# Bridging the Divide: A *New* Tool-Support Methodology for Programming Heterogeneous Multi-Core Machines

Chris Brown, Vladimir Janjic and Kevin Hammond
University of St Andrews, Scotland

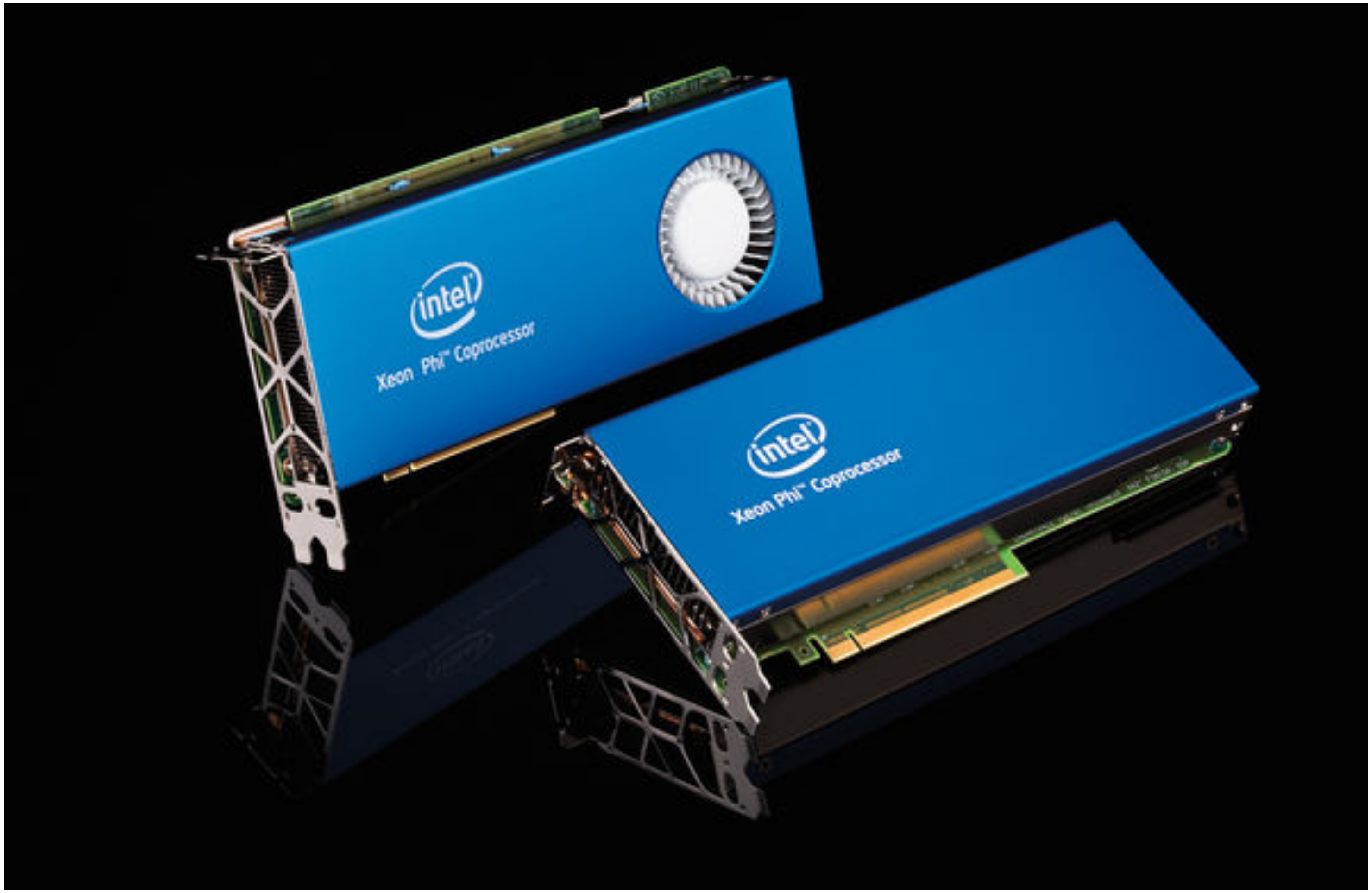
T: *@chrismarkbrown, @paraphrase_fp7*
E: cmb21@*st-andrews.ac.uk*
W: http://www.paraphrase-ict.eu

University of St Andrews

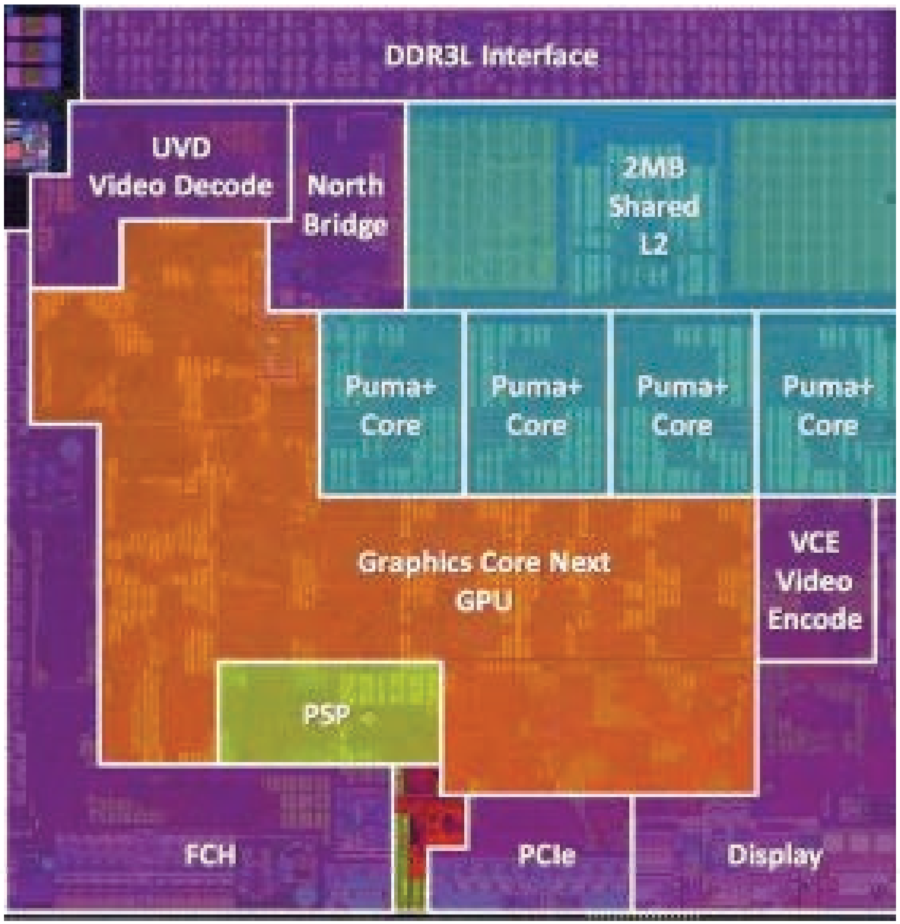
SEVENTH FRAMEWORK PROGRAMME

PARAPHRASE

# 2014: a ManyCore Odyssey



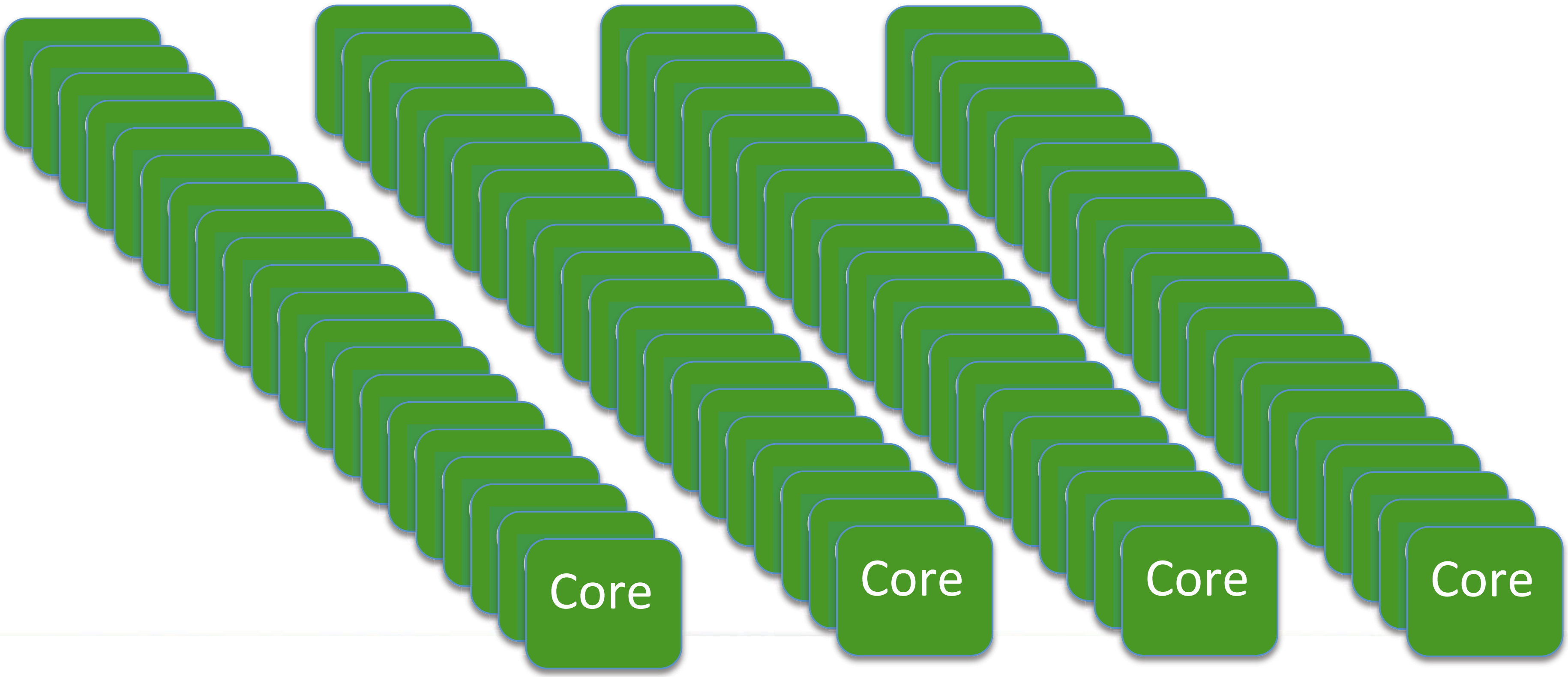
# AMD Mullins/Beema APU




- 4 Core x86 CPU
- 1 ARM PSP Security Core
- Graphics Core next to GPU with 128 cores
  - Used in e.g. Xbox 360
- Power consumption ~4.5W

