University of St Andrews

SEVENTH FRAMEWORK PROGRAMME

Predictable Timing Analysis of x86 Multicores using High-Level Parallel Patterns

Kevin Hammond, Susmit Sarkar and Chris Brown

University of St Andrews, UK

T: *@paraphrase_fp7*
E: *kh@cs.st-andrews.ac.uk*
W: **http://www.paraphrase-ict.eu**

cost

EPSRC

sicsa*

PARAPHRASE

Motivation

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 no other advances will matter!**
- **We need to produce predictable timing models for widely used multicores (e.g. x86, ARM)**

Even Mobile Phones are Multicore!

University of St Andrews

cost

ADVANCE StatArch

ParaPhrase

Current Parallel Methodologies

University of St Andrews

- **Applications programmers *must be systems programmers***
  - insufficient assistance with abstraction
  - too much complexity to manage
- **Difficult/impossible to scale, unless the problem is simple**
- **Difficult/impossible to change fundamentals**
  - scheduling
  - task structure
  - migration
- **Many approaches provide *libraries***
  - **they need to provide abstractions**

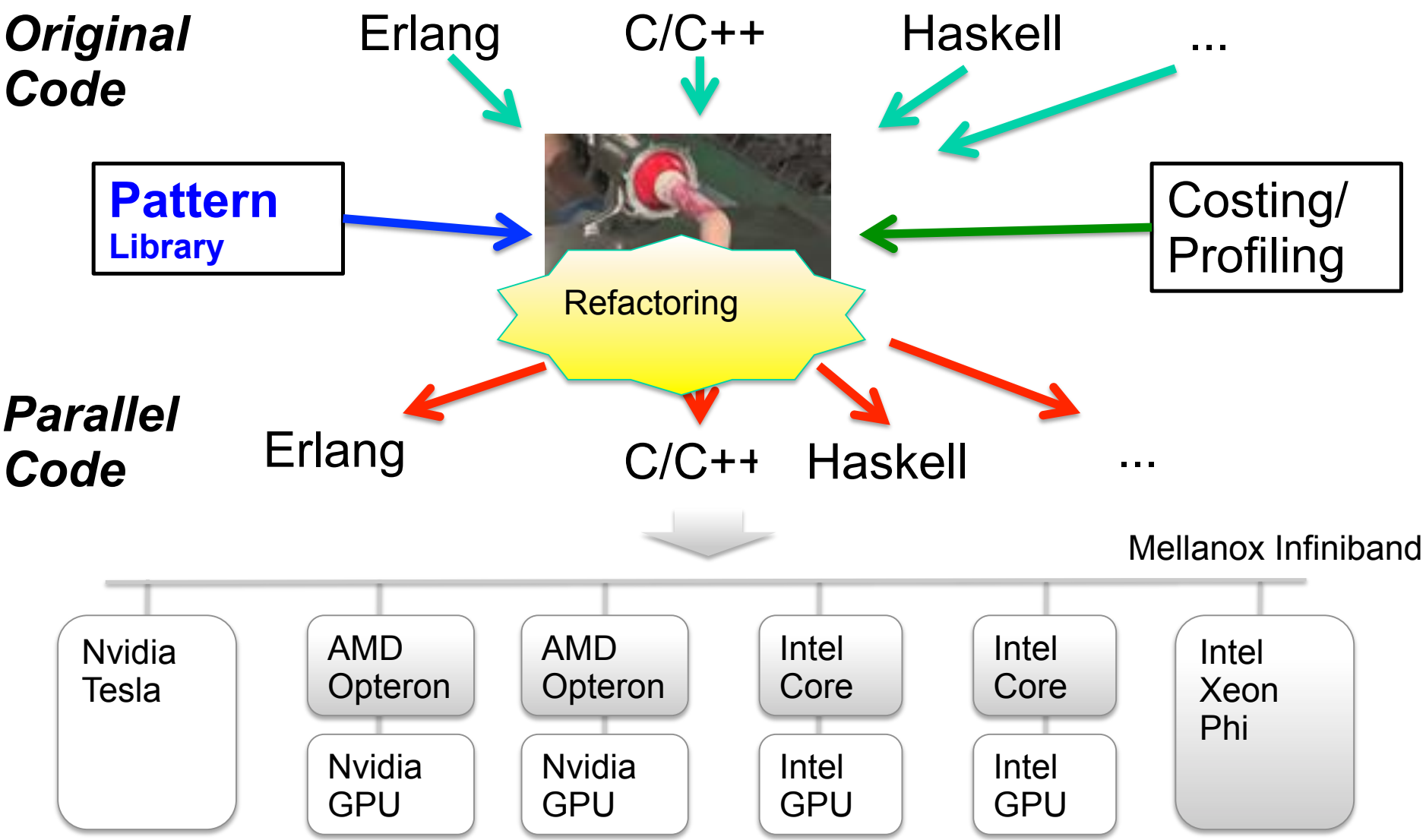
Thinking Parallel

University of St Andrews

- **Fundamentally, programmers must learn to "think parallel"**
  - this requires new *high-level* programming constructs
    - perhaps dealing with large numbers 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!

The ParaPhrase Approach

University of St Andrews



cost

ADVANCE StatArch

ParaPhrase

Components and Abstraction

University of St Andrews

- **Components give some of the advantages of functional programming**
  - clean abstraction
  - pure computations, easily scheduled
  - dependencies can be exposed
- **Hygiene/discipline is necessary**
  - no unwanted state leakage
    (e.g. in terms of implicit shared memory state)

The ParaPhrase Approach

University of St Andrews

- **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

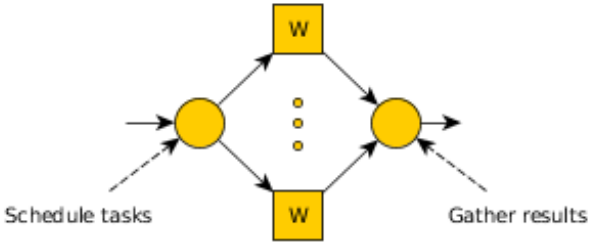
- **RESTRUCTURE/TUNE if necessary! (also using refactoring)**

Some Common Parallel Patterns

University of St Andrews

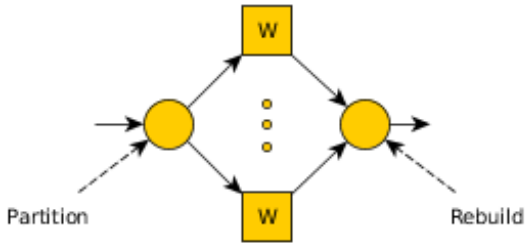
Generally, we need to nest/ combine patterns in arbitray ways

