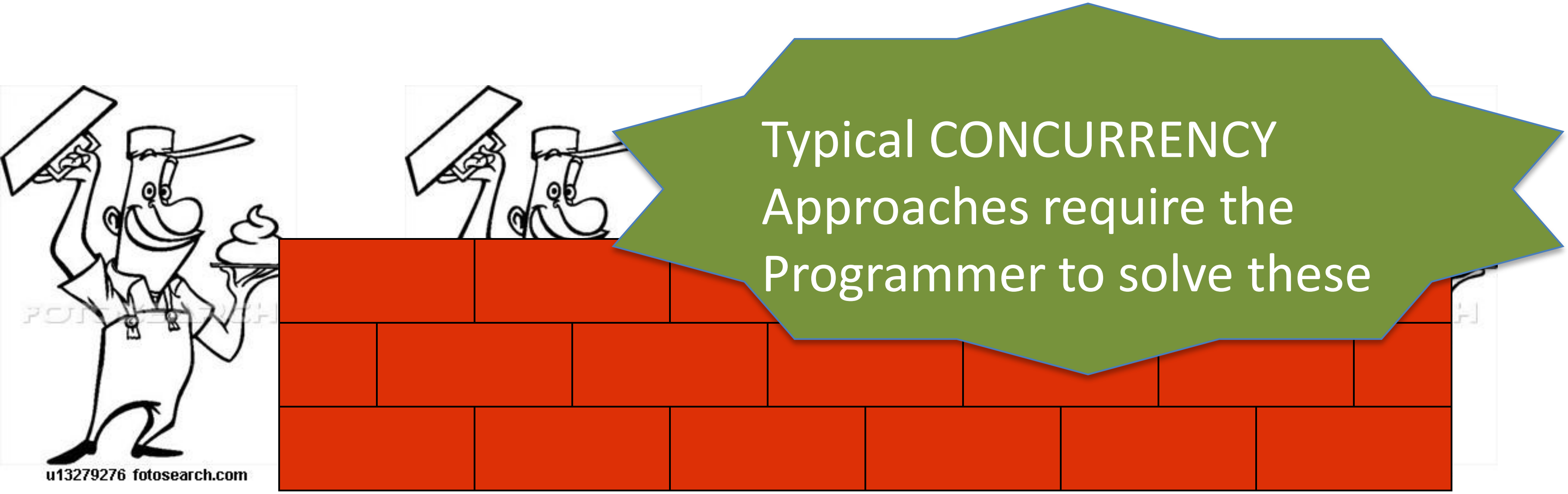


(with apologies to Ian Watson, Univ. Manchester)

How to build a wall *faster*



How NOT to build a wall



Task identification is not the only problem…

Must also consider Coordination, communication, placement, scheduling, …

We need structure
We need abstraction

We don’t need another brick in the wall

Thinking Parallel

- **Fundamentally, programmers must learn to "think parallel"**
  - this requires new *high-level* programming constructs
    - perhaps dealing with hundreds of *millions* of threads
- **You cannot program effectively while worrying about deadlocks etc.**
  - they must be eliminated from the design!
- **You cannot program effectively while fiddling with communication etc.**
  - this needs to be packaged/abstracted!
- **You cannot program effectively without performance information**
  - this needs to be included as part of the design!

A Solution?

"The only thing that works for parallelism is functional programming"