# The Future: “megacore” computers?

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



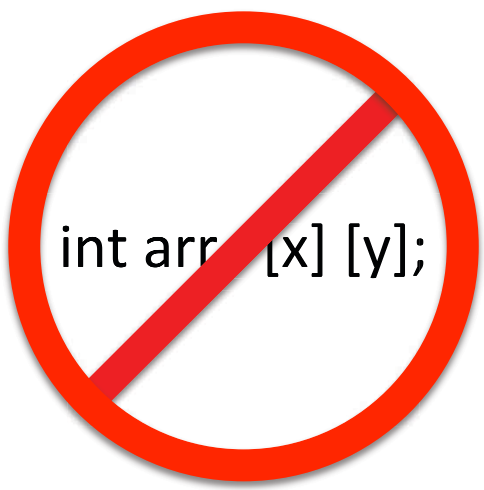
# What will "megacore" computers look like?

University of St Andrews

- **Probably *not* just scaled versions of today's multicore**
  - Perhaps hundreds of dedicated lightweight integer units
  - Hundreds of floating point units (enhanced GPU designs)
  - A *few* heavyweight general-purpose cores
  - Some specialised units for graphics, authentication, network etc
  - possibly *soft* cores (FPGAs etc)
  - ***Highly heterogeneous***

# What will "megacore" computers look like?

- **Probably *not* uniform shared memory**
  - NUMA is likely, even hardware distributed shared memory
  - or even message-passing systems on a chip
  - *shared-memory will not be a good abstraction*



# 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 Fastest Computer in the World

**Tianhe-2, Chinese National University of Defence Technology**

33.86 petaflops/s (June 17, 2013)
16,000 Nodes; each with 2 Ivy Bridge multicores and 3 Xeon Phis
**3,120,000 x86 cores in total!!!**

# It’s not just about large systems



- Even mobile phones are multicore
  - Samsung Exynos 5 Octa has 8 cores, 4 of which are “dark”
- Performance/energy tradeoffs mean systems will be increasingly parallel
- If we don’t solve the multicore challenge, then no other advances will matter!


ALL Future Programming will be Parallel!

# Parallel Hardware Today

- Computer hardware is getting more and more parallel
  - 64-core machines available off-the-shelf for a modest price
- It is also getting more and more heterogeneous
  - Any decent desktop machine comprises a multicore CPU and many-core GPU
  - Even mobile phones come with multiple GPUs


# The Manycore Challenge

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

The **ONLY** important challenge in Computer Science (Intel)

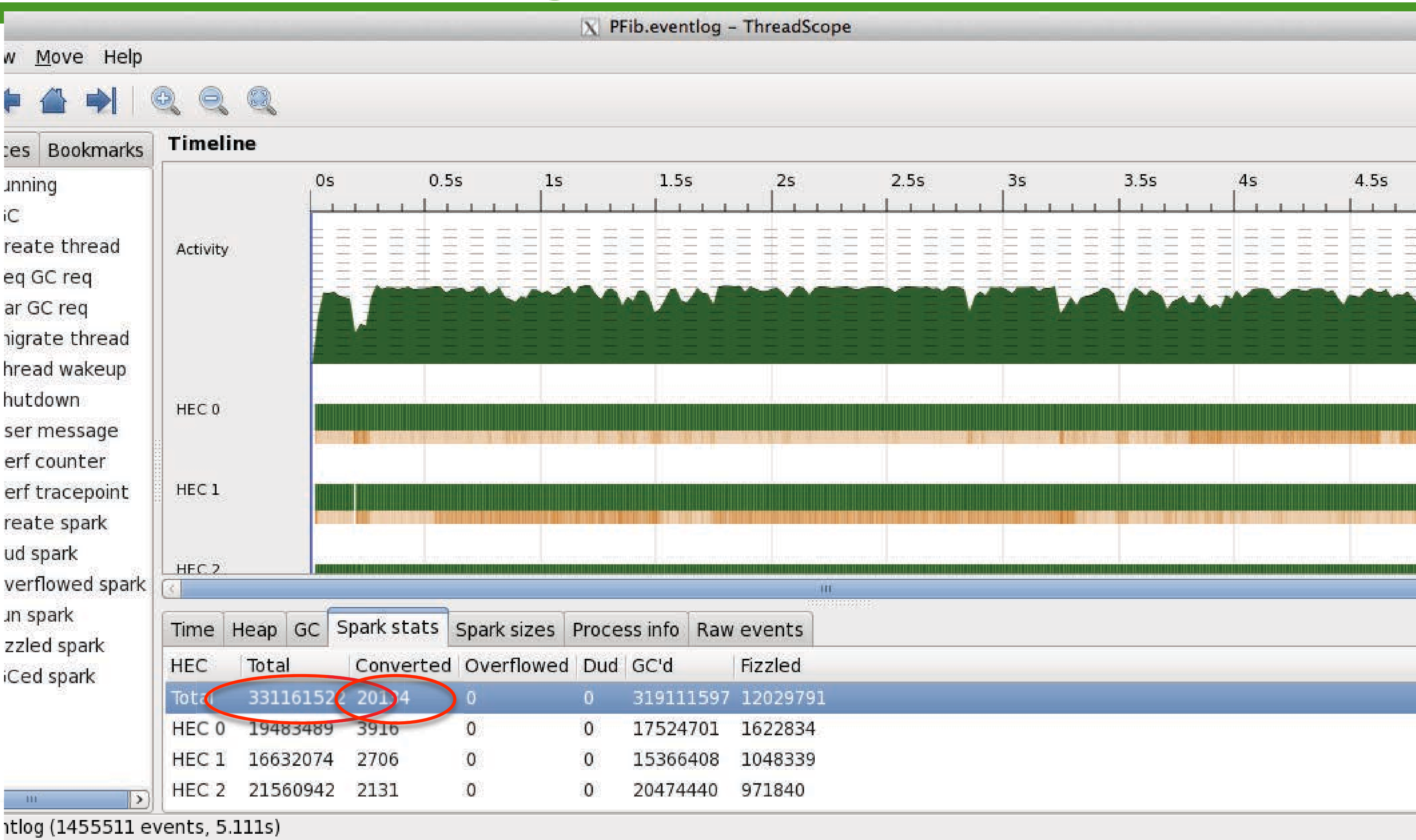
will not ``automagically'' run

Also recognised as thematic priorities by EU and national funding bodies

**Patrick Leonard, Vice President for Product Development**
**Rogue Wave Software**

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

University of St Andrews



# 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!

University of St Andrews

SEVENTH FRAMEWORK PROGRAMME

**ParaPhrase Project: Parallel Patterns for Heterogeneous Multicore Systems (ICT-288570), 2011-2014, €4.2M budget**

**13 Partners, 8 European countries**

**UK, Italy, Germany, Austria, Ireland, Hungary, Poland, Israel**

**Coordinated by Kevin Hammond St Andrews**

AGH

Erlang SOLUTIONS

HLRS

Queen's University Belfast

National College of Ireland

ROBERT GORDON UNIVERSITY·ABERDEEN

UNIVERSITÀ DEGLI STUDI DI TORINO

ALMA UNIVERSITAS TAURINENSIS

Mellanox TECHNOLOGIES

scch software competence center hagenberg

UNIVERSITÀ DI PISA

PARAPHRASE

# The ParaPhrase Approach

- Start bottom-up
  - identify (strongly hygienic) **COMPONENTS**
  - *using semi-automated refactoring*

*both legacy and new programs*

- 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)