Farm

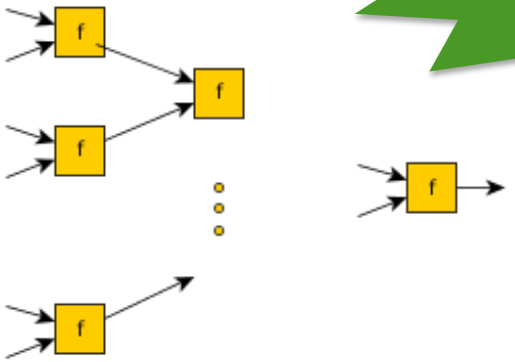


Pipeline

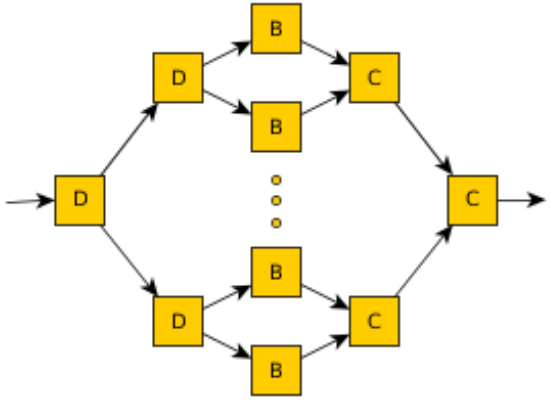
Map



Reduce



Divide&Conquer



Skeletons

University of St Andrews

- **Skeletons are *implementations* of parallel patterns**
- **A skeleton is a template**
  - pluggable *higher-order* functions
  - can be instantiated with concrete worker functions
- **Skeletons *avoid* deadlock, race conditions**
  - communication is implicit and structured

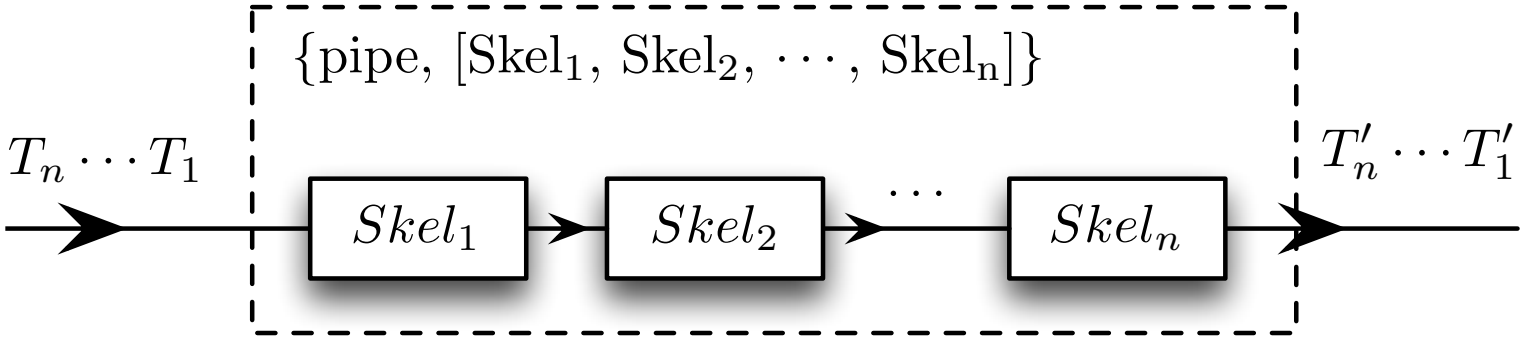
Murray Cole, "Algorithmic Skeletons: structured management of parallel computation" MIT Press, 1989

Horacio González-Vélez and Mario Leyton:
A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers.
Softw., Pract. Exper. 40(12): 1135-1160 (2010)

Parallel Pipeline Skeleton

University of St Andrews

- Each stage of the pipeline can be executed in parallel
- The input and output are streams
- Each stage is itself an instance of a pattern (Skel)

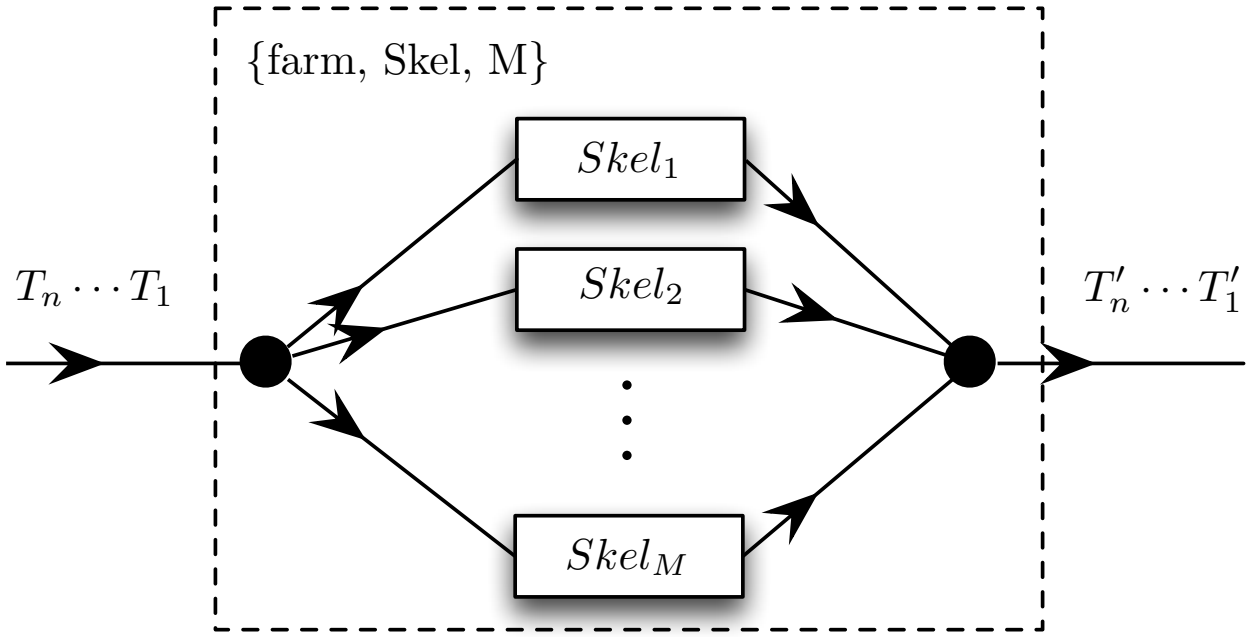



```
skel:do([{pipe,[Skel1, Skel2,..,SkelN]}], Inputs).
```

