Lapedo: A Hybrid Skeletal Framework for Programming Heterogeneous Machines in Erlang

Chris Brown, Vladimir Janjic, Adam Barwell, Kevin Hammond

University of St Andrews, Scotland

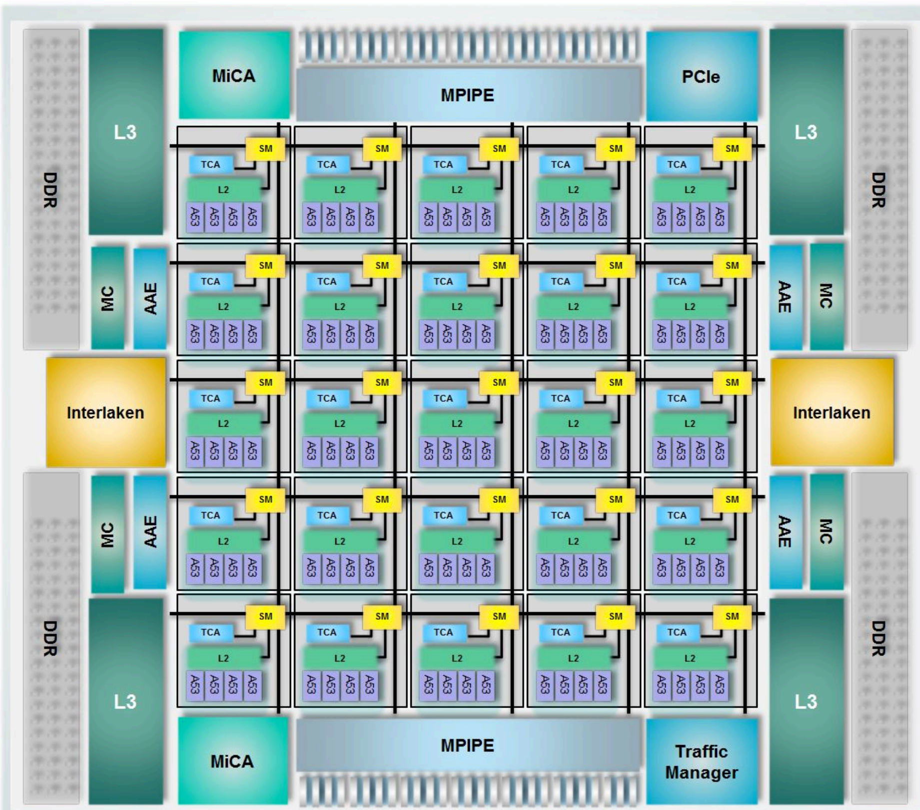
@chrismarkbrown

@rephrase_eu

Dublin – September 2015

The Dawn of a New Age

University of St Andrews



- EZCHIP – TILE-MX100
- 100 64-bit AMD x86 Cores
- 3-level cache with > 40 Mbytes on-chip cache
- Multitude of network accelerators
- Over 200Gbps integrated I/O including ethernet,
- DDR supports up to 1 TB RAM

Heterogeneous Multicores…

University of St Andrews

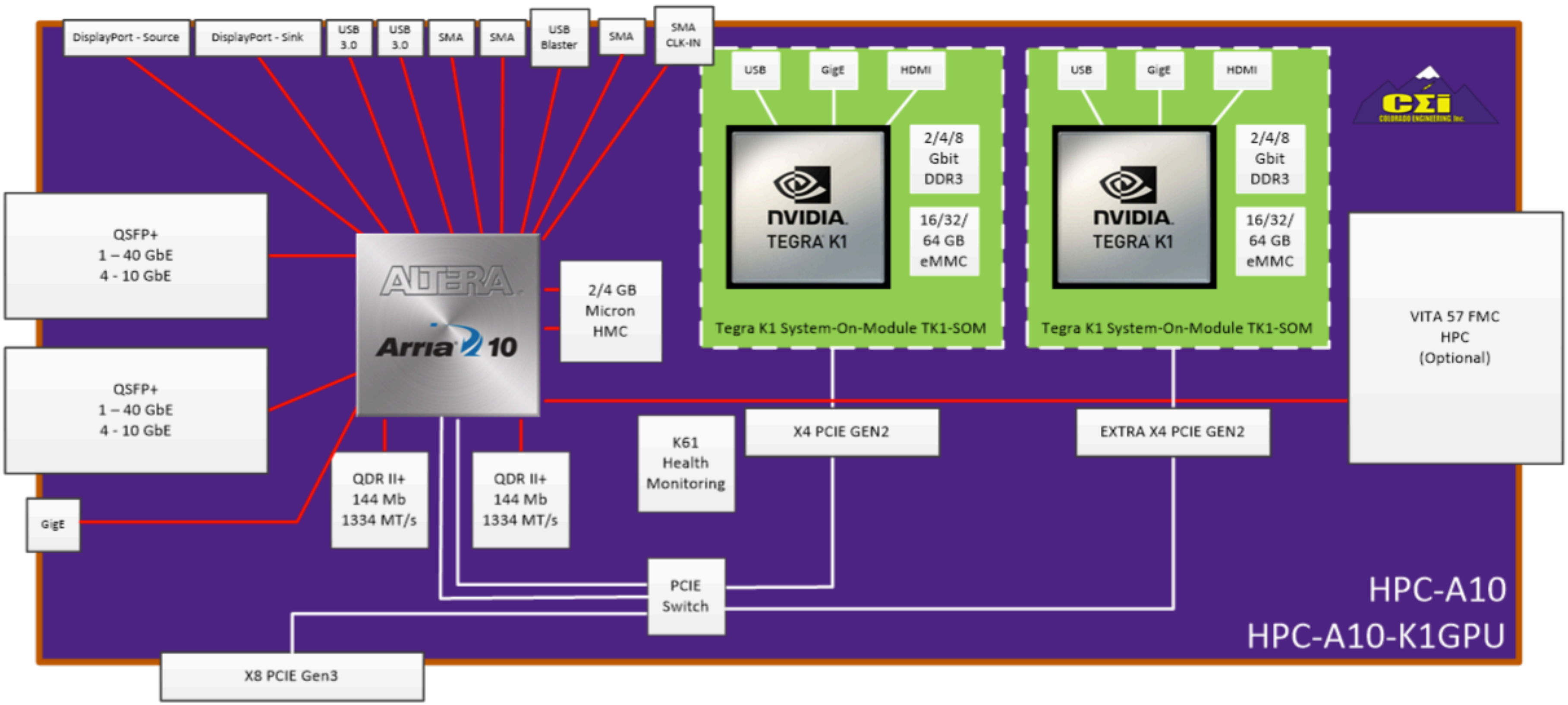
- …are everywhere!