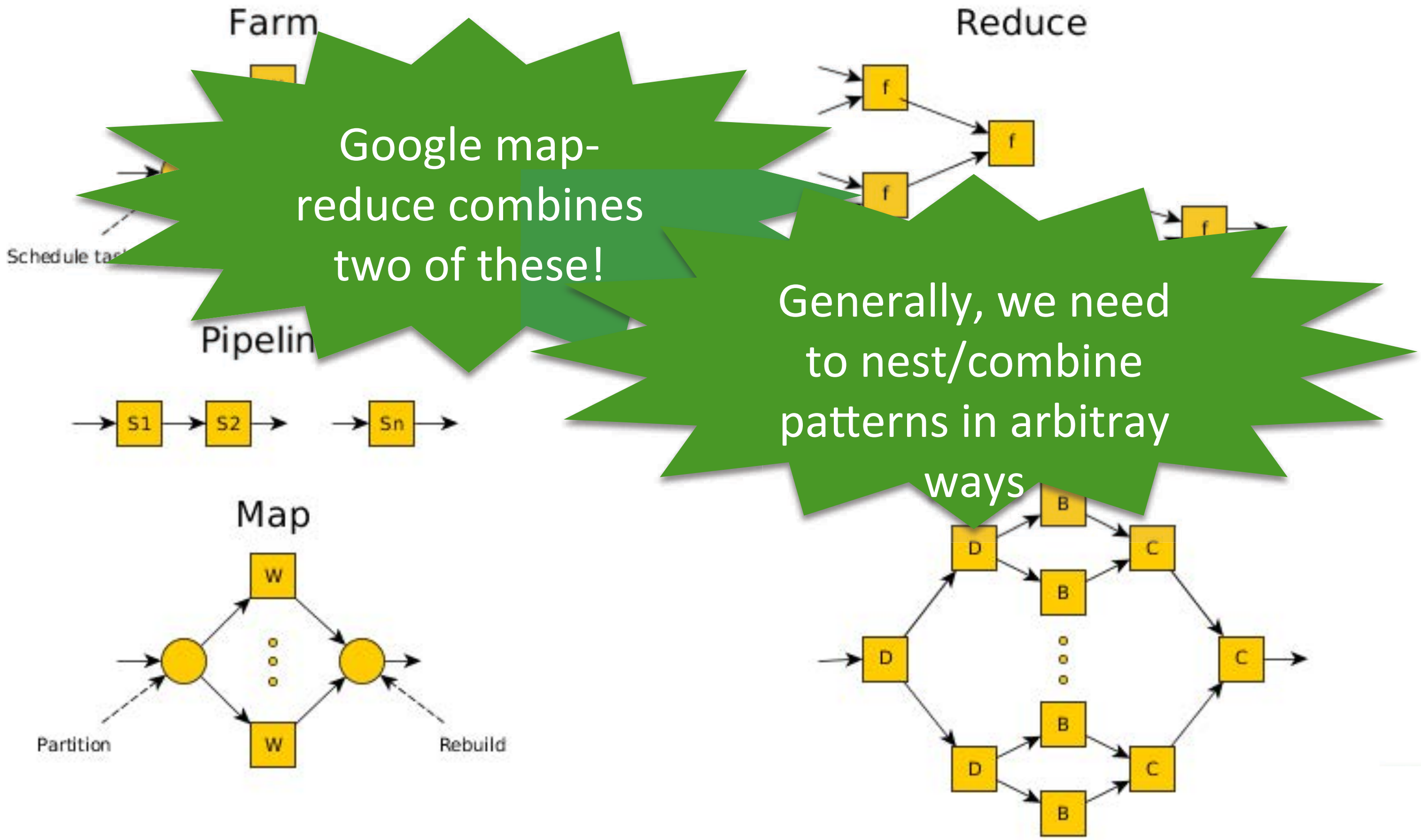
# In This Talk…

University of St Andrews

- Provide an Erlang skeleton library to make it easier to deal with parallelism
- Extend this library to deal with CPU/GPU systems
  - Heterogeneous Erlang skeletons
  - openCL bindings
- Provide refactoring Tool-Support to ease the introduction of the GPU code
- Show initial heterogeneous results for Erlang

# Some Common Patterns

- High-level abstract patterns of common parallel algorithms



# The *Skel* Library for Erlang

University of St Andrews

- Skeletons implement specific parallel patterns
  - Pluggable templates
- **Skel** is a new (AND ONLY!) Skeleton library in Erlang
  - map, farm, reduce, pipeline, feedback
  - instantiated using **skel:do**
- ***Fully Nestable***
- **A DSL for parallelism**

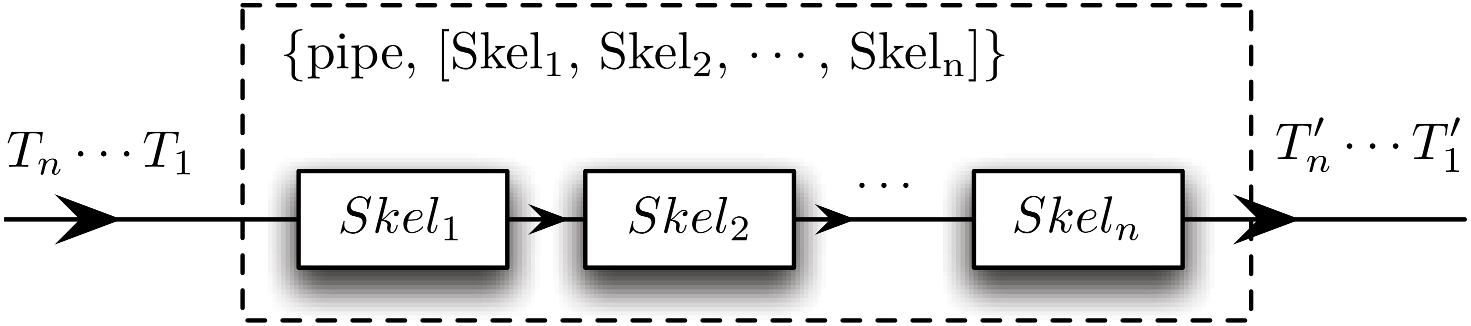
chrisb.host.cs.st-andrews.ac.uk/skel.html

https://github.com/ParaPhrase/skel

```
OutputItems = skel:do(Skeleton, InputItems).
```

# Parallel Pipeline Skeleton

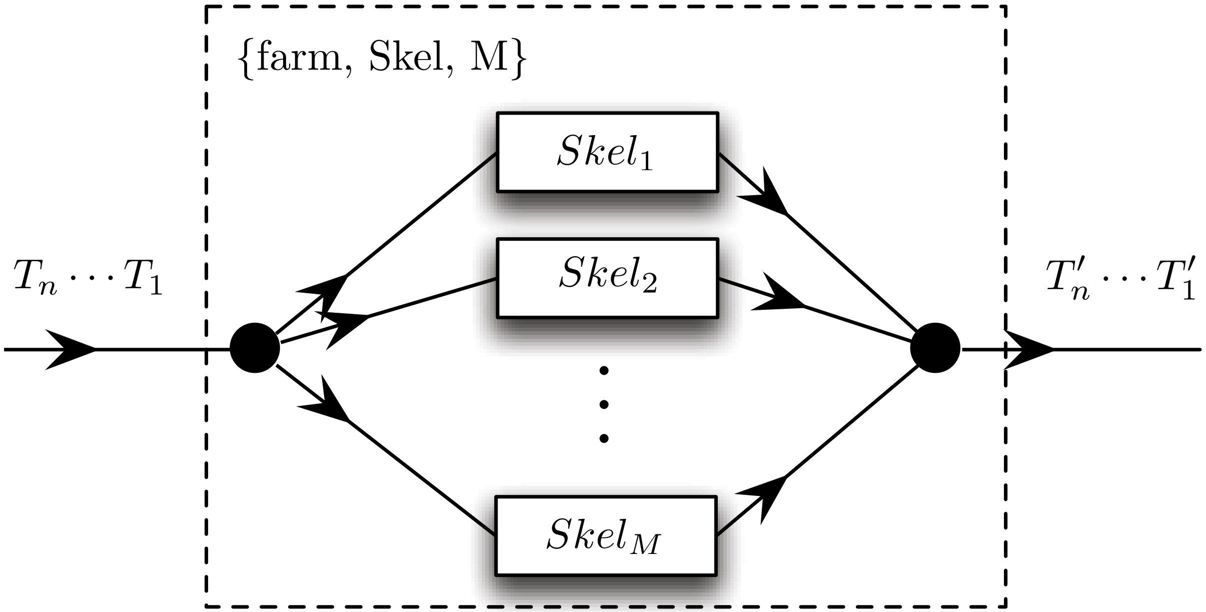
- Each stage of the pipeline can be executed in parallel
- The input and output are streams




```
Skel:do([{pipe,[Skel1, Skel2,..,SkelN]}], Inputs).
```