Parallel Task Farm Skeleton

University of St Andrews

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




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

Example Parallel Structure

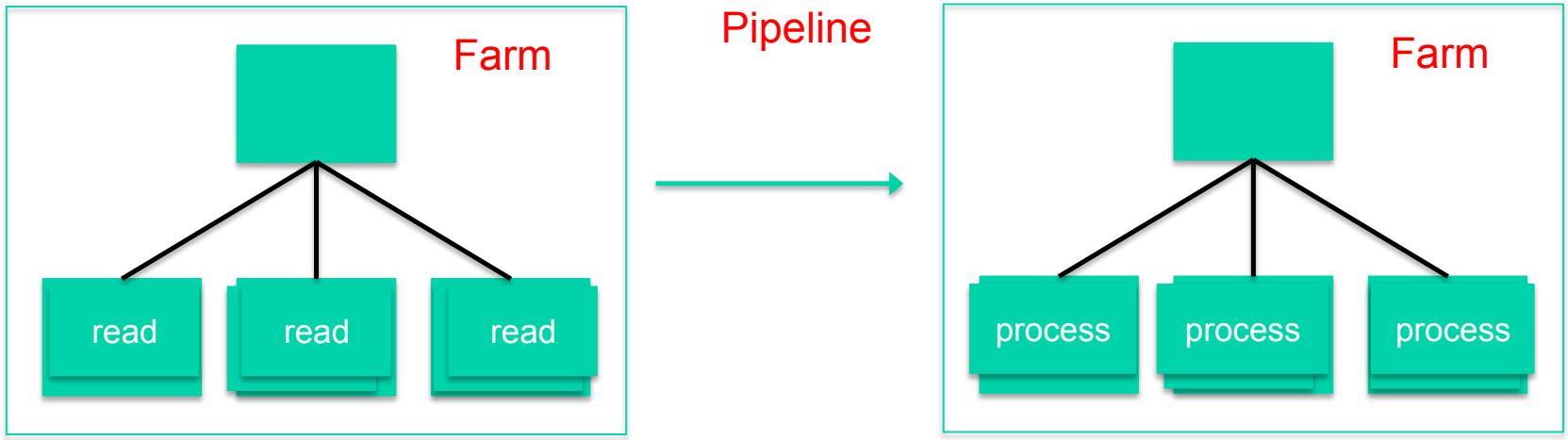
University of St Andrews

Sequential

```
for each image, i.
    process(read i)
```

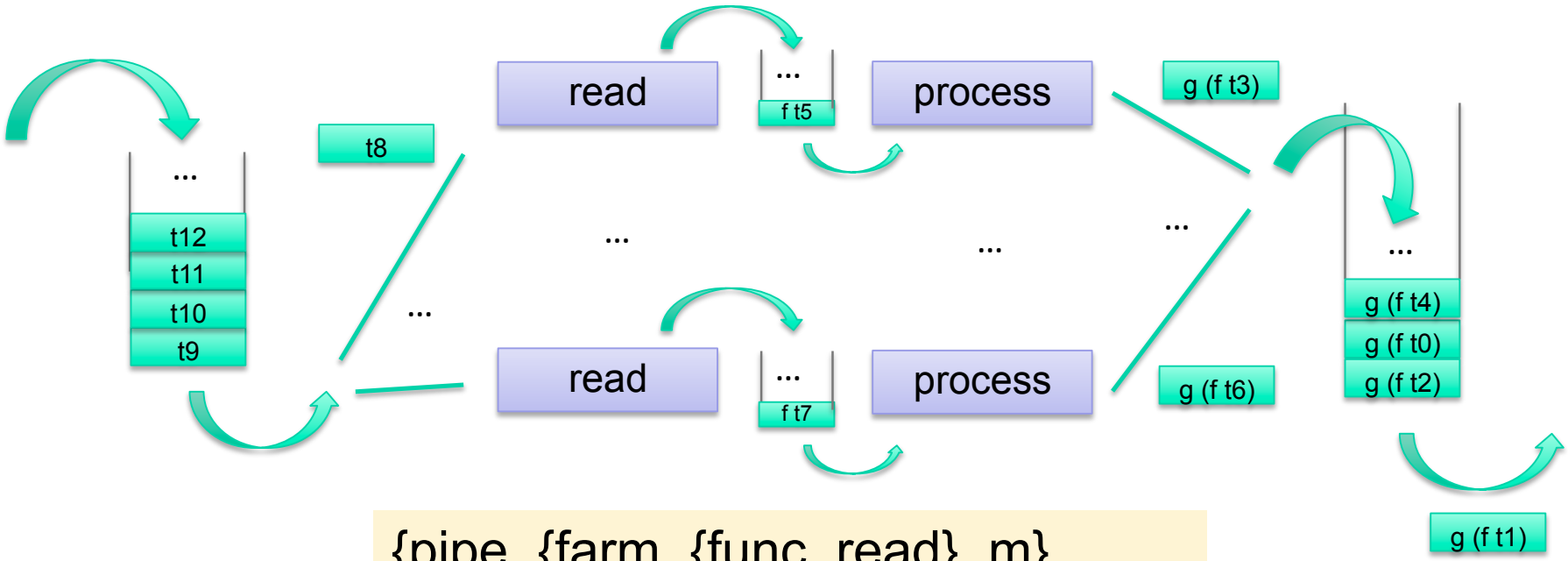
Parallel

```
{pipe, {farm, {func, read}, m},
       {farm, {func, process}, n}}
```



Composing Skeletons

University of St Andrews



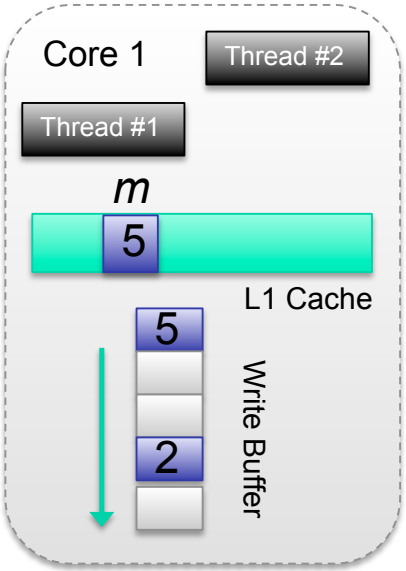

```
{pipe, {farm, {func, read}, m},
       {farm, {func, process}, n}}
```

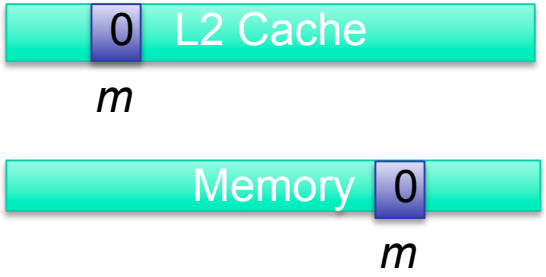
- **Queues link skeletons**

x86 Multicore Cache Design

University of St Andrews

- **Each core has**
  - a local write-back cache
  - a FIFO-ordered ***write buffer***
- **A core may run many threads**
- **Cores share**
  - level 2 (and 3) cache
  - global memory





Sequential Consistency (SC)