- Orders of magnitude more performance than traditional CPU-only systems

- Much better performance-per-watt (important for mobile and embedded systems)

The Future is Interesting…



Programing Heterogeneous Systems…

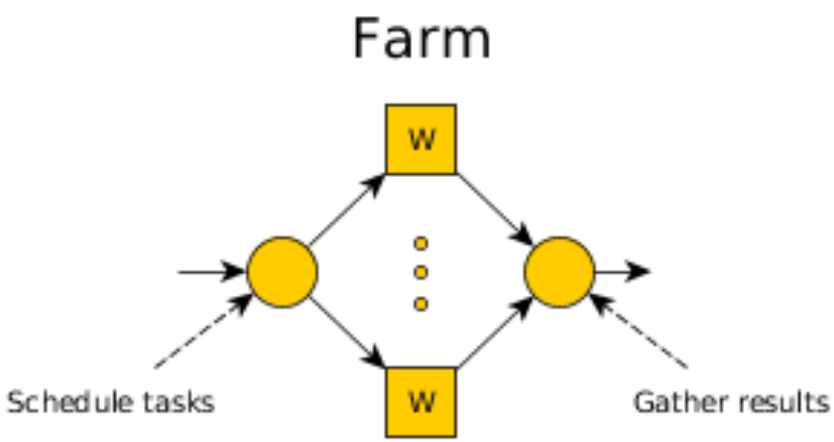
Programing Heterogeneous Systems…

- …is hard!
  - Much harder than CPU only parallelism
- Mainstream programming models (e.g. OpenCL, CUDA+threads) are too low-level for an average programmer
- Many applications can be parallelised in more than one way
- Dividing the work up between CPUs and accelerators is difficult
- Choosing which parallel structure to exploit is a non-trivial problem

Parallel Skeletons

Offer high-level of abstraction of a parallel behaviour

- Basically a “higher-order function” for parallelism



The *Skel* Library

- skel.weebly.com
- The only Skeleton library for Erlang

- Provides fundamental and most popular skeletons

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

`Skeleton` = a skeleton

`InputItems` = list of items to be processed

`OutputItems` = result of skeleton

The *Skel* Library for Erlang

A small set of classical skeletons for Erlang, operating over streams of inputs on CPUs

- *Func* – a wrapper around a sequential function
  - `{func, f}`
- *Pipe* – parallel composition of skeletons
  - `{pipe, {func, f}, {func, g}, {func, h} }`
- *Farm* – same operation over a set of inputs
  - `{farm, {pipe, {func f}, {func g}}, NrWs}`
- Many more…

OpenCL Bindings for Erlang

University of St Andrews

- Provides functions to execute OpenCL kernels from Erlang
  - Enables calling C OpenCL functions
- Basic marshalling between Erlang *binaries* and C arrays
  - Functions to copy binary data to/from the accelerator memory
- Enables Erlang programmers to write code for accelerators, **but it does not simplify accelerator programming**
  - Programmer still needs to write as much code as with C

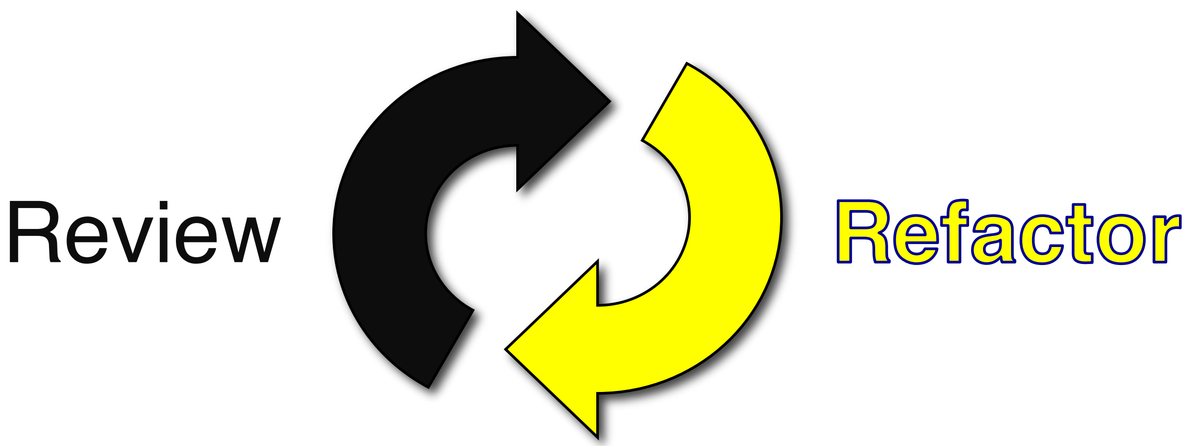
RePhrase

Refactoring

- Refactoring **changes** the **structure** of a program's **source code**

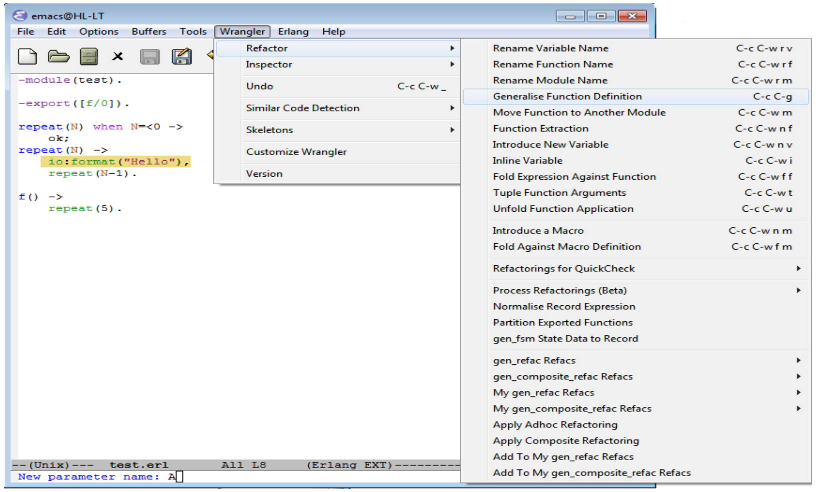
  …… using **well-defined** rules

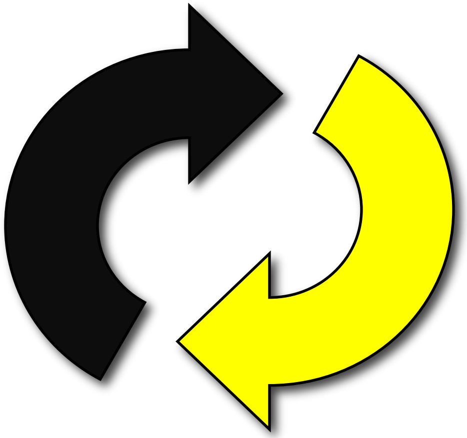
  … *semi-automatically under programmer guidance*



Refactoring can help parallel thinking!

- Helps choose the right abstractions
- Programmer ‘in the loop’
- can be used to introduce (and remove!) parallelism
- Introduce skeletons and then tune for different instances/architectures
- **A parallel playground!**





Program Shaping

- Refactorings that re-structure code to make it ready for parallelisation
  - Changing data types (list to binary)
  - Removing dependencies
  - Simplifying structure
  - Extracting functionality into functions
  - Removing/adding arguments to functions
  - Etc.