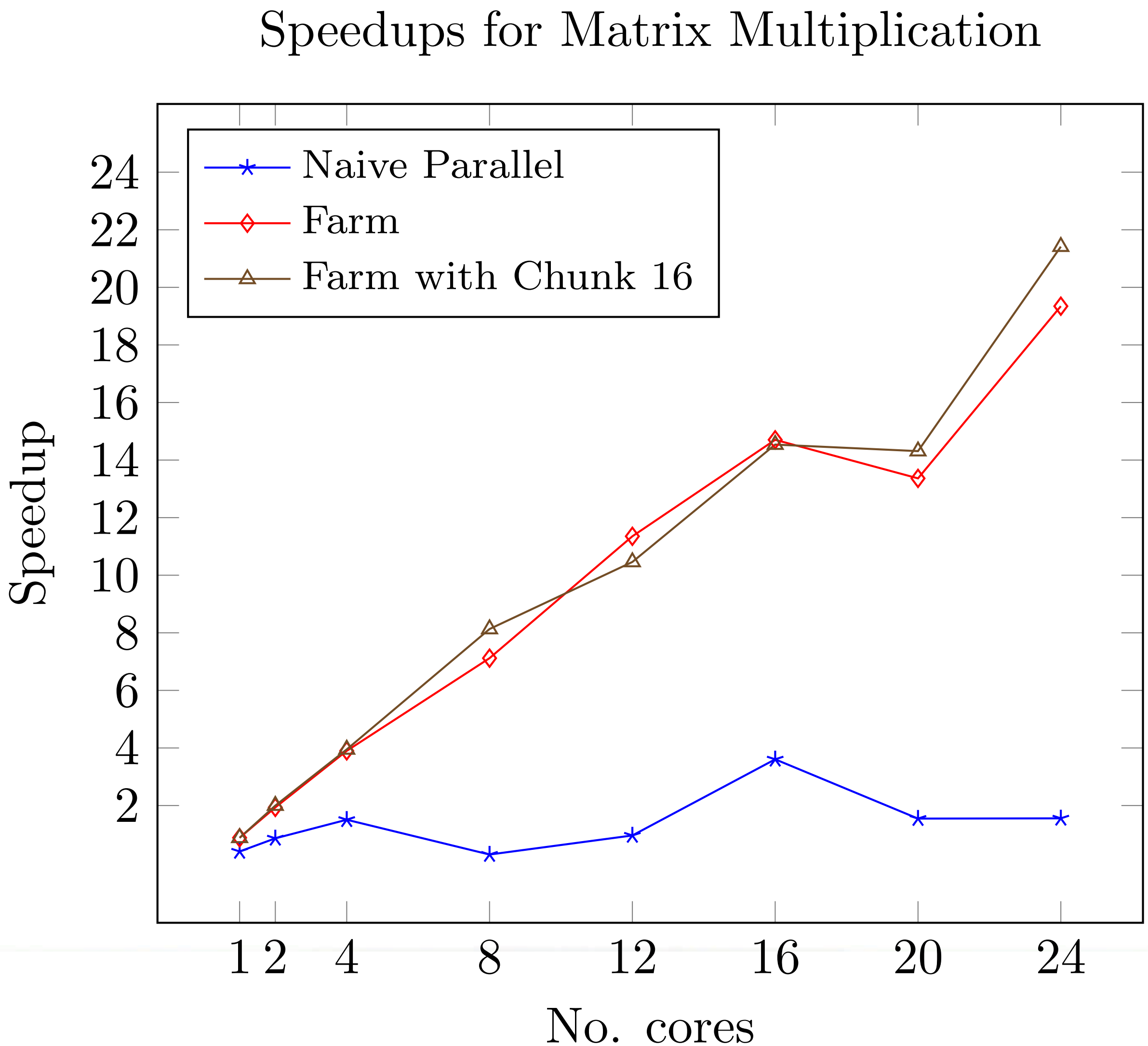
# Farm Skeleton

- Each worker is executed in parallel
- A bit like a 1-stage pipeline



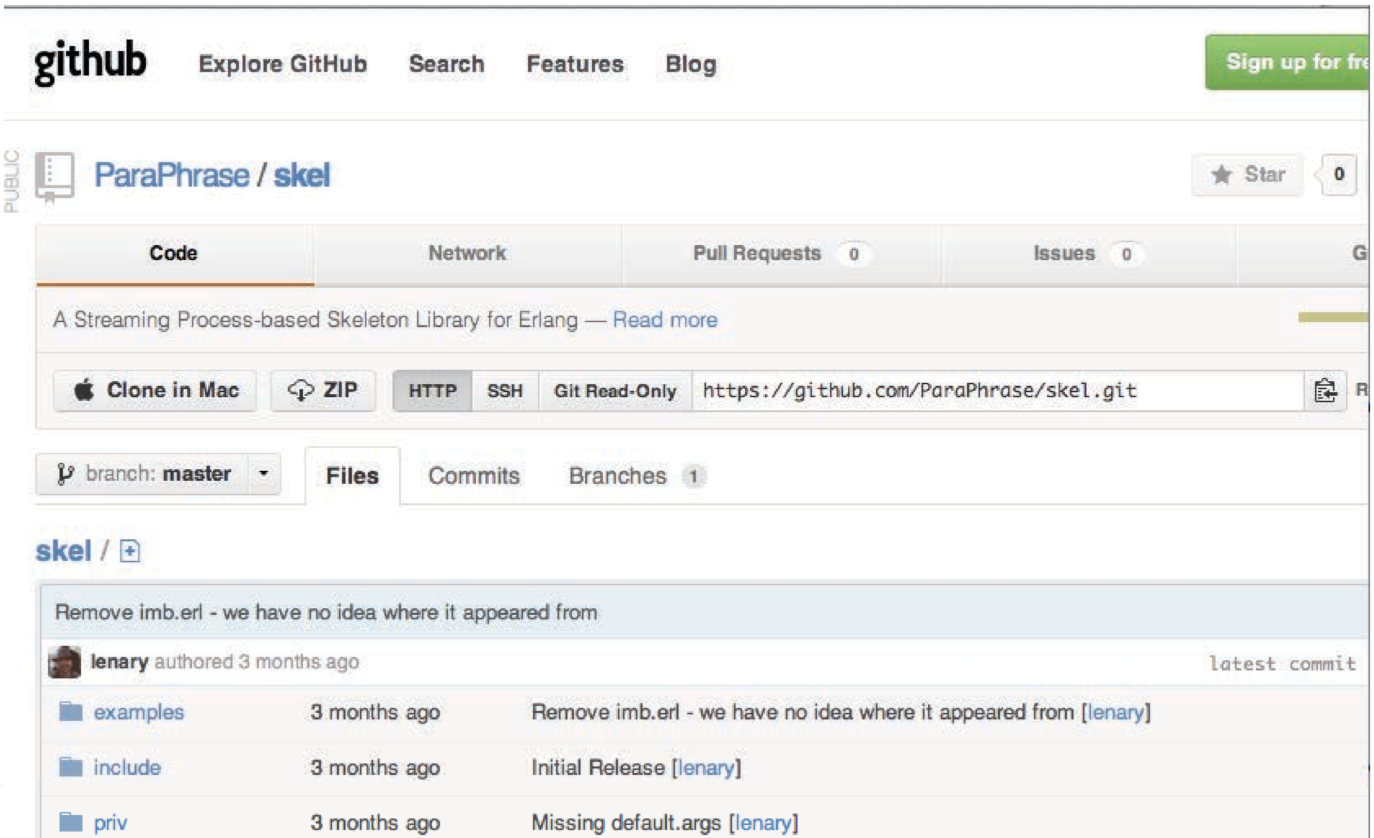

```
skel:do([{farm, Skel, M}], Inputs).
```

# Using The Right Pattern Matters



# Download from ...

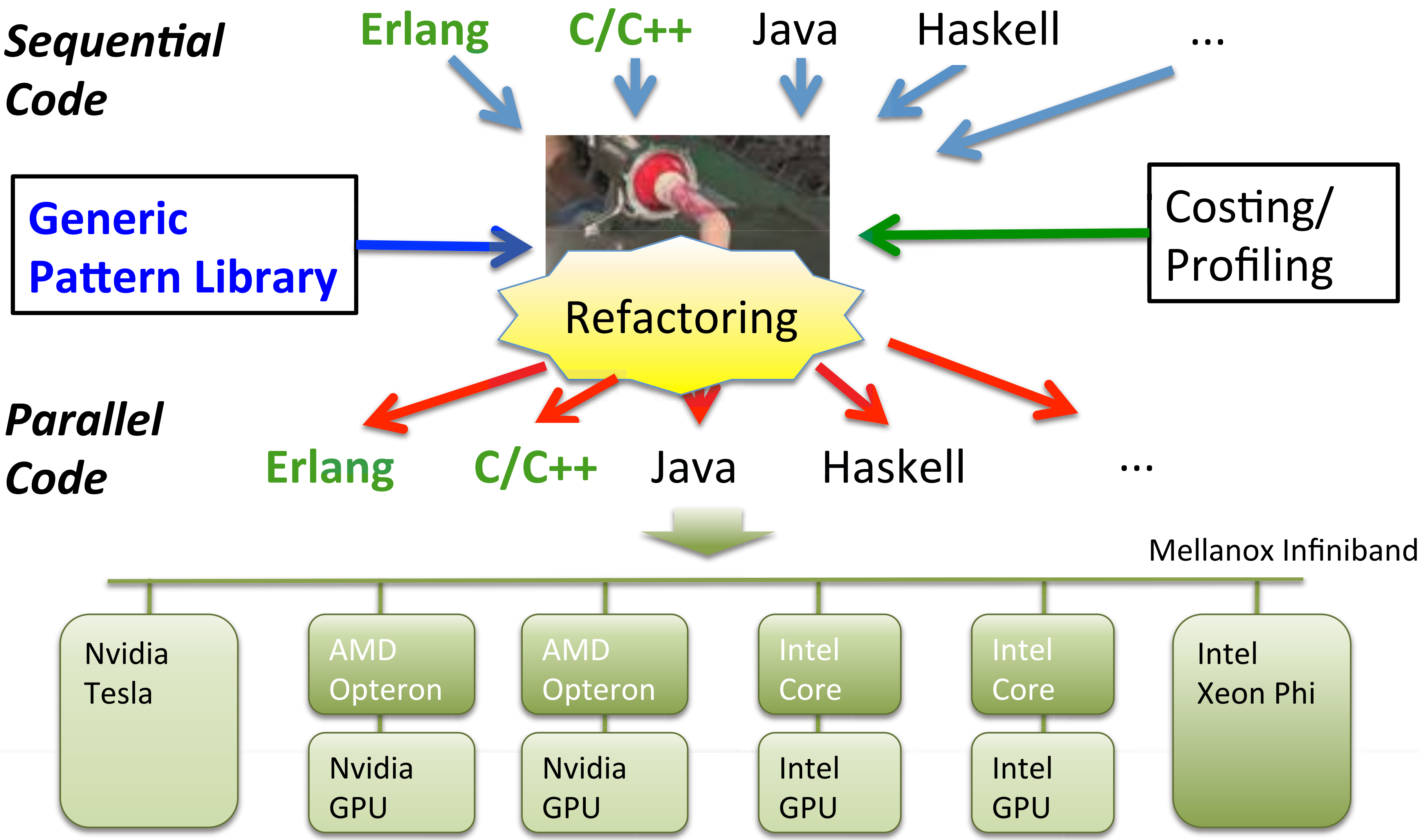
http://www.cs.st-andrews.ac.uk/~chrisb/ParaPhrase_Refactorer.tar.gz (Refactoring Tool)
https://github.com/ParaPhrase/skel (Skel Library)



# Constructing Farms…

- Seq, component wrapper for worker function:
  - `{seq, fun worker/1}`
- Create a farm of workers:
  - `{farm,[{seq, fun worker/1}], nWorkers}`
- Wrap it inside a skel call:
  - `skel:do([{farm, [{seq, fun worker/1}], nWorkers}], input)`