University of St Andrews

```
1  int x = 0, y = 0;
2
3    ...
4
5  { // thread 1                    { // thread 2
6      x = 1;                         y = 2;
7    return(x+y);                   return(x+y);
8  }                                }
```

- **Memory accesses are effectively interleaved**
  - as if run by a single processor
- **Either**
  - both threads return 3
  - thread 1 returns 1, thread 2 returns 3
  - thread 1 returns 3, thread 2 returns 2

Not

- thread 1 returns 0, thread 2 returns 0

x86 Total Store Order (TSO)

University of St Andrews

- **On a multicore, SC can be inefficient**
- **Intel uses a weaker (relaxed memory) consistency model**
  - Total store order (TSO) guarantees that the order that **Writes** are seen by a location is the same as the order they were issued
- **ARM uses an even weaker consistency model**

Basic TSO Rules

University of St Andrews

- **The basic rules are:**
  - (1) **Reads** are not reordered with other **Reads**.
  - (2) **Writes** are not reordered with older **Reads**.
  - (3) **Writes** are not reordered with other **Writes**.
  - (4) **Reads** may be reordered with older **Writes** to different memory locations but not with older **Writes** to the same memory location

- **An Exchange is treated as an indivisible Read/Write pair to a specific memory location**

- **A Fence is treated as both a Read and Write to all memory locations, except that no actual memory transfer occurs**

Simple Spin Lock Implementation

University of St Andrews

```
1  void lock( volatile char *lockcell ) {
2     char old_value ;
3
4     do {
5        old_value = exchange(lockcell,1);
6     } while ( 1 == old_value ) ;
7  }
8
9
10
11 void unlock( volatile char *lockcell ) {
12    *lockcell = 0 ;
13 }
```

x86 Assembly code for spin lock

University of St Andrews

```
1  lockr:
2    push ebp                ; Start new stack frame
3    mov  ebp, esp
4    mov  ebx, [ebp+8]       ; Get address of lock cell
5
6  trylock:
7    mov  eax, 1             ; Set EAX register to 1 (locked)
8    xchg eax, [ebx]         ; Exchange EAX and lock cell
9    test eax, eax           ; Test whether the cell is already locked
10   jnz  trylock            ; Retry the lock if so
11
12   pop ebp                 ; revert stack frame
13   ret                     ; The lock has been acquired
14
15 unlockr:
16   push ebp                ; Start new stack frame
17   mov  ebp, esp
18   mov  ebx, [ebp+8]       ; Get address of lock cell
19
20   mov  eax, 0             ; Set EAX register to 0 (unlocked)
21   mov  [ebx], eax         ; Release the lock
22
23   pop ebp                 ; revert stack frame
24   ret                     ; The lock has been released.
```

Simple Queue using spin lock

University of St Andrews

```
1  Value qget(Queue q) {
2    Value v;
3
4    do {
5      lock(&q.lock_cell);
6
7      if (qempty(q))
8        break;
9
10     unlock(&q.lock_cell);
11   } while (1);
12
13   /* lock is held */
14   v = front(q);
15
16   unlock(&q.lock_cell);
17
18   return(v);
19 }
```

```
21 void qput(Queue q,Value v) {
22   lock(&q.lock_cell);
23
24   addtoq(q,v);
25
26   unlock(&q.lock_cell);
27 }
```

We have used HOL to prove that this is sound wrt the TSO relaxed-memory model

cost

ADVANCE StatArch

ParaPhrase

Simple Timing Model

University of St Andrews

- The worst-case costs if ***n*** threads contend a lock are

$$T_{\texttt{qput}} = n \cdot T_{Exchange} + T_{Write} + T_{Write}$$

$$T_{\texttt{qget}} = n \cdot T_{Exchange} + T_{Read} + 2T_{Write}$$

Timing Model for a Farm

University of St Andrews

- **The amortised average cost for each farm operation is**

$$T_{\mathtt{qget}} + T_f + T_{\mathtt{qput}}$$

which simplifies to:

$$2 \cdot (n+1) \cdot T_{Exchange} + 5 \cdot T_{Write} + T_{Read} + T_f$$

Timing Model for a Pipeline

University of St Andrews

- **If the first stage dominates (function *f*), its cost is**

$$T_{\texttt{qget}} + T_f + T_{\texttt{qput}}$$

which simplifies to:

$$2 \cdot (|f| + 1) \cdot T_{Exchange} + 5 \cdot T_{Write} + T_f$$

- **The total cost for both stages is therefore:**

$$2 \cdot (|f| + 1) \cdot T_{Exchange} + 5 \cdot T_{Write} + T_f + T_g$$

or, if the second stage dominates (function *g*)