Lapedo: a Framework for Hybrid Skeletons

University of St Andrews

- Extends the Skel with hybrid (CPU/GPU) versions of some skeletons
- Based on Erlang OpenCL bindings
- Refactorings to introduce the skeletons automatically
- GPU components need to contain all the code for creating buffers on GPUs, transfer data to/from GPUs, schedule computations...
  - *Lapedo* can generate this code automatically (based on OpenCL)
  - ...but the programmer still needs to provide OpenCL kernels of the desired operations

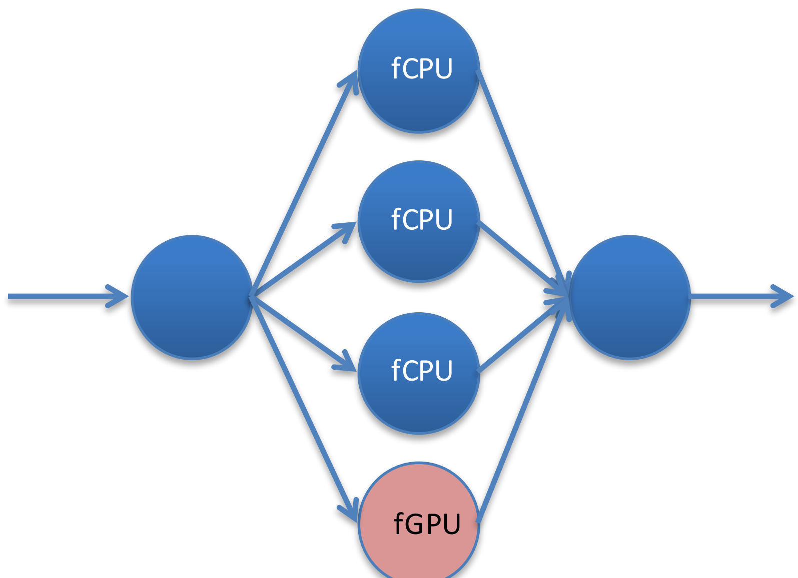
`https://github.com/ParaPhrase/skel`

RePhrase

Hybrid Farm

University of St Andrews

- Applies the same function to a stream of input tasks
- Uses multiple implementations of the function, for different devices in the system
- `{hyb_farm, CPUSkel, GPUSkel, 3, 1}`



(Let's) PaRTE!