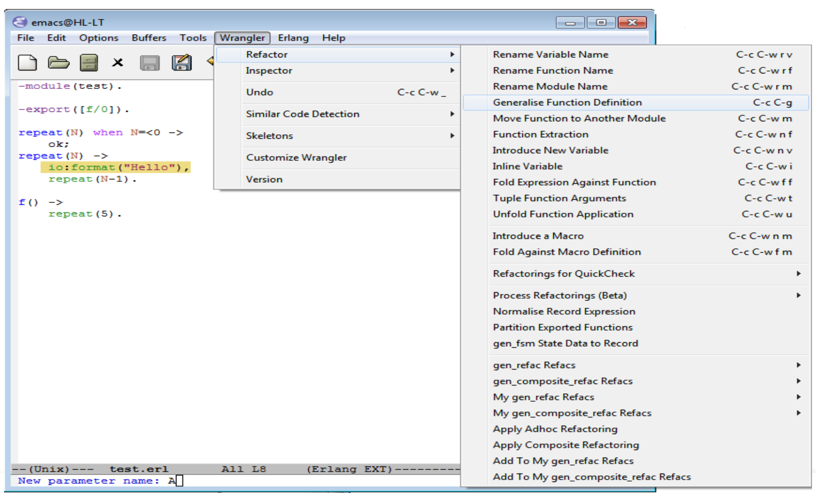
# The ParaPhrase Approach

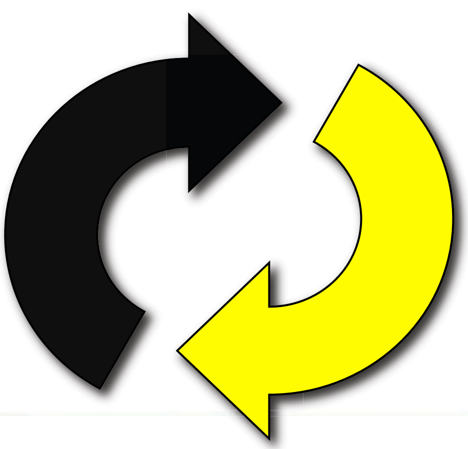


# Refactoring Tool Support

- The process of changing the structure of an application while preserving its functional semantics

- Semi-automated approach that is more general than fully automated parallelisation techniques



# Programing Heterogeneous Systems…

- …is hard!
- Mainstream programming models (e.g. OpenCL, CUDA+pthreads) are too low-level for an average programmer
- Many applications can be parallelised in more than one way
- Choosing which parallel structure to exploit is a non-trivial problem
  - Trial-and-error approach can be very costly

# Linking with OpenCL

- OpenCL binding for Erlang
  - Basically wraps up openCL in Erlang ‘FFI’ like calls
  - User required to provide an openCL kernel
  - Provides GPU setup/offloading/marshalling …
  - Requires kernel parameters to be Erlang binaries
    - Basically a pointer to the raw data
  - https://github.com/tonyrog/cl

# Linking with OpenCL

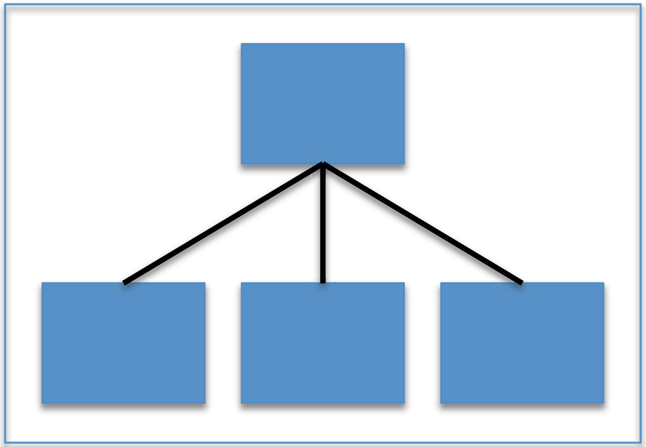
```
E = clu:setup(all),
{ok,Program} = clu:build_source(E, "solve2"),
{ok,Kernel} = cl:create_kernel(Program,
                               "solveKernel”)

cl:create_buffer(E#cl.context, [read_only],
                    byte_size(Argument)),

cl:set_kernel_arg(Kernel, 0,
                         K#kwork.argument),
```

# Linking with OpenCL

```
{ok,E3} =
cl:enqueue_read_buffer(K#kwork.queue,

                    K#kwork.omem,0,Nk,[E2]),

{ok,Bin} = cl:wait(K#kwork.e3),
```

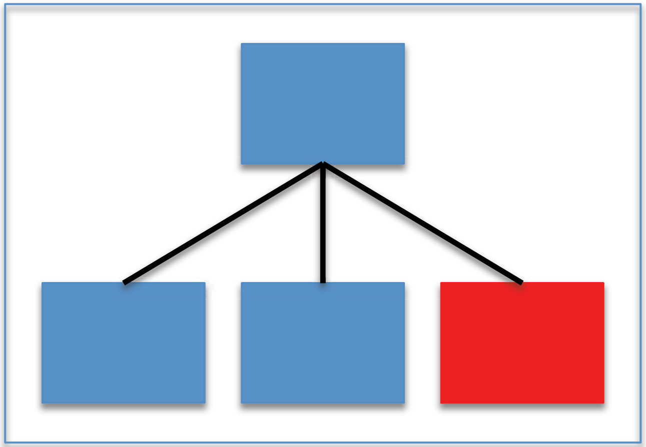
# Heterogeneous Patterns

Farm



# Heterogeneous Patterns

Heterogeneous
Farm



# Heterogeneous Farms

- New types of heterogeneous components:
  - {seqCPU,  fun CPUworker/1, nCPUWorkers}
  - {seqGPU, fun GPUworker/1, nGPUWorkers}
- Heterogeneous Farms:
  - Skel:do([{farm, {seqCPU, fun CPUworker/1, nCPUWorkers},
    {seqGPU, fun GPUworker/1, nGPUWorkers}], inputs)

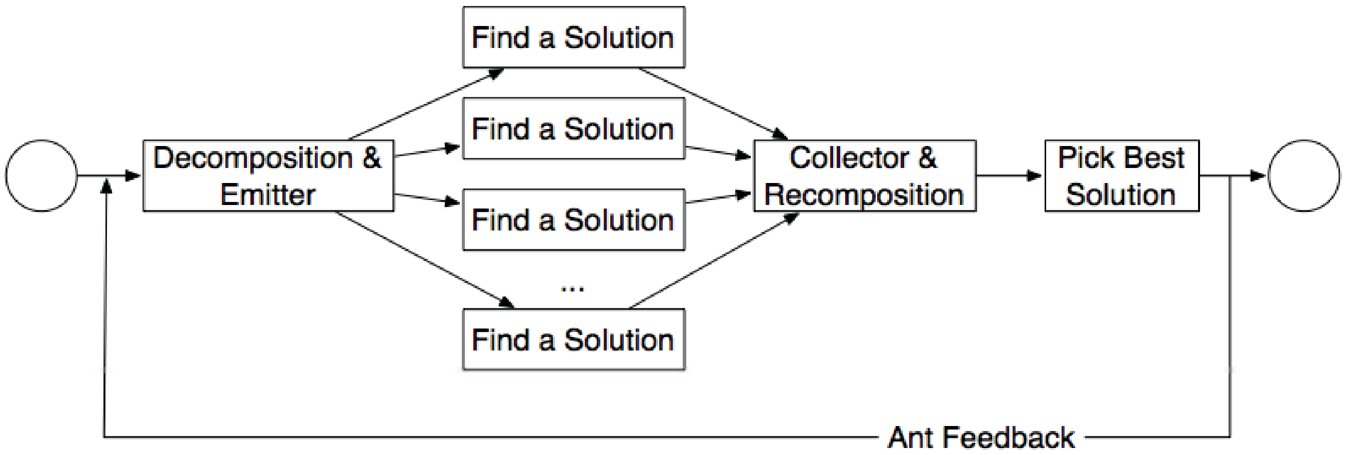
# Heterogeneous Parallel Refactoring

- Generates calls to openCL bindings
  - Uses dialyzer underneath to find the types of the kernel arguments
- Eliminates tedious and massively error-prone openCL writing
- Assumes an already supplied openCL kernel
- Adds in number GPU/CPU workers, using a static mapping technology

# Ant Colony Optimisation



- An ACO algorithm consists of a number of iterations in which each ant finds a solution, partially guided by a *pheromone trail.*
- The *pheromone trail* is updated based on the best solution in each iteration.
- We use ACO to solve the Single Machine Total Weighted Tardiness Problem
- A Skel task farm and feedback skeletons are used to parallelise ACO



# Ant Colony Optimisation

University of St Andrews

```
ant_colony(FName, Num_Ants, Num_Iters, Num_Workers) ->
   {Num_Jobs, Process_Time, Weight, Deadline, Tau} =
binary_ant_init:init(FName),
   Chunk_Size = Num_Ants div Num_Workers,

   Pipe = {pipe, [{farm, [{seqCPU, fun(X) -> lists:map(fun(Y) ->
                         binary_par_solve:find_solution(Y) end, X) end,
                         nCPUWorkers}],

             {seq, fun(X) -> pick_update_spawn_list_lists(Num_Workers,
                                              Chunk_Size, X) end}]},
```