Bob Harper, Carnegie Mellon University

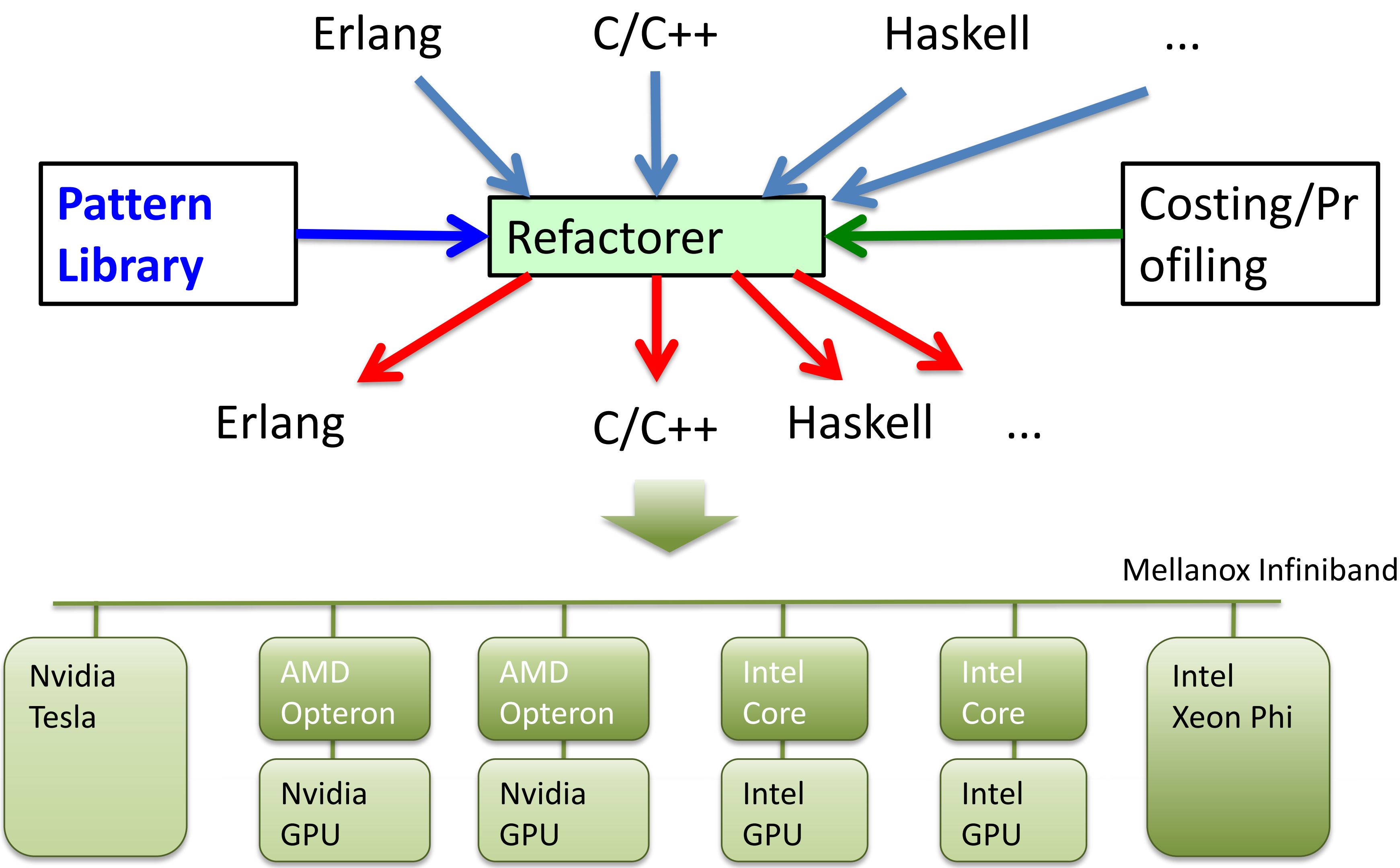
Parallel Functional Programming

- No explicit ordering of expressions
- Purity means no side-effects
  - Impossible for parallel processes to interfere with each other
  - **Can debug sequentially but run in parallel**
  - **Enormous** saving in effort
- Programmer concentrate on solving the problem
  - Not porting a sequential algorithm into a (ill-defined) parallel domain
- **No locks, deadlocks or race conditions!!**
- **Huge productivity gains!**
- Much shorter code

The ParaPhrase Approach

- Start bottom-up
  - identify (strongly hygienic) **COMPONENTS**
  - *using semi-automated refactoring*
- Think about the **PATTERN** of parallelism
  - e.g. map(reduce), task farm, parallel search, parallel completion, ...
- **STRUCTURE** the components into a parallel program
  - ***turn the patterns into concrete (skeleton) code***
  - Take performance, **energy** etc. into account (multi-objective optimisation)
  - also using refactoring
- **RESTRUCTURE** if necessary! (also using refactoring)

The ParaPhrase Approach



Example: Simple matrix multiplication

- Given two NxN matrices, A and B

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix}$$

- Their product is

$$\mathbf{AB} = \begin{pmatrix} (AB)_{11} & (AB)_{12} & \cdots & (AB)_{1p} \\ (AB)_{21} & (AB)_{22} & \cdots & (AB)_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ (AB)_{n1} & (AB)_{n2} & \cdots & (AB)_{np} \end{pmatrix}$$

where $(AB)_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}.$

Example: Simple matrix multiplication

University of St Andrews

- The sequential Erlang algorithm iterates over the rows
  - *mult (A, B)* multiplies the *rows* of A with the *columns* of B

```
mult (Rows, Cols) -> [ mult1row(R,Cols) || R <- Rows  ].

...
```

- *[mult1Row(R,Cols) || R <- Rows]* does *mult1Row(R,Cols)* with R set to each row in turn

Example: Simple matrix multiplication

- The sequential Erlang algorithm iterates over the rows
  - *mult (A, B)* multiplies the *rows* of A with the *columns* of B
  - *mult1row (R, B)* multiplies one *row* of A with all the *columns* of B

```
mult (Rows, Cols) -> [ mult1row(R,Cols) || R <- Rows  ].

mult1row (R, Cols) ->
            lists:map(fun(C) -> ... end, Cols).


...
```

- *lists:map* maps an in-place function over all the columns

Example: Simple matrix multiplication

- The sequential Erlang algorithm iterates over the rows
  - *mult (A, B)* multiplies the *rows* of A with the *columns* of B
  - *mult1row (R, B)* multiplies one *row* of A with all the *columns* of B
  - *mult1row1col (R, C)* multiplies one *row* of A with one *column* of B

```
mult (Rows, Cols) -> [ mult1row(R,Cols) || R <- Rows  ].

mult1row (R, Cols) ->
        lists:map(fun(C) -> mult1row1col(R,C) end, Cols).


...
```

- *lists:map* maps an in-place function over all the columns

Example: Simple matrix multiplication

- The sequential Erlang algorithm iterates over the rows
  - *mult (A, B)* multiplies the *rows* of A with the *columns* of B
  - *mult1row (R, B)* multiplies one *row* of A with all the *columns* of B
  - *mult1row1col (R, C)* multiplies one *row* of A with one *column* of B

```
mult (Rows, Cols) -> [ mult1row(R,Cols) || R <- Rows  ].

mult1row (R, Cols) ->
        lists:map(fun(C) -> mult1row1col(R,C) end, Cols).

multi1row1col(R,C) -> ... multiply one row by one column ...
```

Example: Simple matrix multiplication

- To parallelise it, we can *spawn* a process to multiply each row.