One stop shop for parallel programming in Erlang

Combines pattern discovery, Refactoring, Skel and **Lapedo**

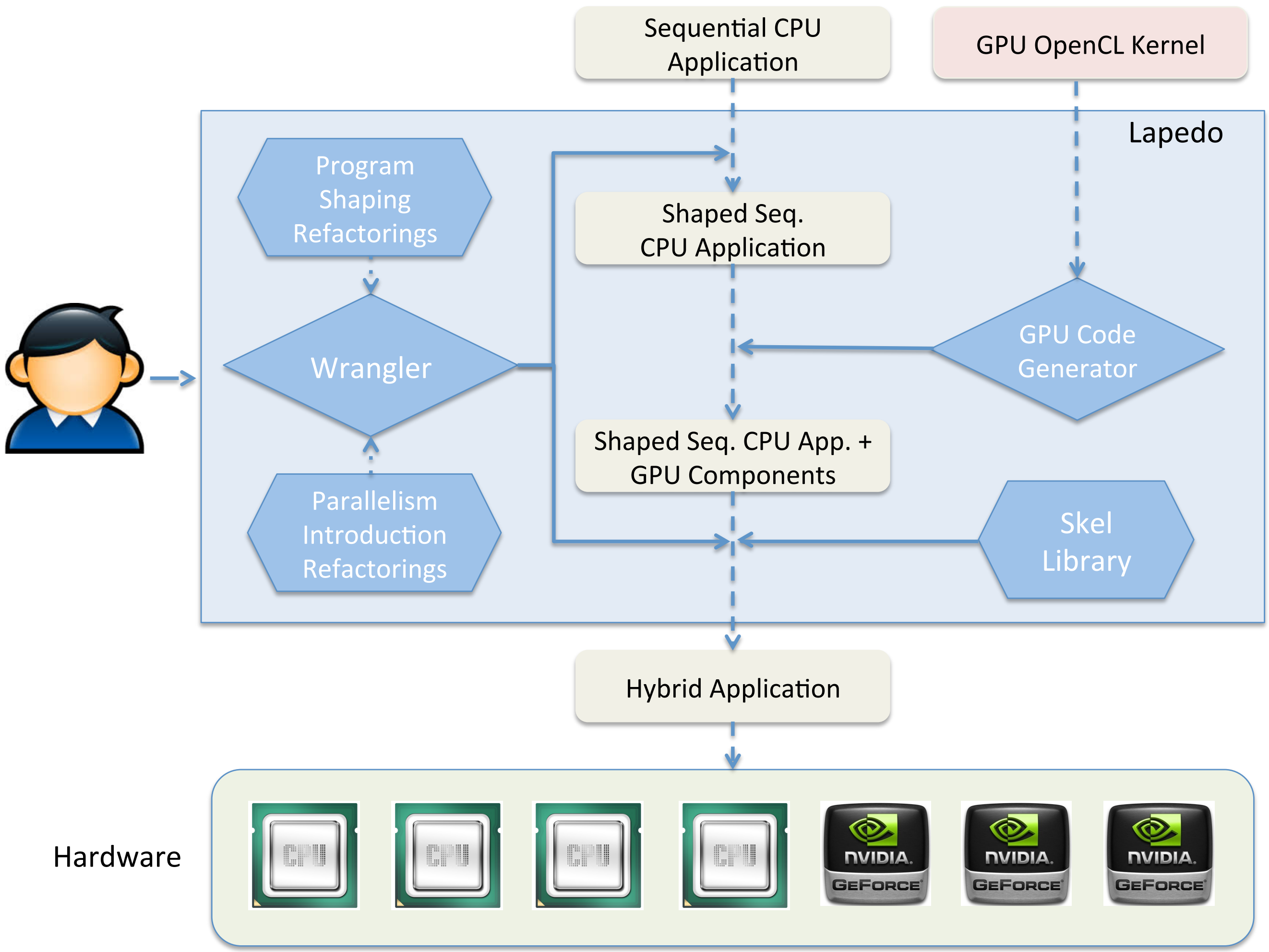
Program Shaping

Pattern Discovery

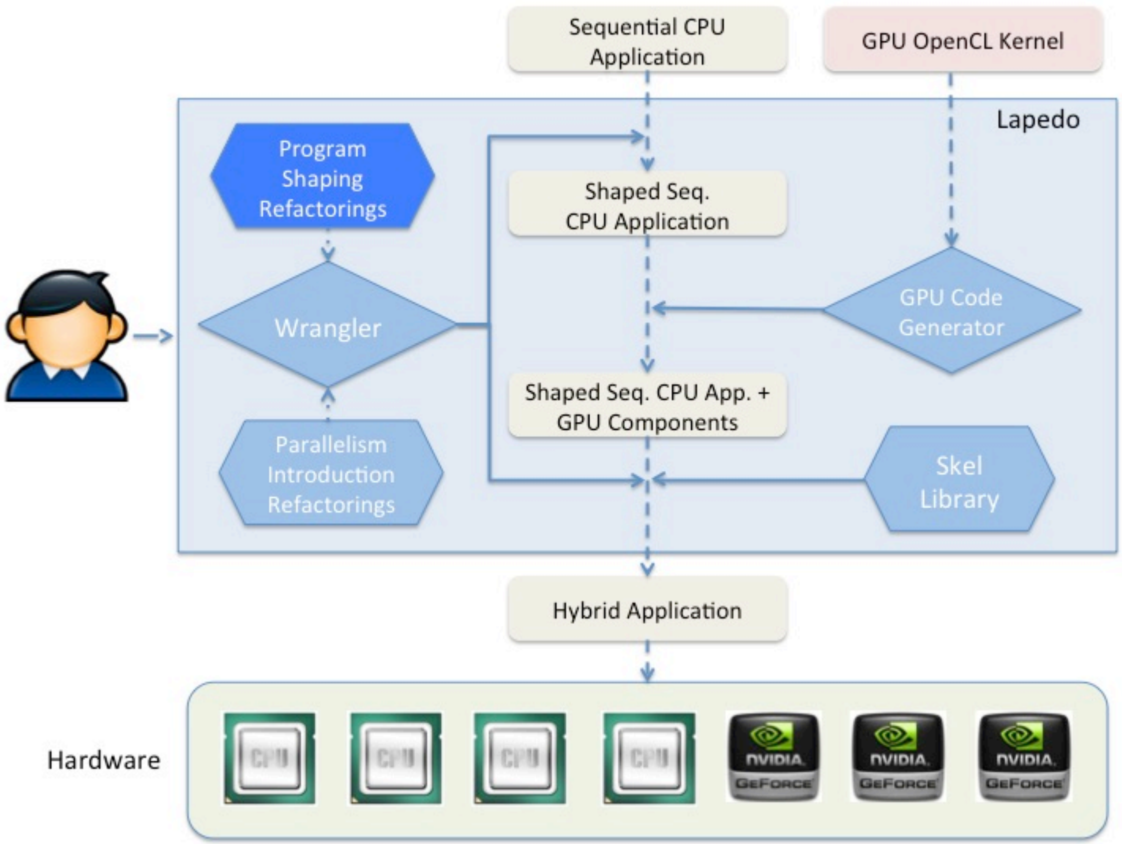
Refactoring

Lapedo

Lapedo

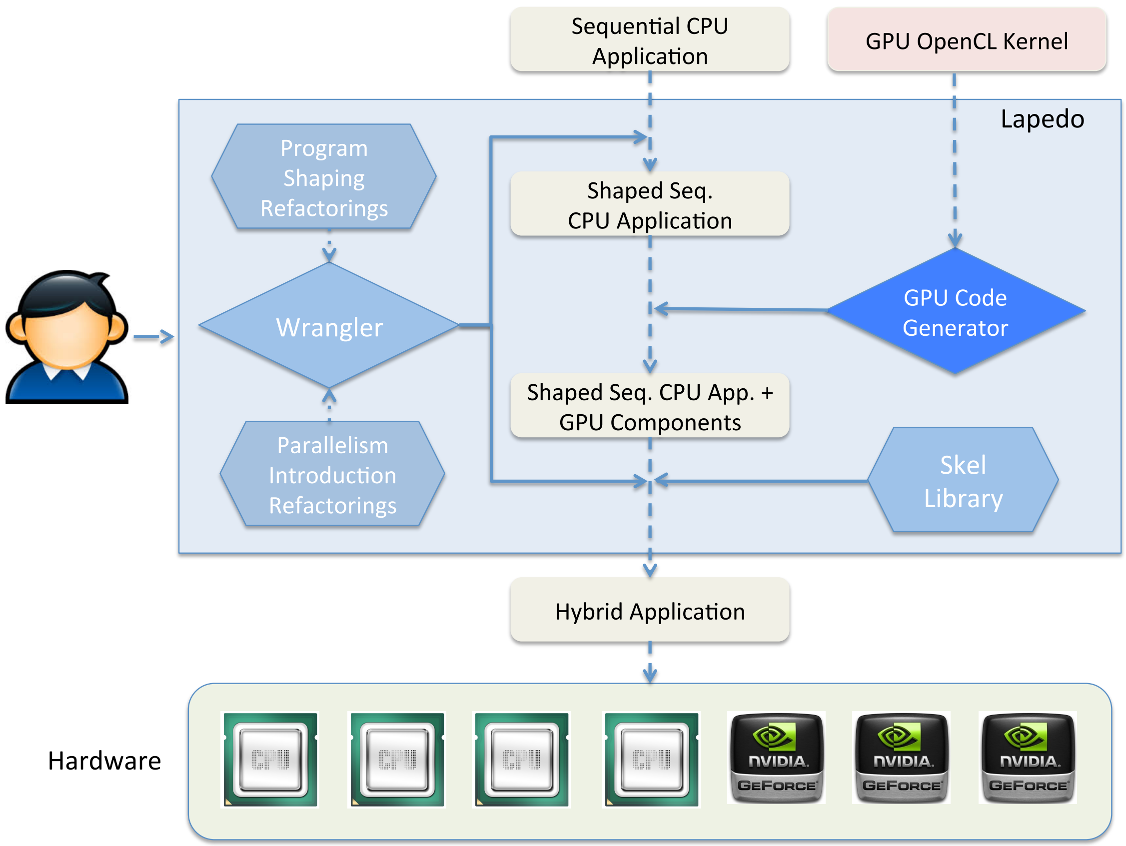

Sequential CPU Application
GPU OpenCL Kernel
Lapedo
Program Shaping Refactorings
Shaped Seq. CPU Application
Wrangler
GPU Code Generator
Shaped Seq. CPU App. + GPU Components
Parallelism Introduction Refactorings
Skel Library
Hybrid Application
Hardware
CPU
CPU
CPU
CPU
NVIDIA GEFORCE
NVIDIA GEFORCE
NVIDIA GEFORCE

Program Shaping



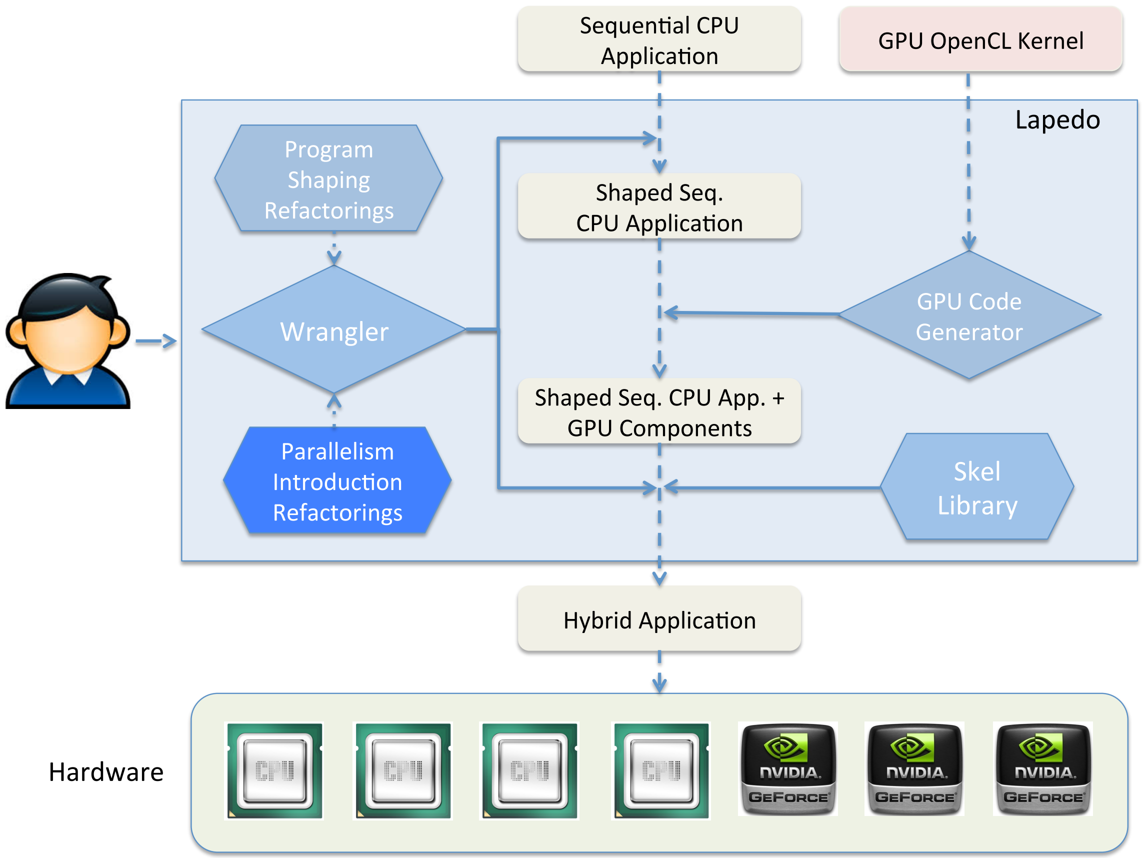
- Refactorings that shape the code so that parallelism can be introduced
- The resulting code is still sequential (or concurrent)
- E.g. converting list into binaries so that they can be copied to accelerator memory