$$T_f + (2 \cdot |g| + |f| + 1) \cdot T_{Exchange} + 5 \cdot T_{Write} + T_g$$

Including Store-Buffer Flushing

University of St Andrews

- The cost of an exchange depends on items to be flushed, *b*

$$T_{Exchange} = b \cdot T_{Fl} + T_{JustX}$$

- The cost of a spin-lock on *t* contending threads is

$$b \cdot T_{Fl} + t \cdot T_{JustX}$$

- The costs of queue operations change slightly

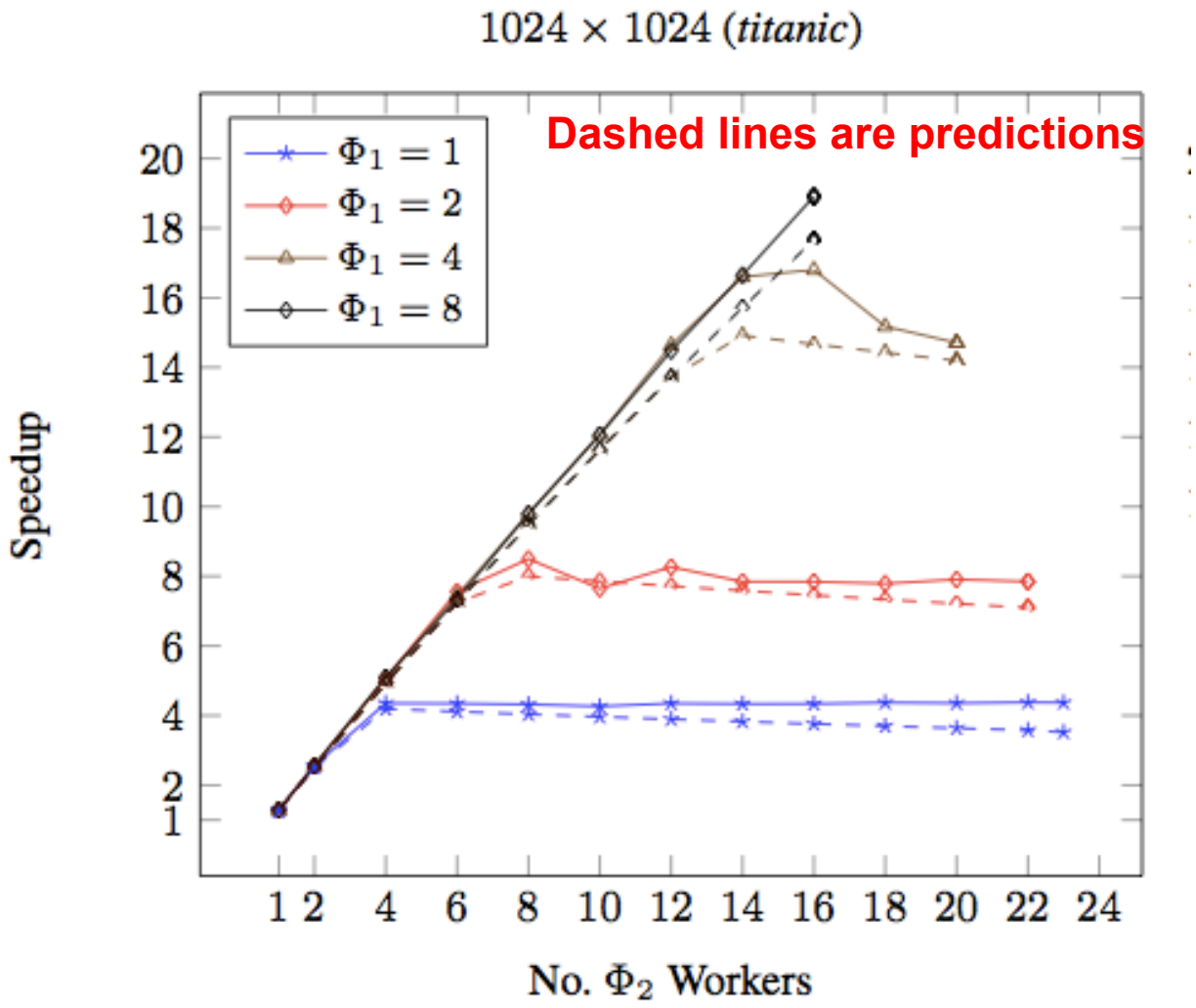
$$T_{\mathtt{qput}} = b \cdot n \cdot T_{Fl} + n \cdot T_{JustX} + T_{Write} + T_{Write};$$

$$T_{\mathtt{qget}} = b \cdot n \cdot T_{Fl} + n \cdot T_{JustX} + T_{Read} + 2 \cdot T_{Write};$$

- The cost of a farm is:

$$2 \cdot (n+1) \cdot b \cdot T_{Fl} + (n+1) \cdot T_{JustX} + 4T_{Write} + T_{Read} + T_f$$

Performance Predictions (Image Convolution, 1024x1024)



24 core machine at Uni. Pisa

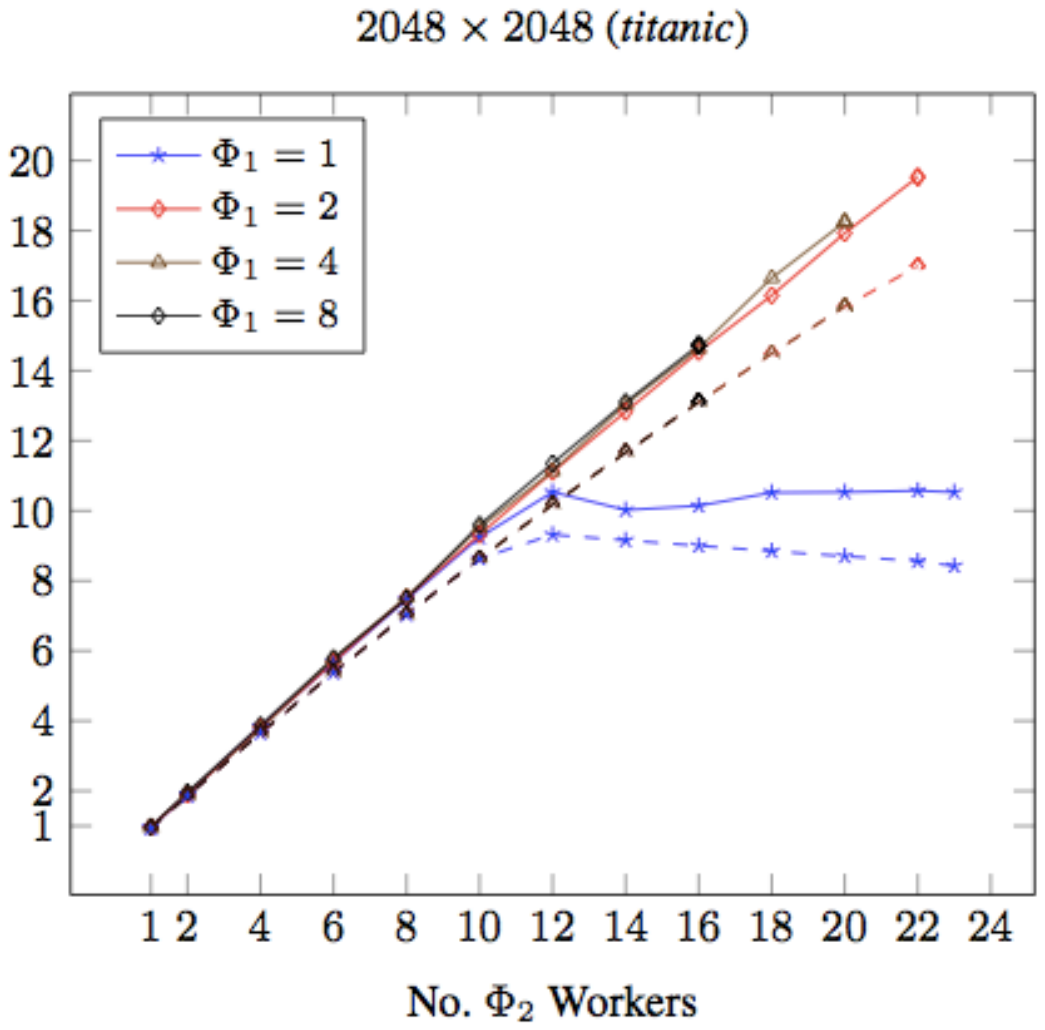
2xAMD Opteron 6176. 800 Mhz

32GB RAM

1 x NVidiia Tesla C2050 GPU

Performance Predictions (Image Convolution, 2048x2048)