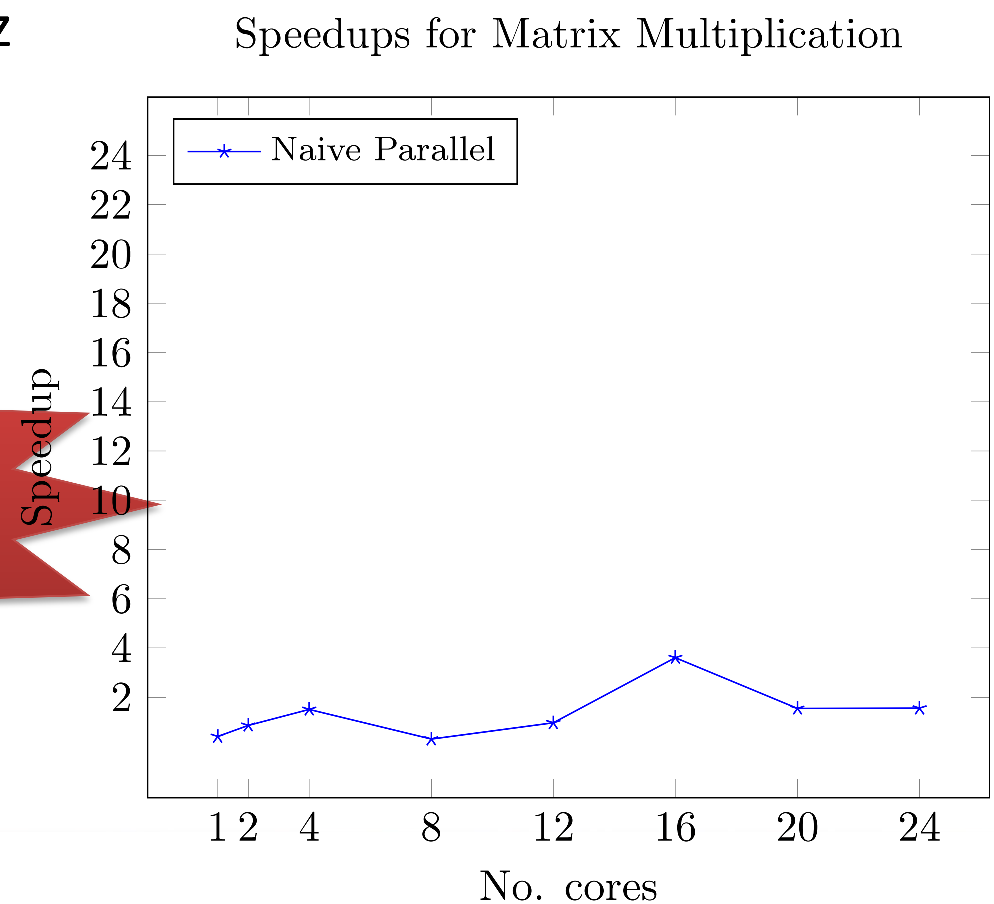
```
mult (Rows, Cols) ->
        ...
        join(
              [ spawn( fun() -> ... mult1row(R,Cols) end ) || R <- Rows ]
        ).

...
```

Speedup Results

- 24 core machine at Uni. Pisa
- AMD Opteron 6176. 800 Mhz
- 32GB RAM

Yikes - SNAFU!!



What's going on?

- We have too many small processes
  - 1,000,000 for our 1000x1000 matrix
  - each process carries setup and scheduling overhead
  - Erlang does not automatically merge processes!

And how can we solve this?

Introduce a

Task Farm

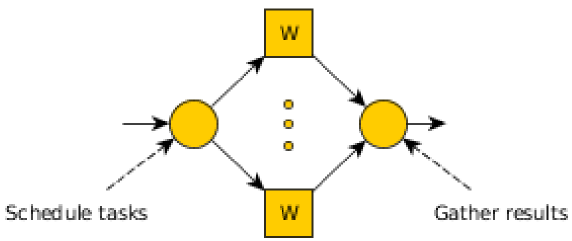
- A high-level *pattern of parallelism*
- A farmer hands out tasks to a fixed number of worker processes
  - This increases granularity and reduces process creation costs