GPU Code Generation



- An Erlang *wrapper* and *Kernel* are generated
- Uses OpenCL bindings
- Creates necessary buffers and sends the data to the GPUs
- Offloads the computations to the GPUs and fetches the results

Skeleton Introduction



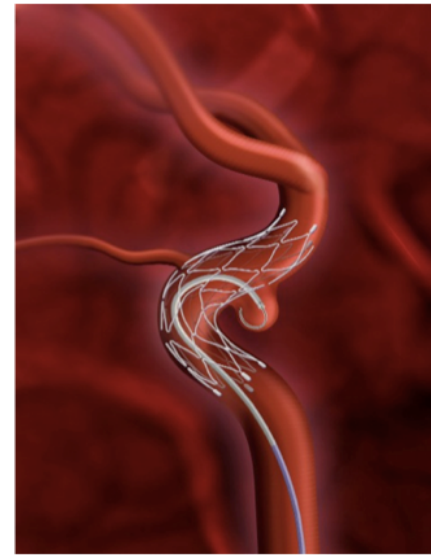
- Programmer uses Lapedo to introduce hybrid skeletons into the shaped application
- This results in a hybrid application that uses the accelerator components from the previous step
- *Semi-automatic*

An Example: NBody

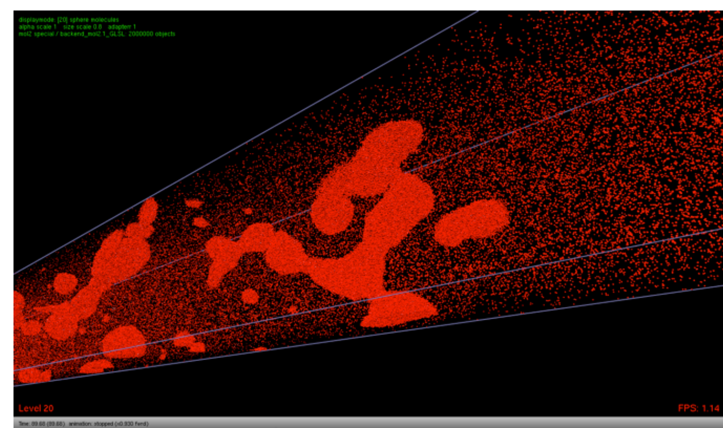
University of St Andrews

- Simulation of a dynamic system of particles under the influence of external forces

- We start with a sequential Erlang application and OpenCL kernel for computing one iteration on a set of particles



```
nbody(Parts,Dt,0) -> Parts;
nbody(Parts, Dt, NrIters) ->
     NewParts = lists:map(fun(X) ->
          nbody_one (X,Parts,Dt) end, Parts),
          nbody(NewParts, Dt, NrIters-1).
```



An Example: Nbody (2)

```
nbody_one (Part, Particles,Dt) ->
    {Ax,Ay,Az} = calc_acc_vector_list (Part, Particles),
    {X,Y,Z,M,Vx,Vy,Vz,_} = Part,
    Xnew = X + Dt*Vx + 0.5*Dt*Dt*Ax,
    Ynew = Y + Dt*Vy + 0.5*Dt*Dt*Ay,
    Znew = Z + Dt*Vz + 0.5*Dt*Dt*Az,
    Vxnew = Vx + Dt*Ax,
    Vynew = Vy + Dt*Ay,
    Vznew = Vz + Dt*Az,
    {Xnew,Ynew,Znew,M,Vxnew,Vynew,Vznew,0}.

calc_acc_vector_list (Part,Particles) ->
    lists:foldl(fun(X,Sum) -> add_to_acc_list(X,Sum,Part)
end,
                {0,0,0}, Particles).
```

Nbody – Program Shaping

University of St Andrews

- Eventually, we want to introduce a hybrid map pattern
- For this, we need to convert operations over lists into operations over binary data, as GPUs only accept binaries
- This operation uses refactoring to automatically change the type.

```
nbody_chunk_cpu(Chunk, Particles,Dt) ->
    binary8Map(fun nbody_one/3, Chunk).

calc_acc_vector_list (Part,Particles) ->
    binary8Foldl(fun(X,Sum) ->
add_to_acc_list(X,Sum,Part)
                              end,
                 {0,0,0}, Particles).
```

RePhrase

Nbody – GPU Code Generation

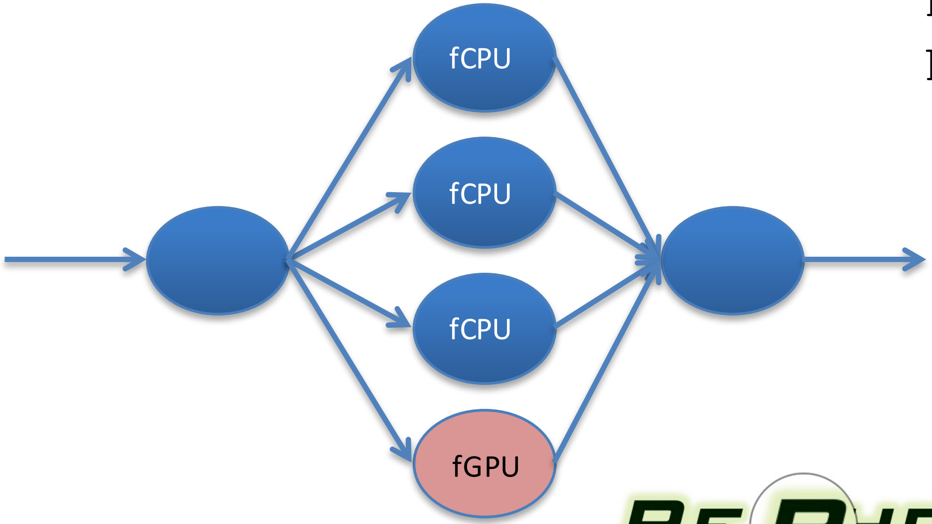
```
nbody_chunk_gpu(Chunk, Particles, Dt) ->
    %% set up the environment
    E = clu:setup(all),
    {ok,Program} = clu:build_source(E, program(ok)),
    {ok,Kernel} = cl:create_kernel(Program, "nbodyKernel"),
    ...
    %% create buffers
    {ok,ChunkInBuffer} = cl:create_buffer(E#cl.context,[read_only],
                                          byte_size(Chunk)),
    ...
    %% data transfers
    {ok,E1} = cl:enqueue_write_buffer(K#kwork.queue, ChunkInBuffer, 0,
                                      byte_size(Chunk), Chunk, []),
    ...
    %% Set kernel arguments
    ok = cl:set_kernel_arg(Kernel, 0, ChunkInBuffer),
    ...
    %% enqueue the kernel
    Global = K#kwork.globalSize,
    Local = K#kwork.localSize,
    {ok,E6} = cl:enqueue_nd_range_kernel(K#kwork.queue, Kernel, [Global], [Local],
    %% read back from the GPU
    {ok,E7} = cl:enqueue_read_buffer(K#kwork.queue, ChunkOutBuffer, 0,
byte_size(Chunk),[E6]),
    Result = case cl:wait(E7,3000) of {ok, Data} -> Data; Res3 -> Res3 end,
    …
```