# Ant Colony Optimisation

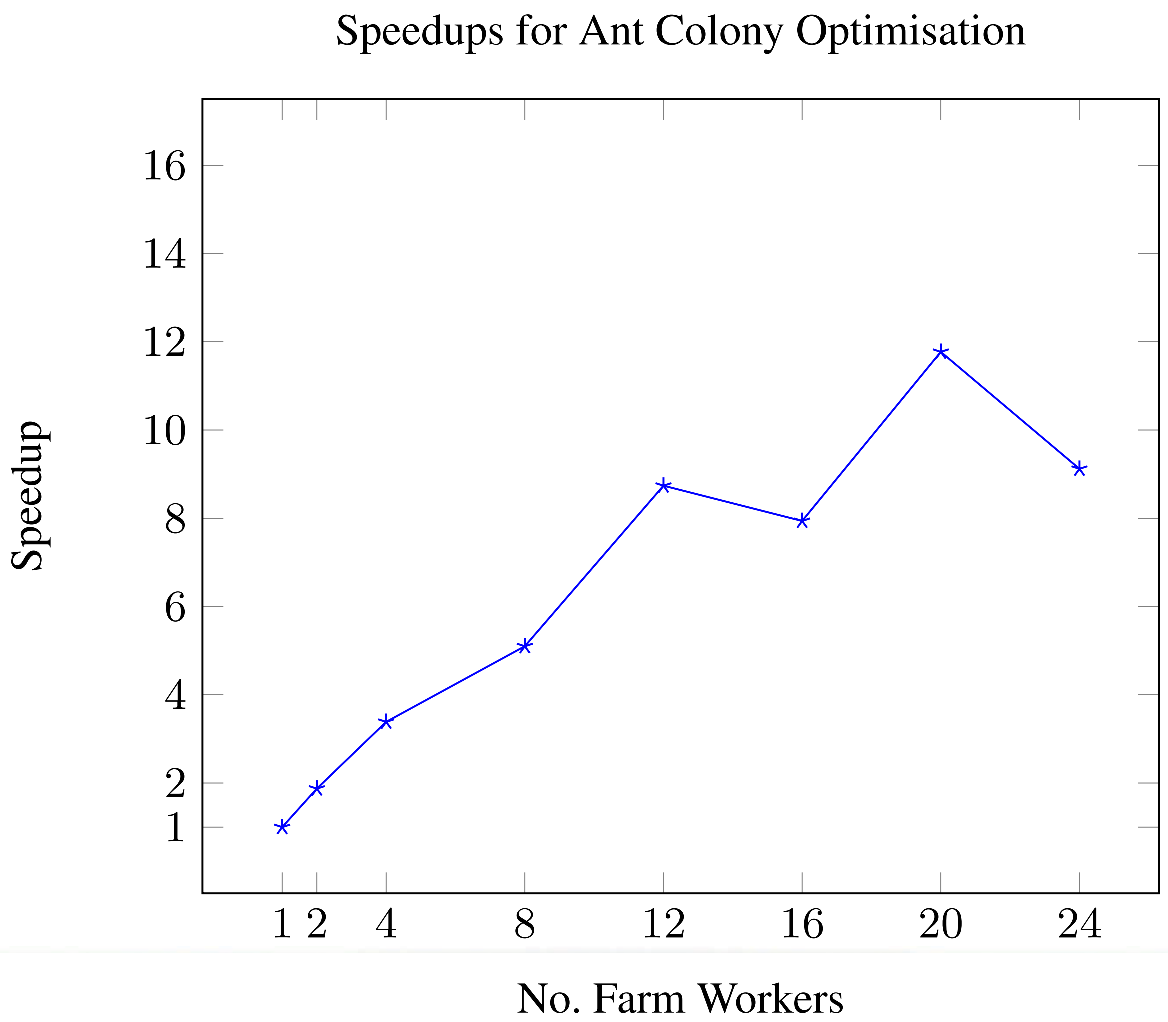
```
Feedback = {feedback, [Pipe], fun ant_feedback/1},
   skel:do([Feedback], [lists:duplicate(Num_Workers,
                        lists:duplicate(Chunk_Size,
                              {Num_Jobs, Process_Time, Weight,
Deadline, Tau, Num_Iters}))]).
```

# Experimental Machine

All measurements

- 2.4GHz 24-core, dual AMD Opteron 6176 architecture
- Nvidia Tesla C2050 Fermi GPU (448 CUDA cores)
- Centos Linux 2.6.18-274.e15.
- Erlang 5.9.1 R15B01,
- Averaging over 10 runs

# Parallel ACO with Skel



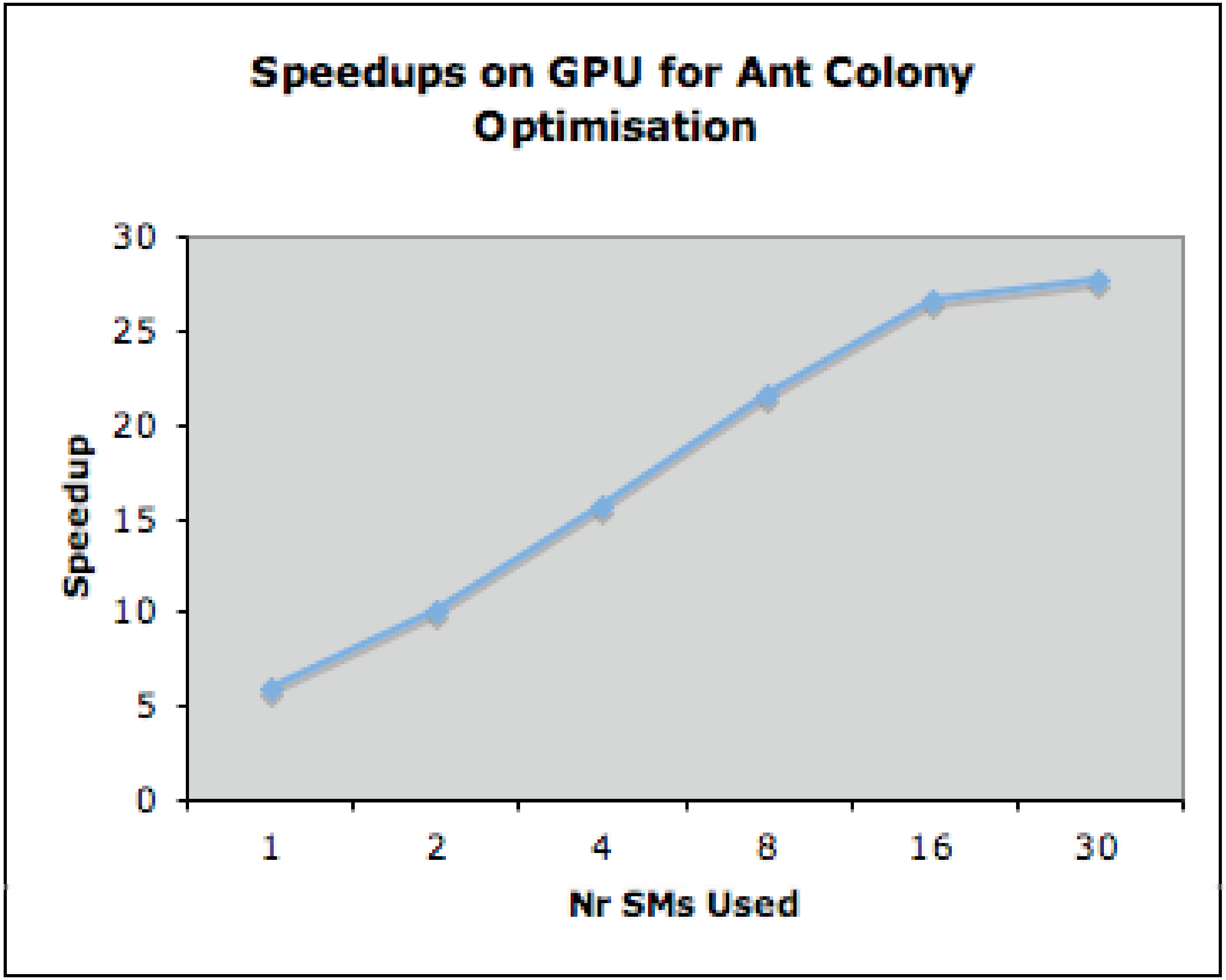
# GPU ACO, Refactored

```
Pipe = {pipe, [{farm, [{seqGPU, fun(X) -> lists:map(fun(Y) ->
                          binary_gpu_solve:find_solution(Y) end, X) end,
                          nGPUWorkers}],

          {seq, fun(X) -> pick_update_spawn_list_lists(Num_Workers,
                                                      Chunk_Size, X) end}]},
```