Some Common Patterns

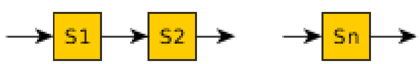
University of St Andrews

- High-level abstract patterns of common parallel algorithms

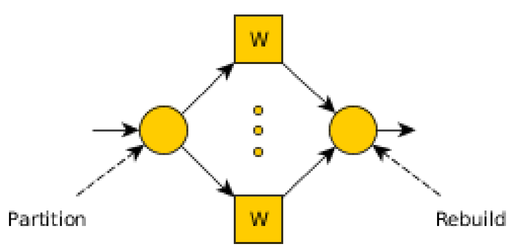
Farm



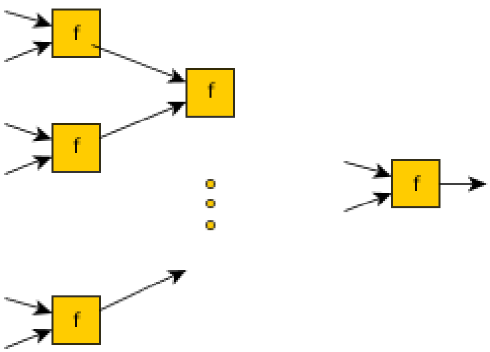
Pipeline



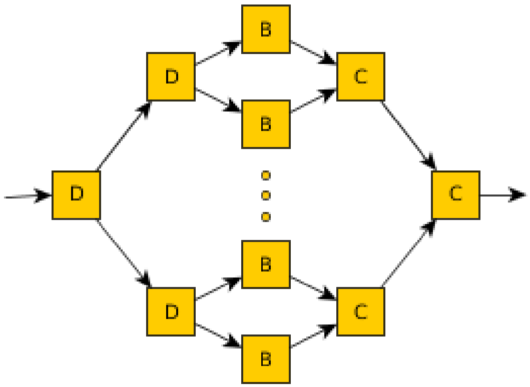
Map



Reduce

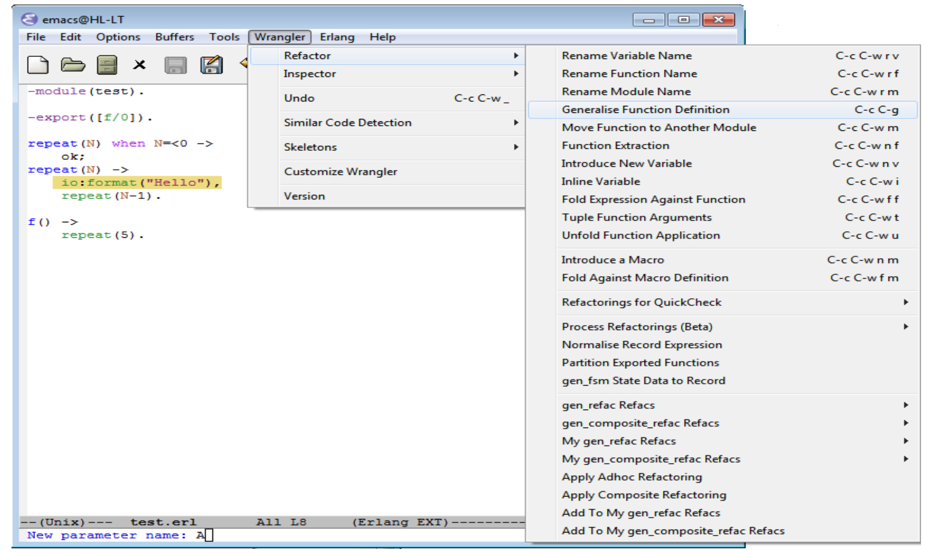


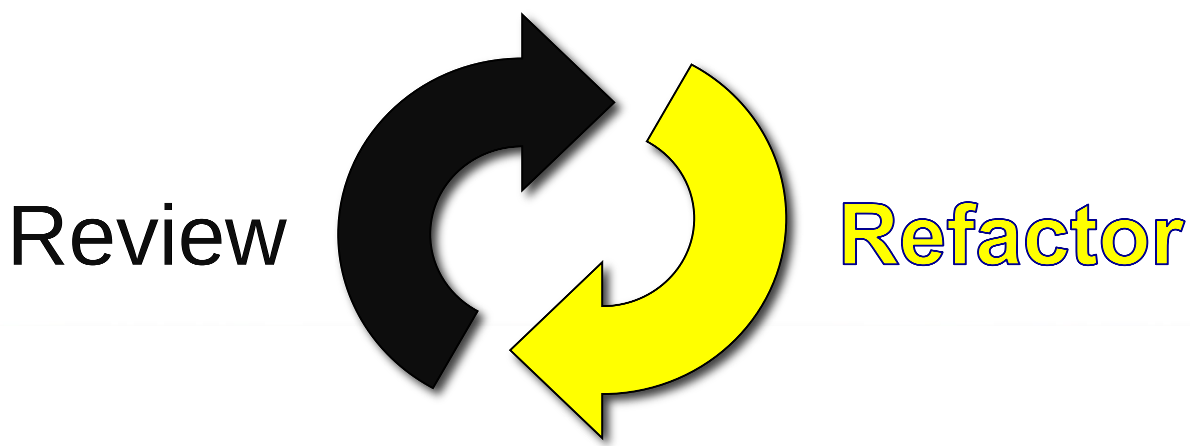
Divide&Conquer



Refactoring

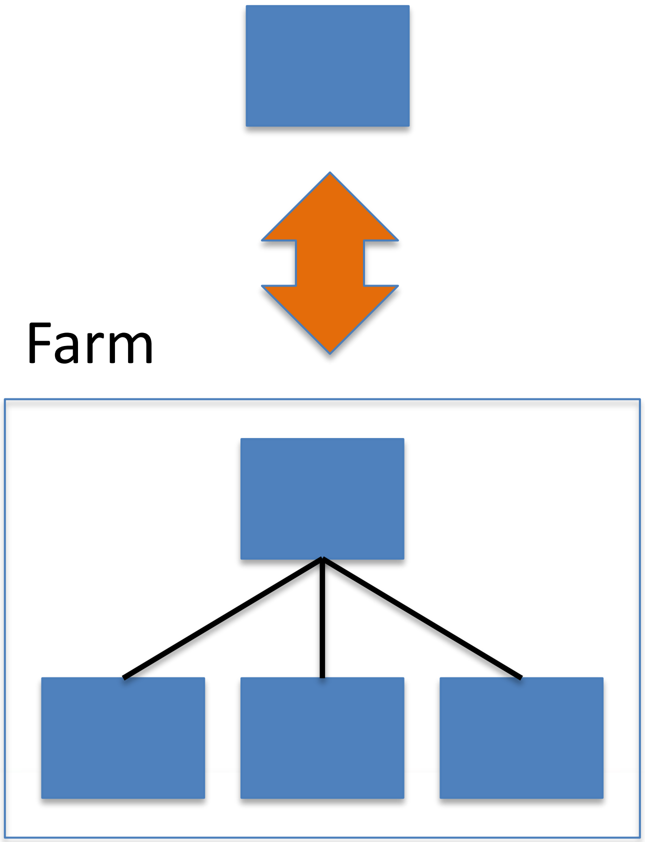
- Refactoring **changes** the **structure** of the **source code**
  - using well-defined rules
  - *semi-automatically under programmer guidance*