University of St Andrews



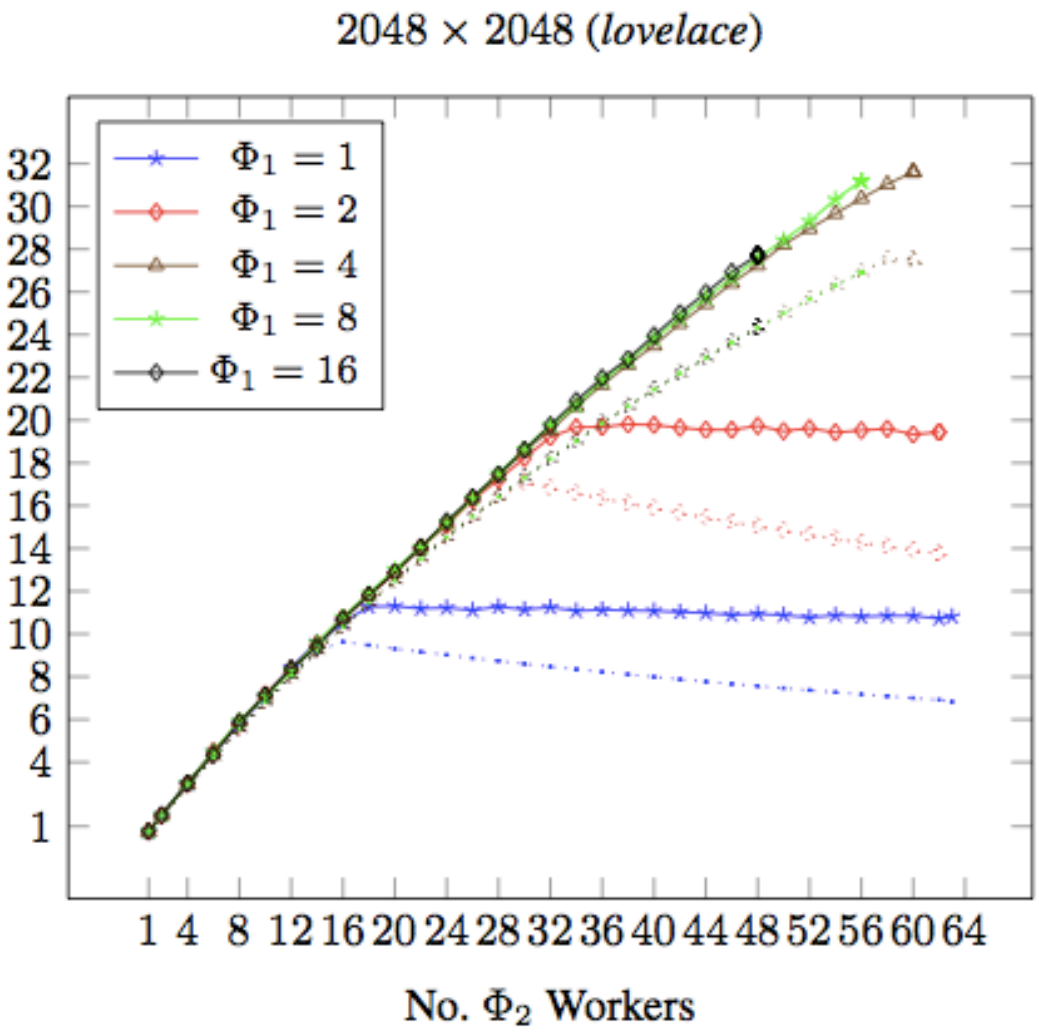
cost

ADVANCE StatArch

PARAPHRASE

Performance Predictions (Image Convolution, 2048x2048)

University of St Andrews



64-core machine at Uni. St Andrews

8xAMD Opteron 6376. 2.3Ghz

32GB RAM

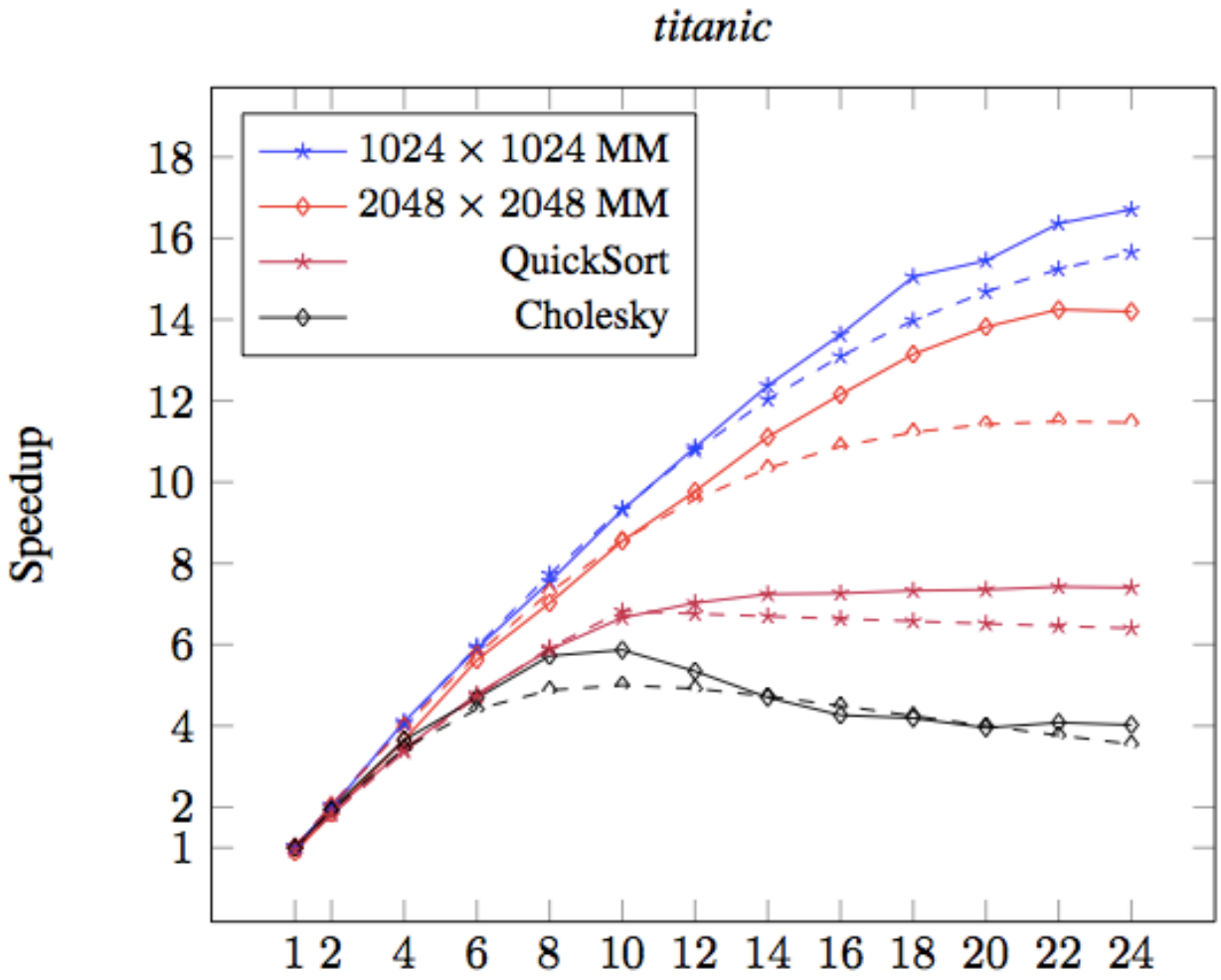
cost

ADVANCE StatArch

ParaPhrase

Performance Predictions (Matrix Multiplication etc)