NBody – Introducing Hybrid Patterns

Finally, using Lapedo, we **refactor** to introduce the hybrid skeletons, using the CPU and GPU components that were created in the previous two steps

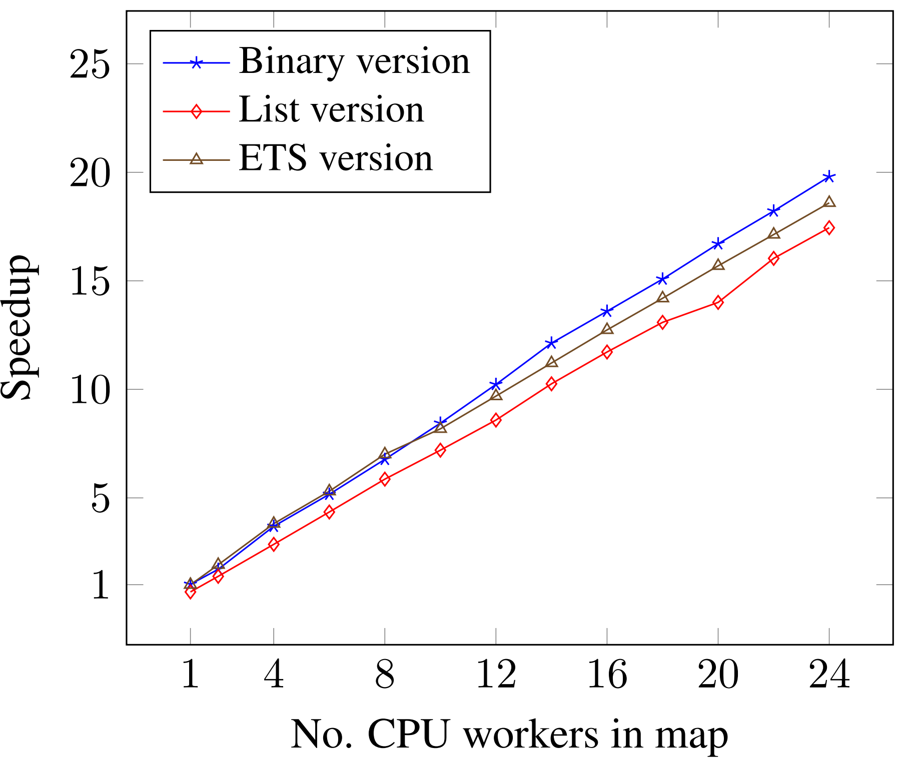
```
nbody_par(Particles, Dt, NCPUWorkers, NGPUWorkers) ->
    Map = {hyb_farm, {func, fun nbody1/1}
                     {func, fun do_nbody_hybrid/1}],

skel:do([Map],create_task_list(Particles, Particles, Dt,
                               NCPUWorkers,
                               NGPUWorkers)).
```



NBody – Hybrid Version

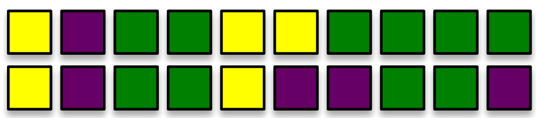
Speedups for first parallel version of N-Body



Speedup of over 200 when adding a GPU!