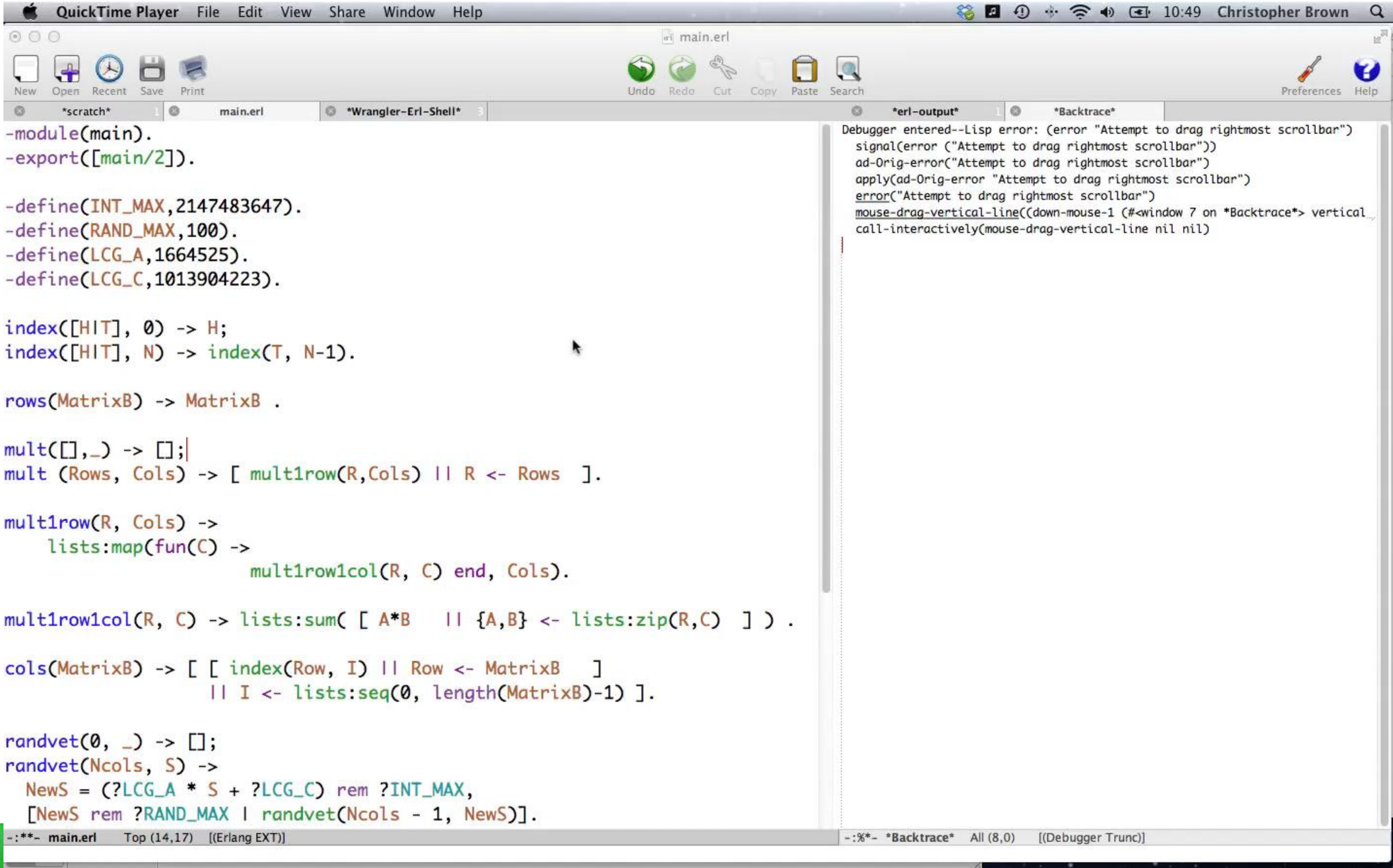


Refactoring: Farm Introduction

$$S \quad \equiv \quad Farm(S) \qquad \textit{farm intro/elim}$$



Demo: Adding a Farm



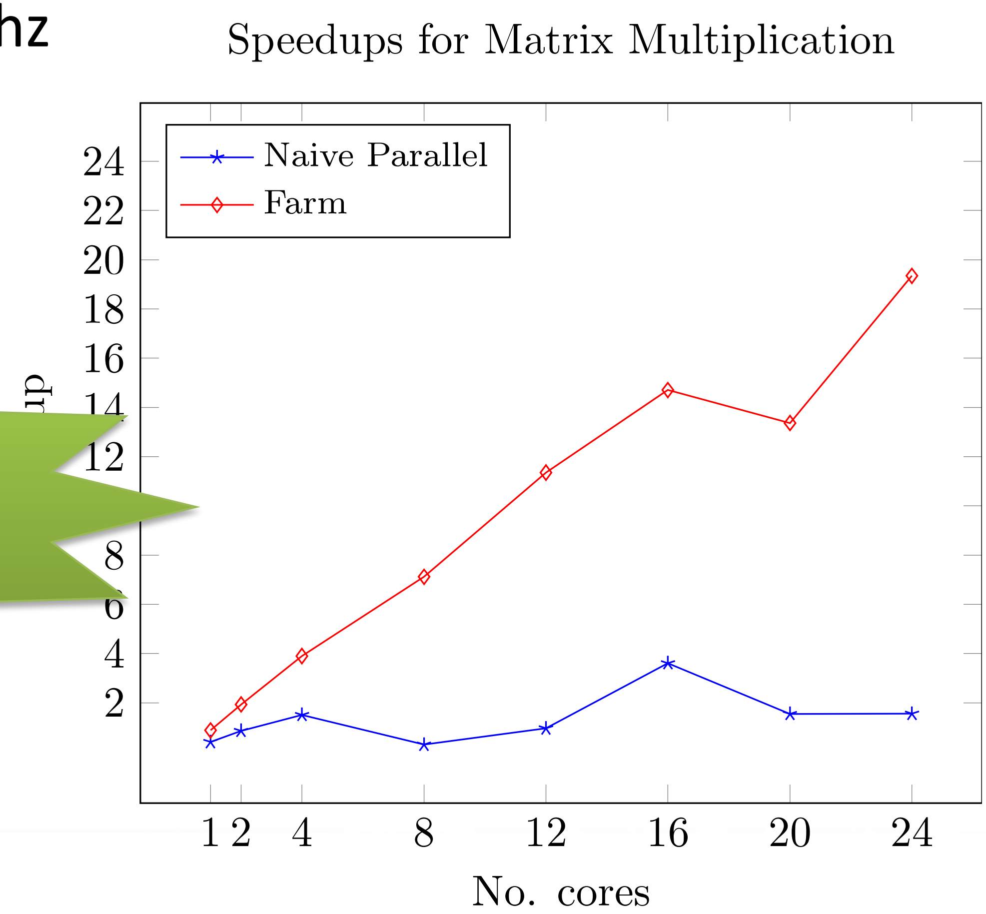
This uses the new Erlang ‘skel’ Library

```
mult([],_) -> [];
mult(Rows,Cols) ->
            skel:run(
               [{farm, ...
                        fun(R) -> lists:map(
                                         fun(C) -> mult_prime(R, C) end,
                                         Cols),
                        ...}],
                 Rows).
```