University of St Andrews



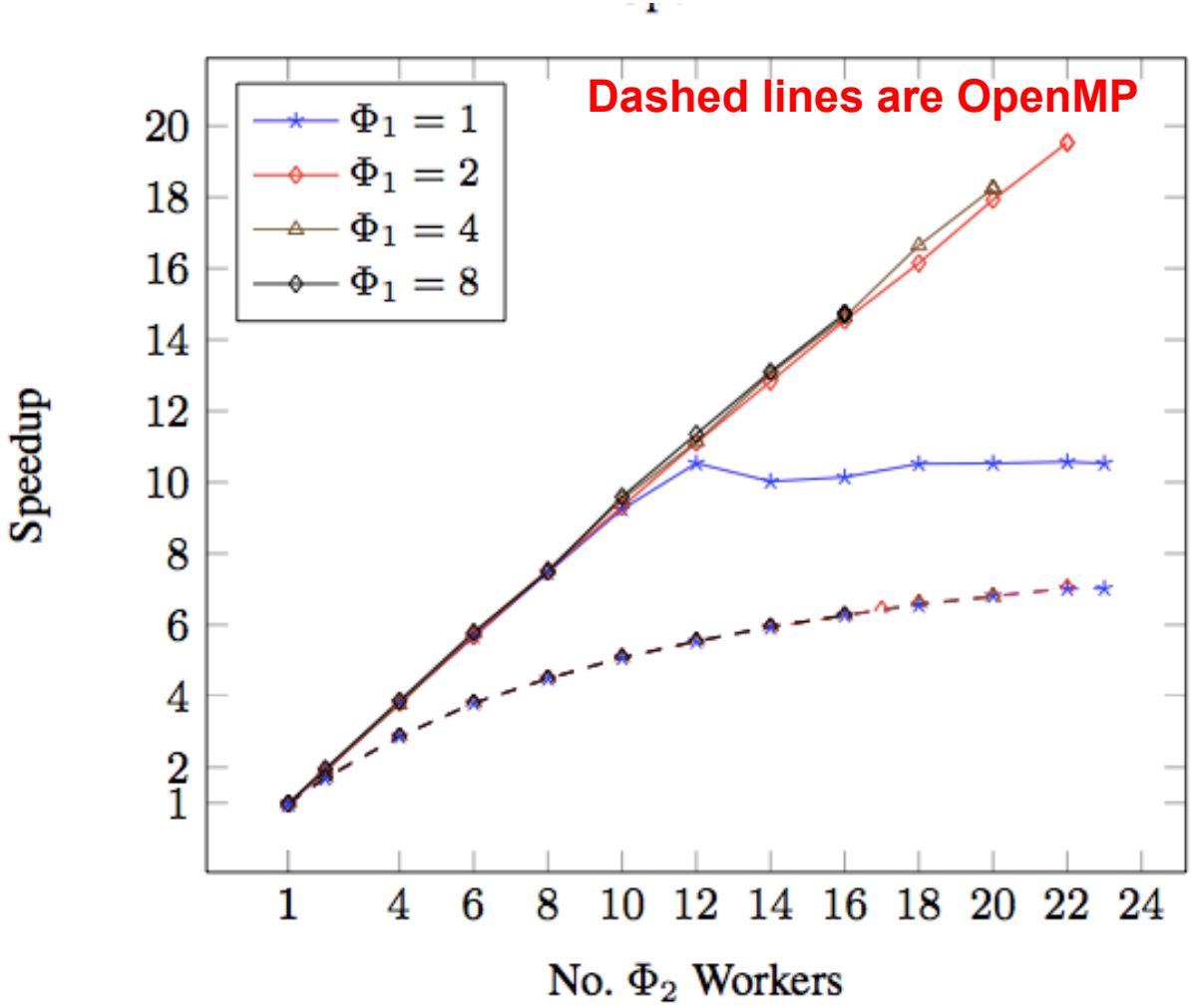
cost

ADVANCE StatArch

PARAPHRASE

Comparison with OpenMP

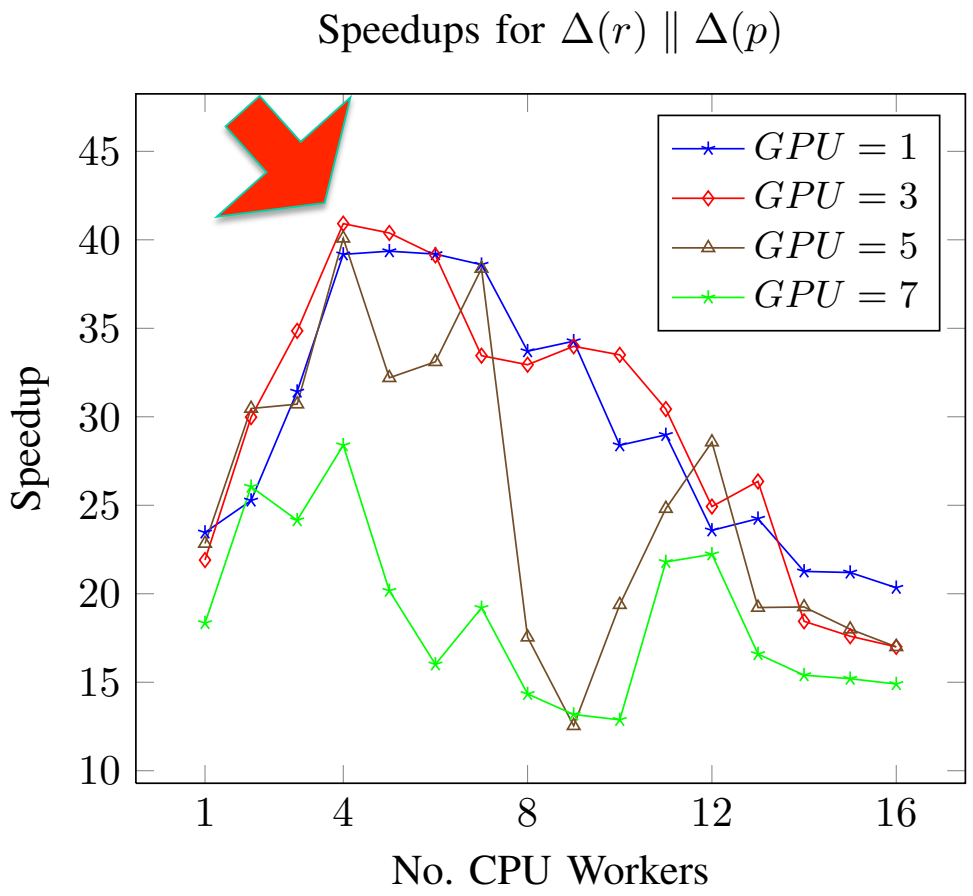
University of St Andrews



Combining CPUs and GPUs

University of St Andrews

- **Machine Learning chooses**
  - best combination of patterns
  - CPU/GPU allocations
- **Excellent Results**
  - within 5% of optimal
  - > 40x speedup **over sequential CPU**