Distributing Work to CPU/GPUs

Profile



Profile to derive a **ratio** of work between CPUs and GPUs

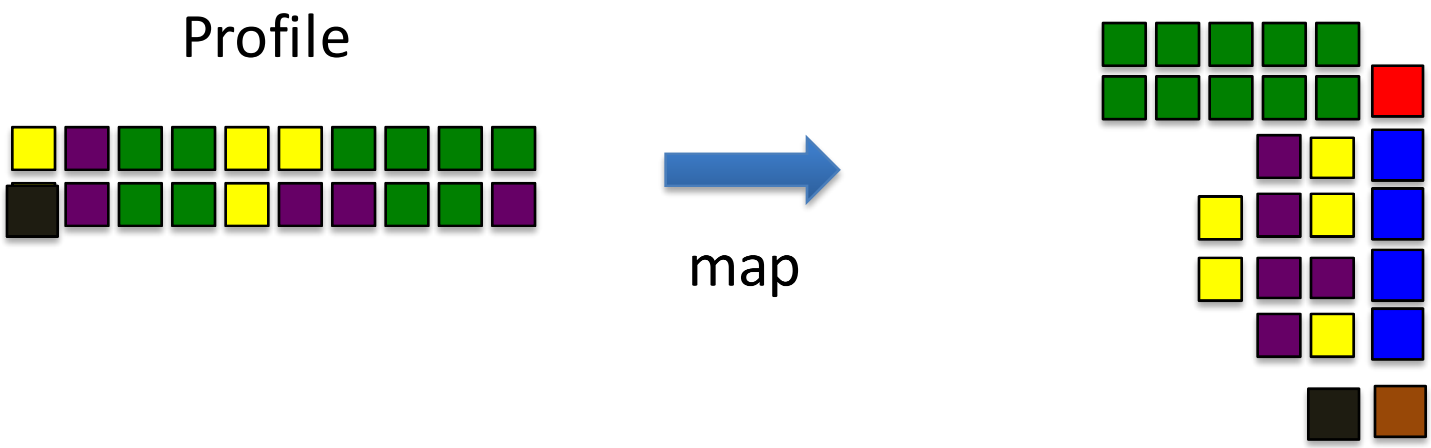
Components

CPU

GPU

CPU/ GPU

Distributing Work to CPU/GPUs



Components

- CPU
- GPU
- CPU/ GPU
- Parallela …

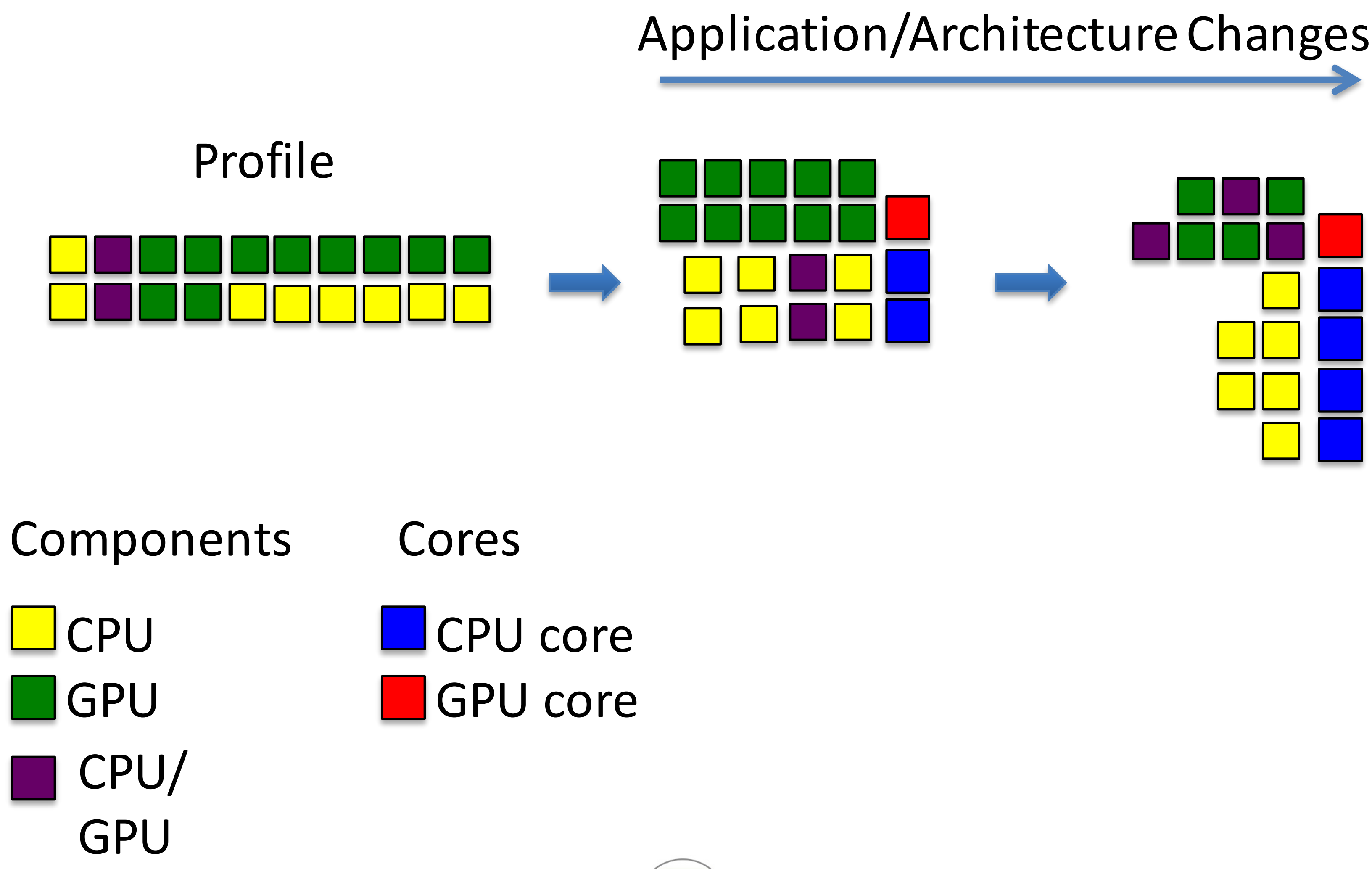
Cores

- CPU core
- GPU core
- Parallela …