- Available from
  https://github.com/ParaPhrase/skel

Speedup Results

- 24 core machine at Uni. Pisa
- AMD Opteron 6176. 800 Mhz
- 32GB RAM

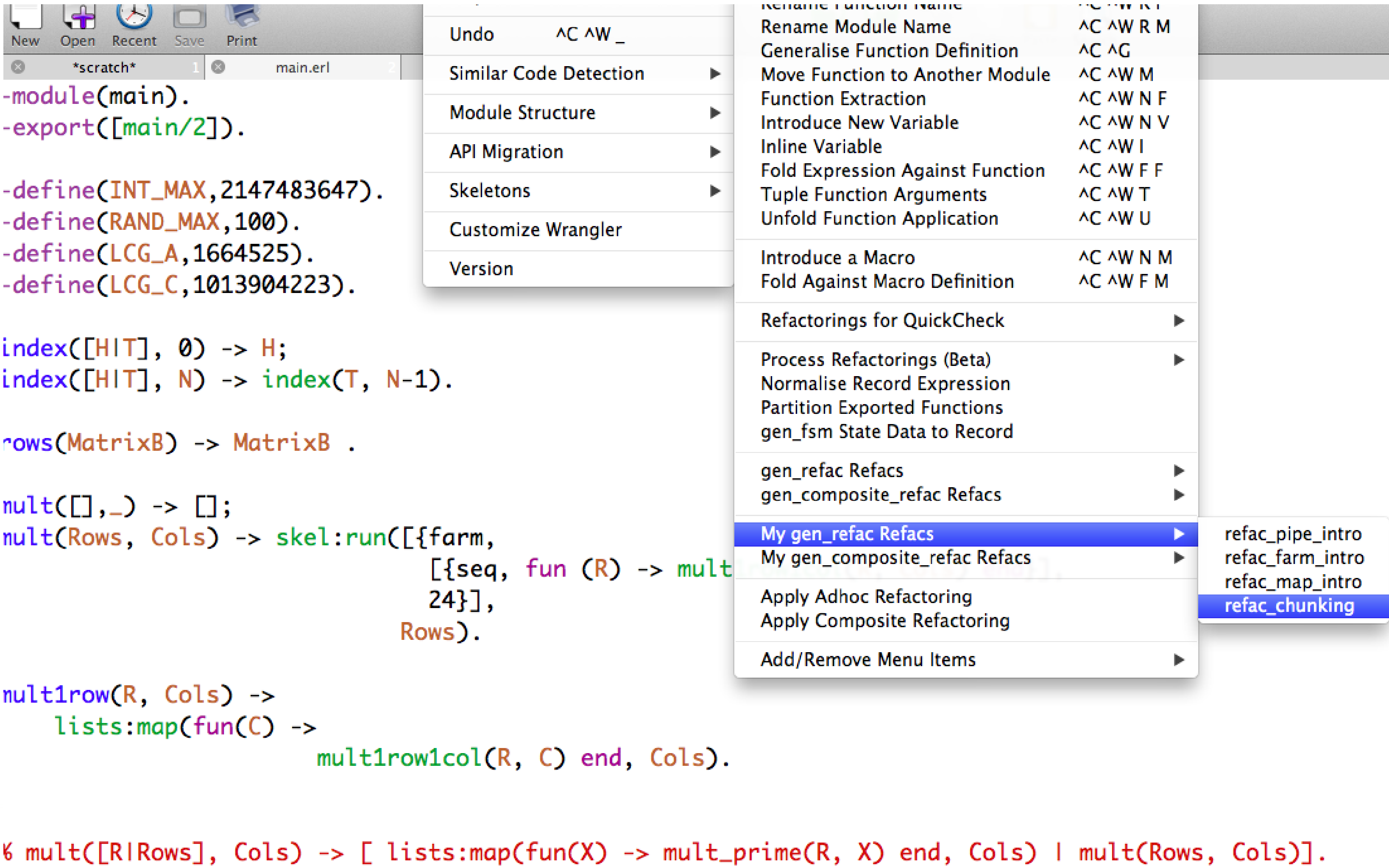


This is much better!

But I don't want to give you that...

- I want to give you more...
- There are ways to improve task size further
  - e.g. "chunking" – combine adjacent data items to increase granularity
  - a poor man's *mapReduce*
- *Just change the pattern slightly!*

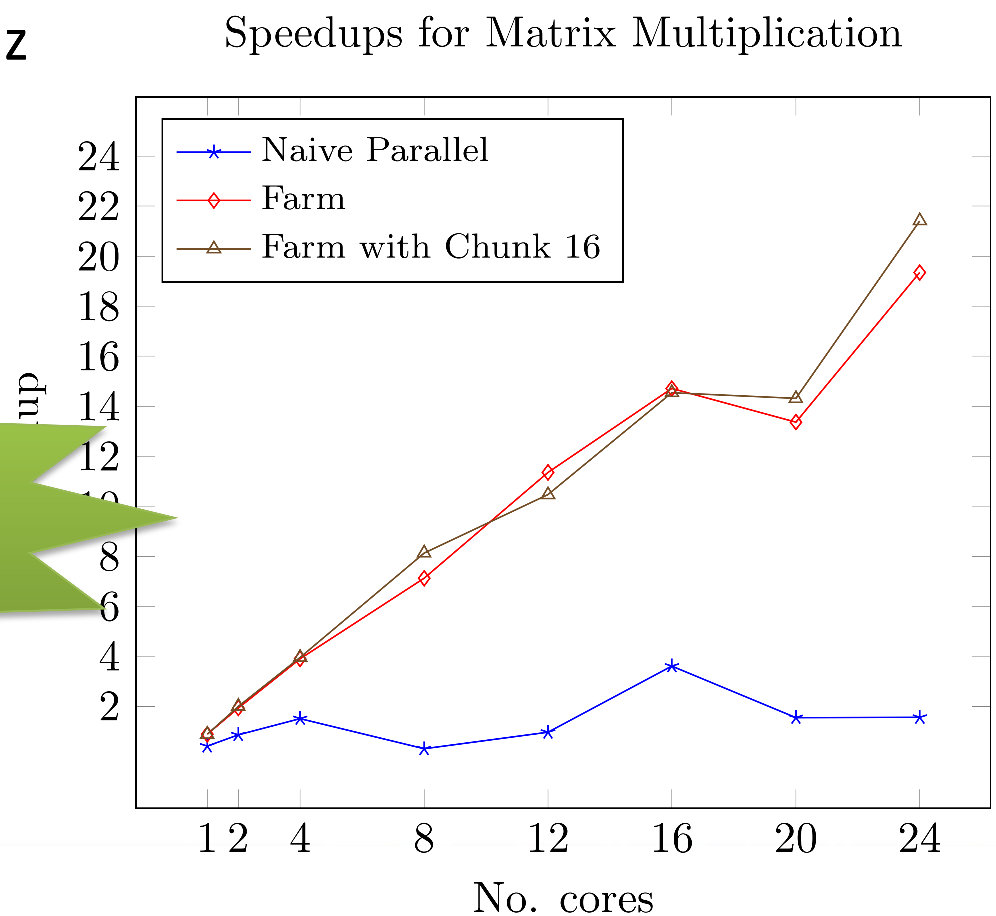
Adding Chunking



Speedup Results

- 24 core machine at Uni. Pisa
- AMD Opteron 6176. 800 Mhz
- 32GB RAM

Chunking gives more improvements!



Conclusions

- Functional programming makes it easy to introduce parallelism
  - No side effects means any computation could be parallel
    - millions of *ultra-lightweight* threads (sub micro-second)
  - Matches pattern-based parallelism
  - Much detail can be abstracted
  - automatic mechanisms for granularity control, synchronisation etc
- Lots of problems can be avoided
  - e.g. Freedom from Deadlock
  - Parallel programs give the same results as sequential ones!
- *But still not completely trivial!!*
  - Need to choose granularity carefully!
    - e.g. thresholding
  - May need to understand the execution model
    - e.g. **pseq**

Isn't this all just wishful thinking?

University of St Andrews

NO!