# GPU ACO, Refactored

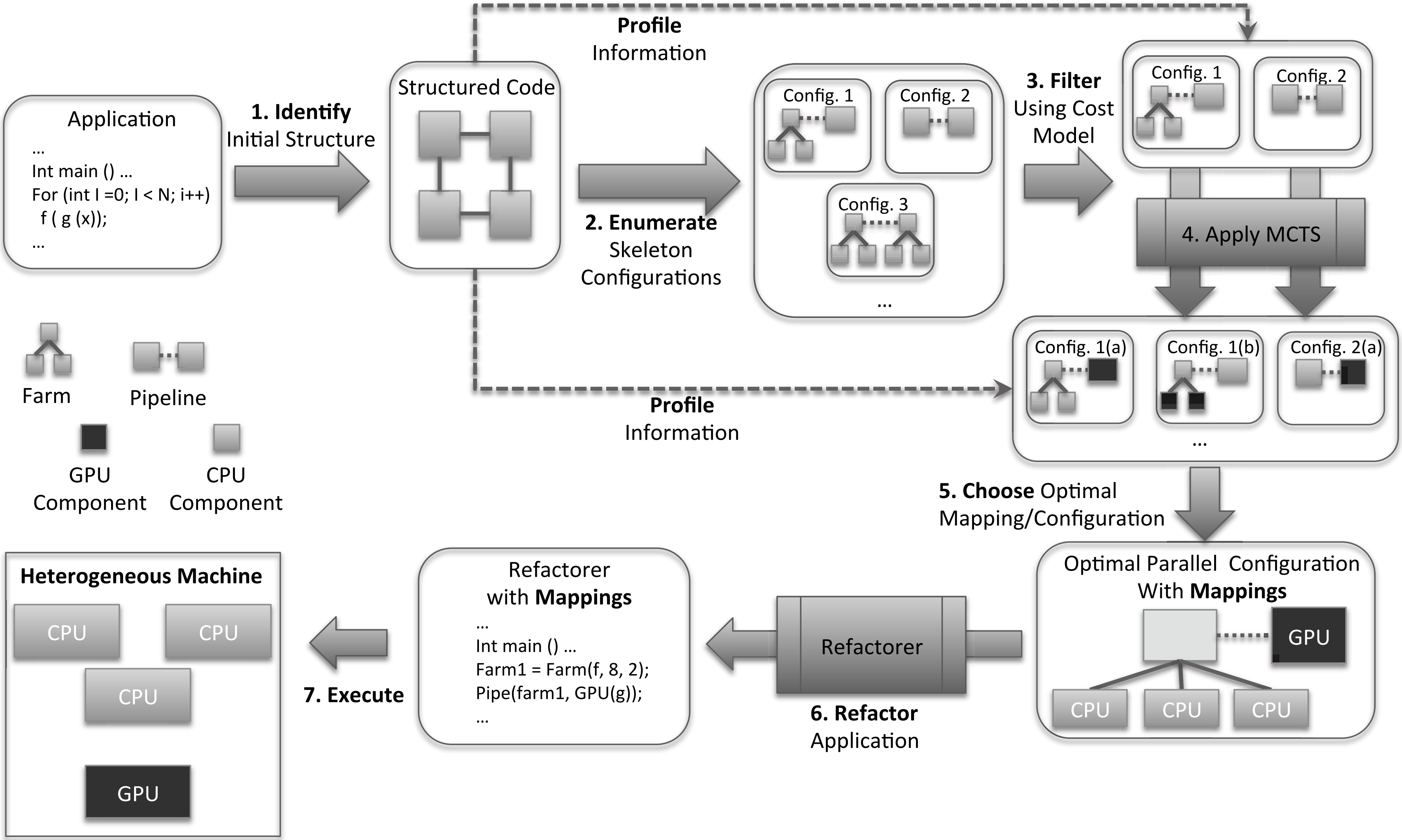
```
find_solution_gpu(...) ->
      E = clu:setup(all),
      {ok,Program} = clu:build_source(E, "solve2"),
      {ok,Kernel} = cl:create_kernel(Program, "solveKernel"),
      Random_Seed = 0,
      Tabus = list_to_tuple(lists:duplicate(Num_Jobs,1)),
Kws =
      map(
        fun(Device) ->
              {ok,Queue} = cl:create_queue(E#cl.context,Device,[]),
              {ok,Local} = cl:get_kernel_workgroup_info(Kernel,Device,
                                work_group_size),
              {ok,Freq} = ...
Kws3 = map(
          fun(K) ->
                {ok,ProcessTimeBuffer}  = cl:create_buffer(E#cl.context,[read_only],byte_size(Process_Time)),
 {ok,E1} = cl:enqueue_write_buffer(K#kwork.queue,
                                K#kwork.imem,
                                0, Nk,
                                K#kwork.idata, []),

ok = cl:set_kernel_arg(Kernel, 0, K#kwork.resultsBuffer),
            ...
                         Global = Count,
            {ok,E2} = cl:enqueue_nd_range_kernel(K#kwork.queue,
                                  Kernel,
                                  [Global], [K#kwork.local],
                                  [E1]),

                ...

                K#kwork { processTimeBuffer=ProcessTimeBuffer  ...  }
          end, Kws),
 Bs = map(
         fun(K) ->
               {ok,Bin} = cl:wait(K#kwork.e3),
               cl:release_mem_object(K#kwork.imem),
               cl:release_mem_object(K#kwork.omem),
               cl:release_queue(K#kwork.queue),
                          Bin
         end, Kws4),
```

# GPU Results

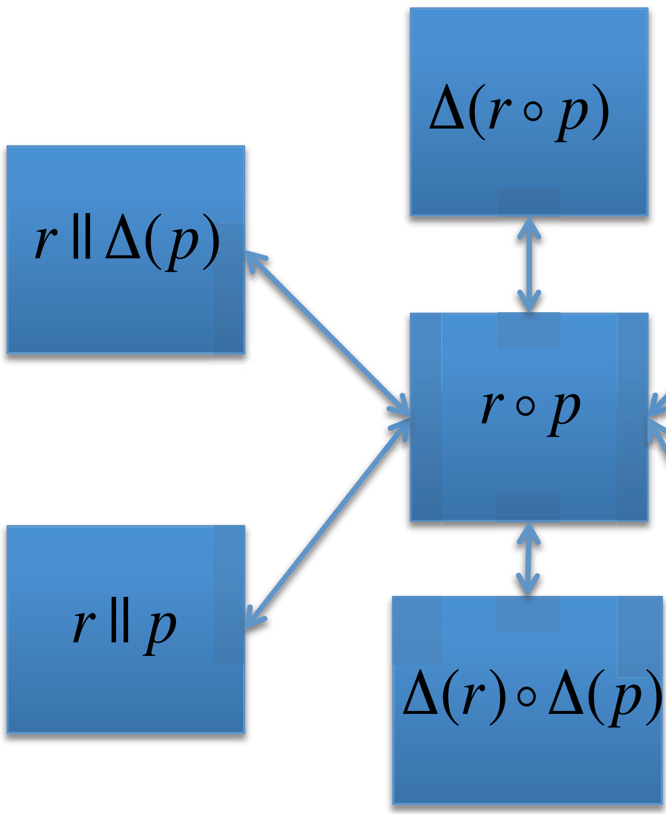


Speedups on GPU for Ant Colony Optimisation
Speedup
30
25
20
15
10
5
0
1
2
4
8
16
30
Nr SMs Used

# Heterogeneous Parallel Programming



# Example: Enumerate Skeleton Configurations for Image Convolution




| Configuration | Est. runtime |
|---|---|
| $r \circ p$ | 5.6 |
| $r \parallel p$ | 3.88 |
| $\mathbf{\Delta(r) \parallel p}$ | **1.60** |
| $r \parallel \Delta(p)$ | 4.00 |
| $\mathbf{\Delta(r) \parallel \Delta(p)}$ | **0.40** |
| $\mathbf{\Delta(r \parallel p)}$ | **0.56** |
| $\Delta(r) \circ \Delta(p)$ | 2.00 |
| $\Delta(r) \circ p$ | 2.00 |
| $r \circ \Delta(p)$ | 5.60 |

$r$ : read image file

$p$ : process image file

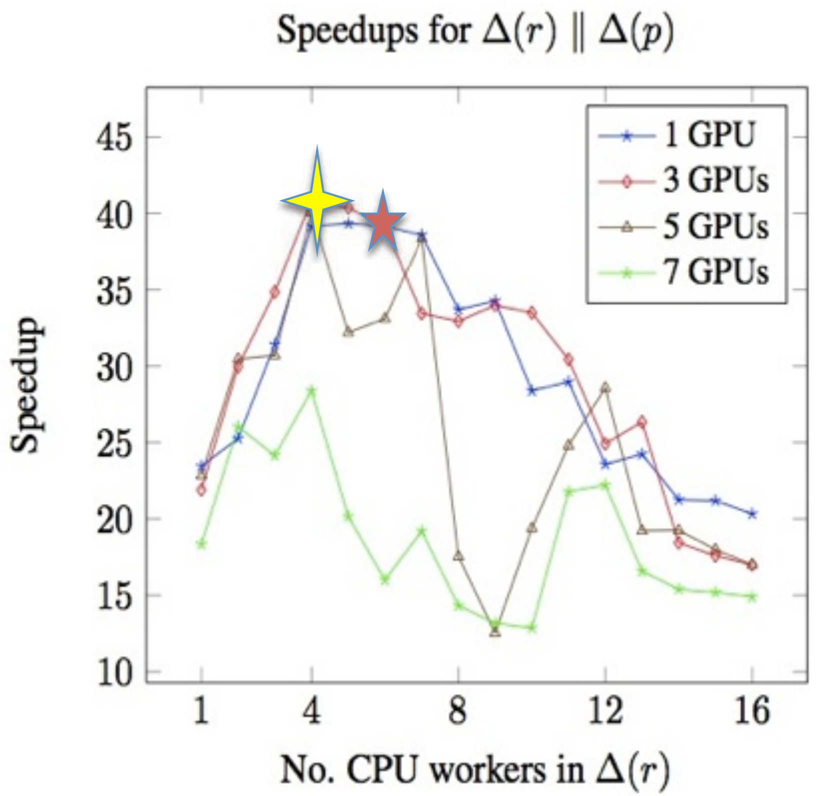
# Results on Benchmark: Image Convolution

MCTS Mapping (C, G):