Lock-Free Queue Implementation

University of St Andrews

```
16 Value qgetlf(Queue q) {
17   NodePtr first;
18
19   do {
20     if (qempty(q))
21       continue;
22
23     first = q->first;
24
25     if(first==NULL)
26       continue;
27
28   }  while (!dcas(q->first,first,first->next));
29
30   return(first->value);
31 }
```

- **This uses a double compare-and-swap variant of Exchange**
  - atomically swaps two values
  - allows us to avoid ABA errors by including a count field

Lock-Free Queue Implementation

University of St Andrews

```
34 void qputlf(Queue q,Value v) {
35   Node n = new Node(v);          // new queue node
36   NodePtr np = new NodePtr(n); // queue pointer
37   Queue last;
38
39   do {
40     last = q->last;
41
42     np.count = last.count+1;
43
44   } while(!dcas(q->last,last,np));
45 }
```

- **At the pattern level, this is plug-replaceable with a lock**
  - The cost model needs to change but most details are the same
  - All proof is the same above the lock-free level

Comparison of Development Times

University of St Andrews

	Man.Time	Refac. Time
Convolution	3 days	3 hours
Ant Colony	1 day	1 hour
BasicN2	5 days	5 hours
Graphical Lasso	15 hours	2 hours

Conclusions

University of St Andrews

- **High-level Patterns help structure parallel computations**
  - avoid deadlock, race conditions etc (formal proof in paper!)
  - reduce development time by an order of magnitude
  - allow us to construct predictable cost models
- **Cost model for x86 constructed from first principles**
  - Predictable timings for x86 (provably correct from TSO semantics)
  - Highly Accurate
  - All previous formal models have been for much simpler memory models (e.g. PPC)
- **Proved to be deadlock-free**
- **Applicable to energy as well as time**

cost ADVANCE StatArch PARAPHRASE

Funded by

University of St Andrews

- **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 /**

  •€300K, 2012-2015

Some of our Industrial Connections

Mellanox Inc.

Erlang Solutions Ltd

SAP GmbH, Karlsrühe

BAe Systems

Selex Galileo

Biold GmbH, Stuttgart

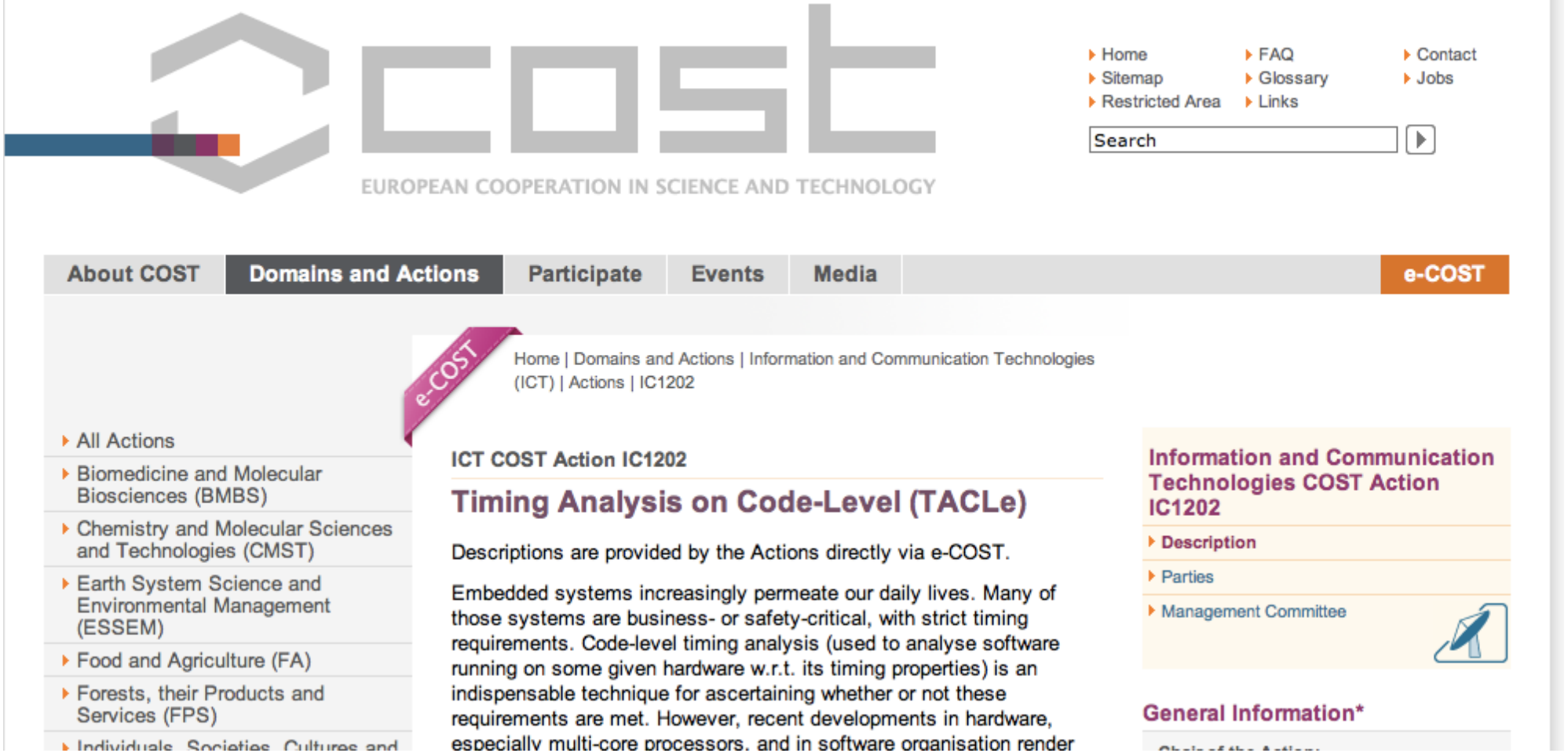
Philips Healthcare

Software Competence Centre, Hagenberg

Microsoft Research

Well-Typed LLC

University of St Andrews



cost

ADVANCE StatArch

PARAPHRASE

University of St Andrews

THANK YOU!

http://www.paraphrase-ict.eu

http://www.project-advance.eu

@paraphrase_fp7