University of St Andrews

- C++11 has lambda functions
- Java 8 will have lambda (closures)
- Apple uses closures in Grand Central Dispatch

ParaPhrase Parallel C++ Refactoring

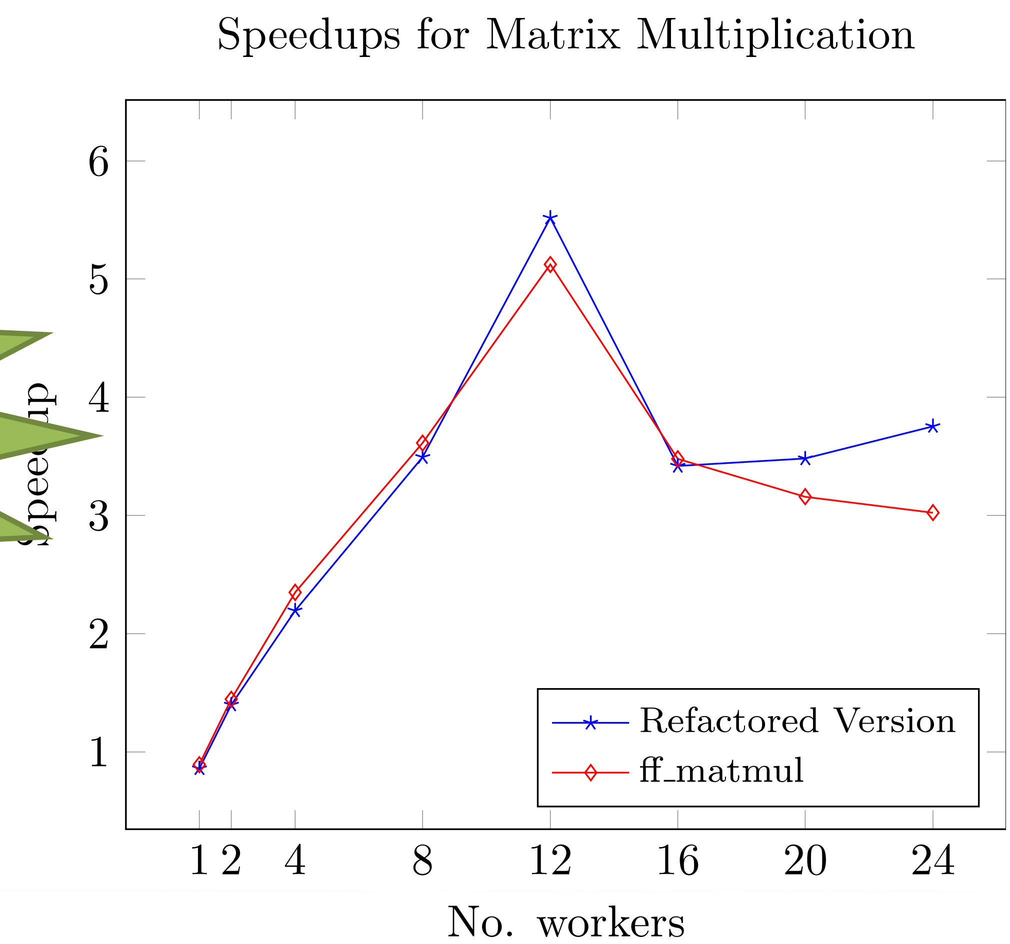
University of St Andrews

- Integrated into Eclipse
- Supports full C++(11) standard
- Uses strongly hygienic components
  - functional encapsulation (closures)

Performance of FastFlow C++ Library

- 5.5 speedup on 12 cores

Compared with 5.1 speedup from a hand-optimised version



Further Reading

Chris Brown. Marco Danelutto, Kevin Hammond, Peter Kilpatrick and Sam Elliot
"Cost-Directed Refactoring for Parallel Erlang Programs"
Proc. 2013 International Symposium on High-level Parallel Programming and Applications (HLPP), Paris, France, June 2013

Chris Brown. Hans-Wolfgang Loidl and Kevin Hammond
"ParaForming Forming Parallel Haskell Programs using Novel Refactoring Techniques"
Proc. 2011 Trends in Functional Programming (TFP), Madrid, Spain, May 2011

Henrique Ferreiro, David Castro, Vladimir Janjic and Kevin Hammond
"Repeating History: Execution Replay for Parallel Haskell Programs"
Proc. 2012 Trends in Functional Programming (TFP), St Andrews, UK, June 2012

Funded by

- **ParaPhrase (EU FP7), Patterns for heterogeneous multicore,**
  €2.6M, 2011-2014
- **SCIEnce (EU FP6), Grid/Cloud/Multicore coordination**
  - €3.2M, 2005-2012
- **Advance (EU FP7), Multicore streaming**
  - €2.7M, 2010-2013
- **HPC-GAP (EPSRC), Legacy system on thousands of cores**
  - £1.6M, 2010-2014
- **Islay (EPSRC), Real-time FPGA streaming implementation**
  - £1.4M, 2008-2011
- **TACLE: European Cost Action on Timing Analysis**
  - €300K, 2012-2015

Industrial Connections

Mellanox Inc.

Erlang Solutions Ltd

SAP GmbH, Karlsrühe

BAe Systems

Selex Galileo

BioId GmbH, Stuttgart

Philips Healthcare

Software Competence Centre, Hagenberg

Microsoft Research

Well-Typed LLC

THANK YOU!

http://www.paraphrase-ict.eu

http://www.project-advance.eu

@paraphrase_fp7