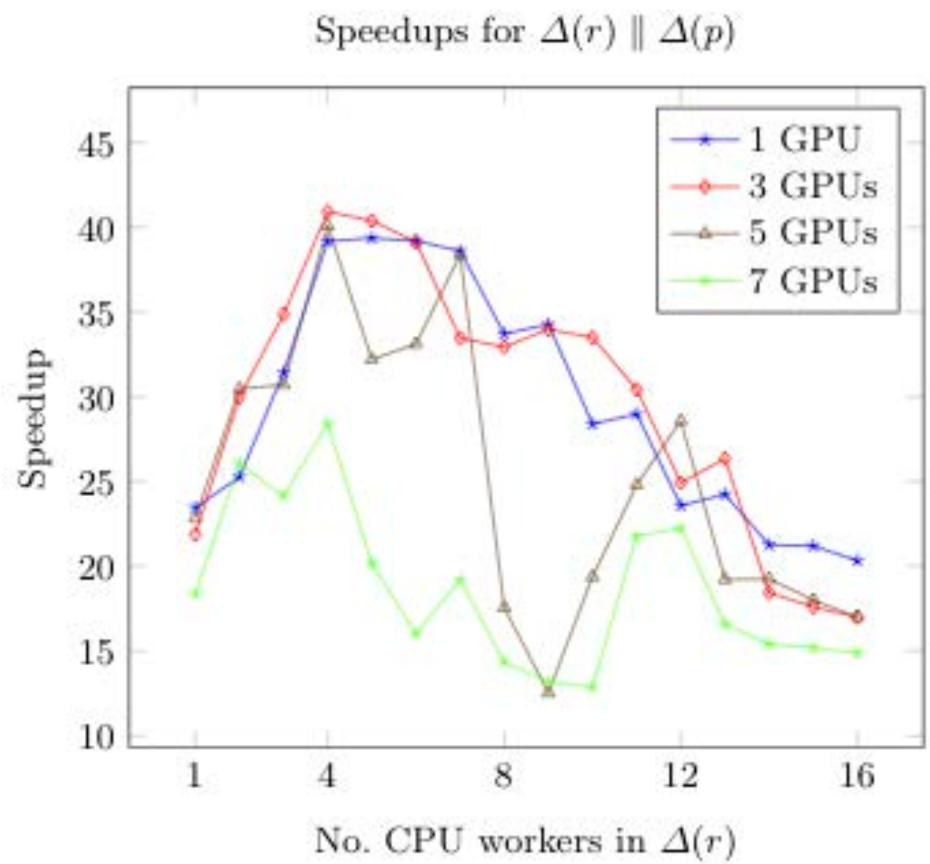
(6, 0) | | (0, 3)

**Speedup 39.12**

Best Speedup: 40.91



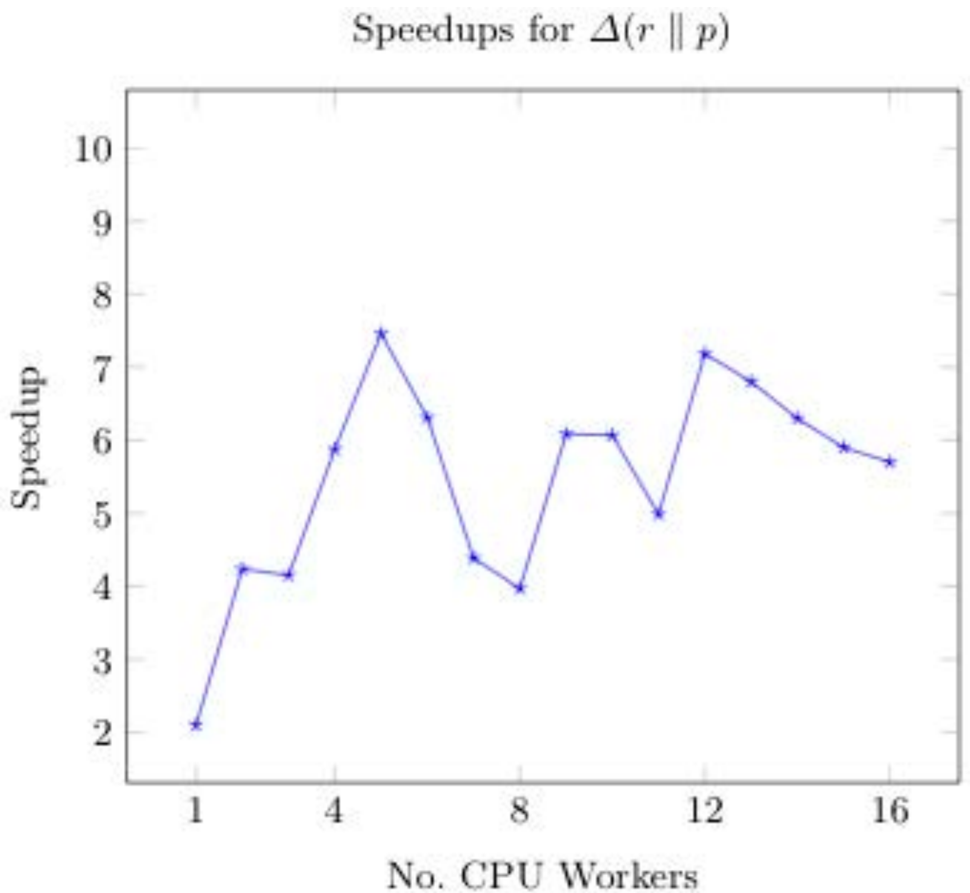
# Adding Mapping...




- The best speedup was predicted for $\Delta(r, 6, 0) \parallel \Delta(p, 0, 3)$

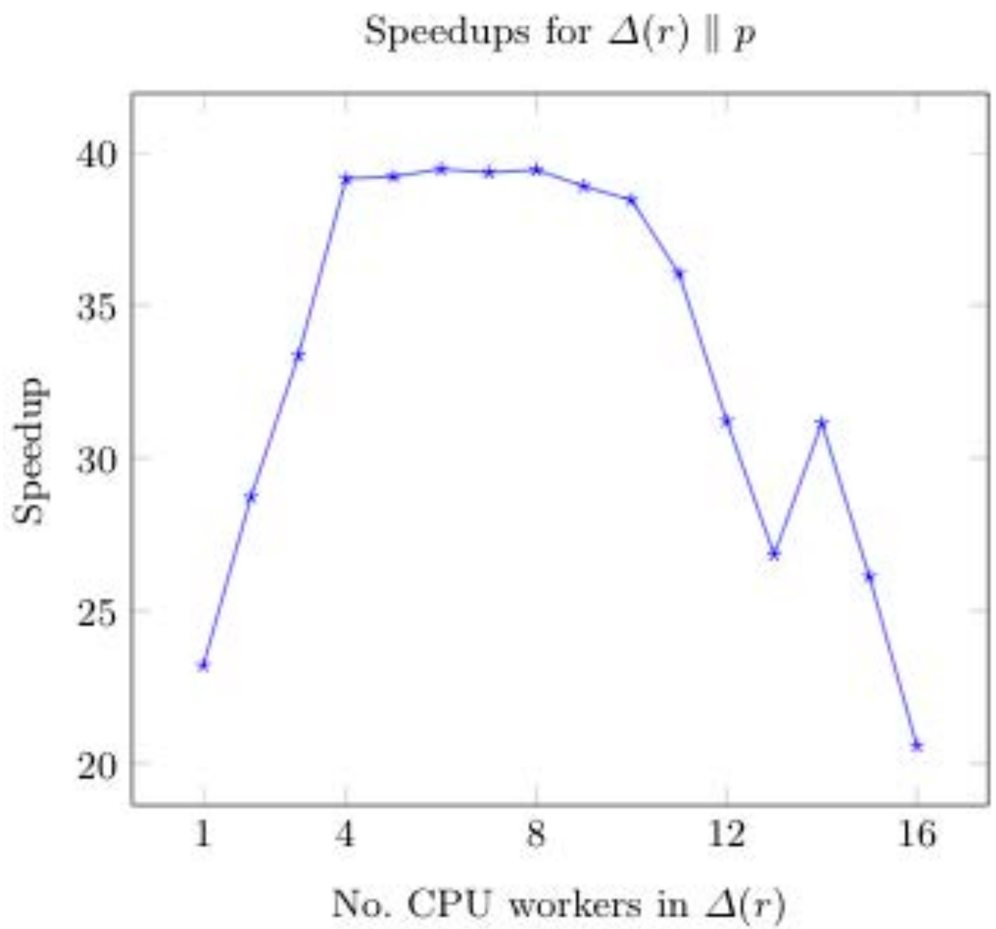
# Adding Mapping (2)

University of St Andrews




- The best speedup was predicted for $\Delta(r \parallel p, 5, 5)$

# Adding Mapping (3)




- The best speedup was predicted for $\Delta(r, 4, 0) \parallel p_G$

# Conclusions

- New heterogeneous skeletons for Erlang
- New Heterogeneous refactoring approach, semi-automatically introduces openCL bindings, and skeletal configuration
- Initial results for an ant colony optimisation
  - Skeletal, farm with feedback version, 12 speedup
  - GPU version, 26 speedup

# Conclusions

University of St Andrews

- The manycore revolution is upon us
  - Computer hardware is changing very rapidly (more than in the last 50 years)
  - The **megacore** era is here (aka exascale, BIG data)
- Heterogeneity and energy are both important
- Most programming models are too low-level
  - concurrency based
  - need to expose mass parallelism
- Patterns and *functional programming* help with abstraction
  - millions of threads, easily controlled

# Conclusions (2)

- Functional programming makes it easy to introduce parallelism
  - (Controlled) side effects means any computation could be parallel
  - Matches pattern-based parallelism
  - Much detail can be abstracted
- Lots of problems can be avoided
  - e.g. Freedom from Deadlock
  - Parallel programs give the same results as sequential ones!
- Automation is very important
  - Refactoring dramatically reduces development time
    (while keeping the programmer in the loop)
  - Machine learning is very promising for determining complex performance settings

# Future Work

- Allow further integration into skeletons
  - A living mixture of CPU/GPU components
- Wider range of skeletons
  - Parallel workpools
  - Divide-and-conquer
  - Map-reduce
  - BSP
- More case studies, and from different domains:
  - Physics, computer algebra, …
- Include dynamic remapping and distributed computing environments

# Funded by

- **ParaPhrase (EU FP7), Patterns for heterogeneous multicore,**
  €4.2M, 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

# Some of our 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

# ParaPhrase Needs You!

- Please join our mailing list
  and help grow our user community
  - news items
  - access to free development software
  - chat to the developers
  - free developer workshops
  - bug tracking and fixing
  - Tools for both Erlang and C++

- Subscribe at

  https://mailman.cs.st-andrews.ac.uk/mailman/listinfo/paraphrase-news

- We're also looking for open source developers...

# THANK YOU!

http://www.paraphrase-ict.eu

http://www.project-advance.eu

*@paraphrase_fp7*