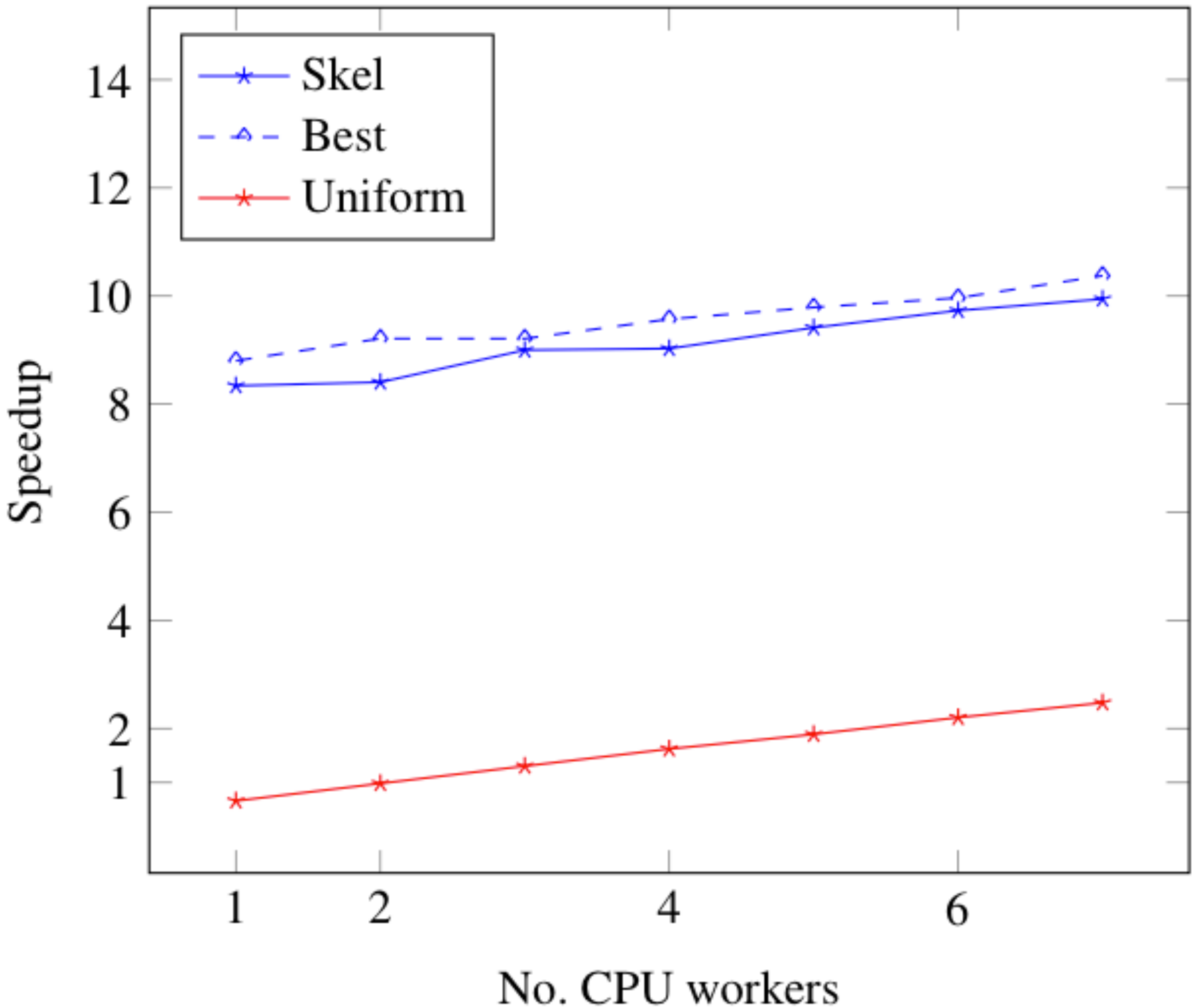
Distributes work **statically**

Calculates optimal scheduling

Distributing Work to CPU/GPUs



Work Division Evaluation



Ant Colony – Speedups

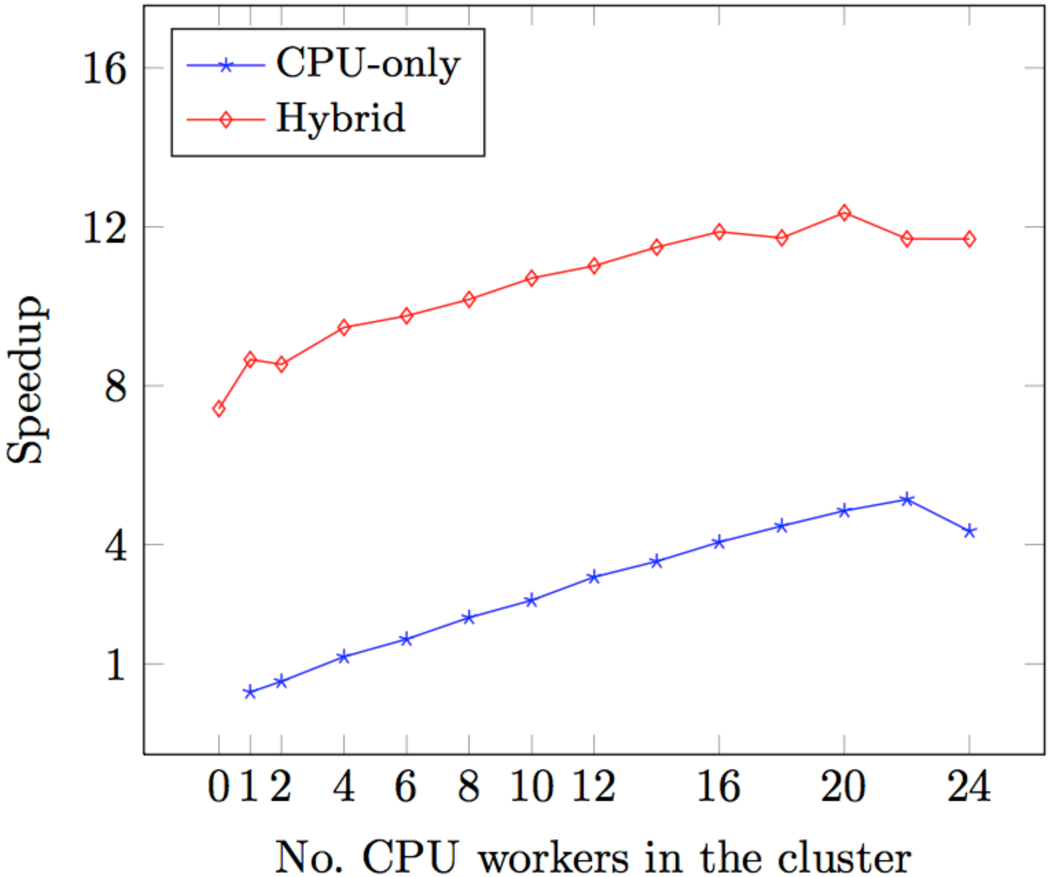
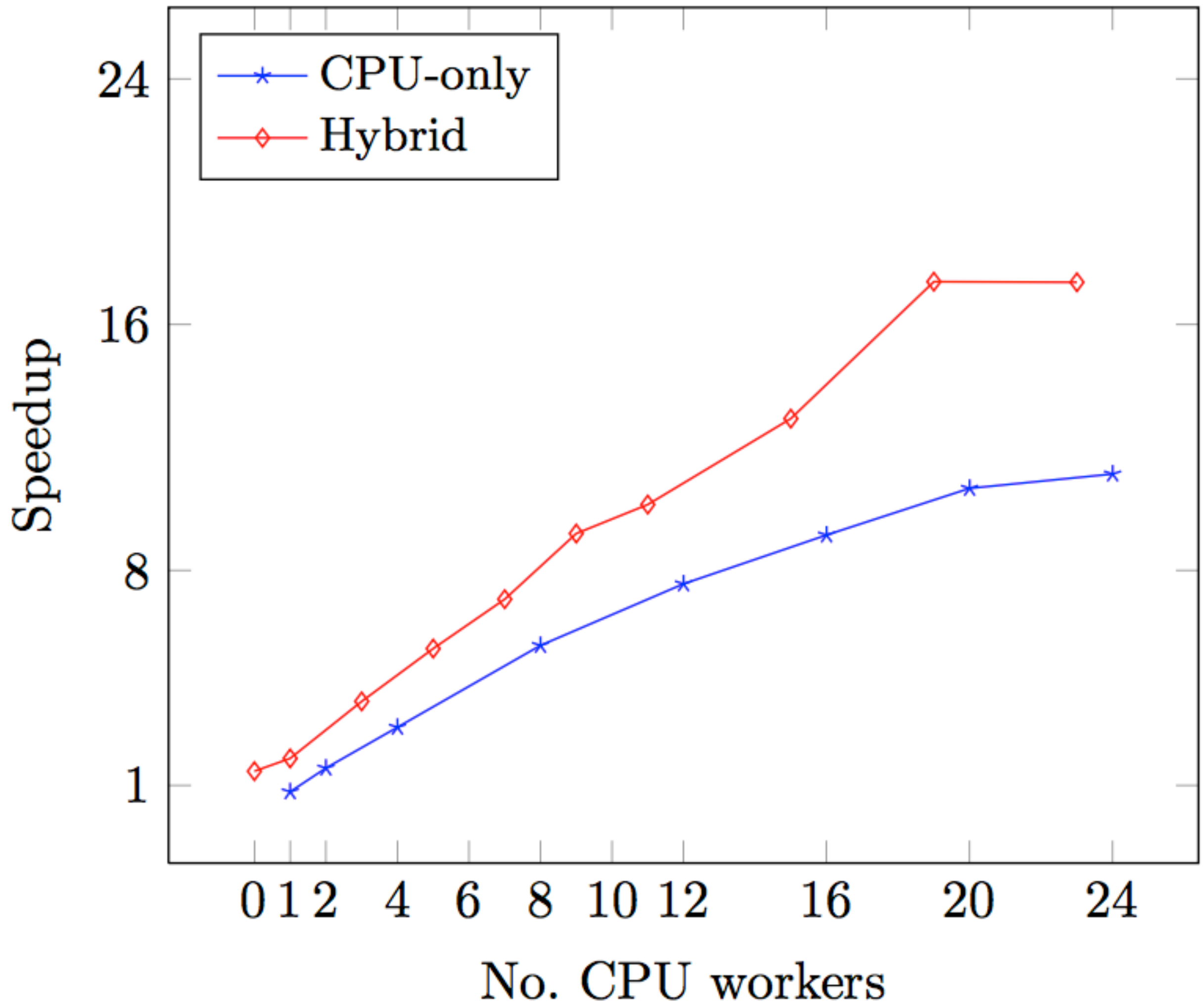


Image Merge



CPU-only
Hybrid
Speedup
24
16
8
1
0
1
2
4
6
8
10
12
16
20
24
No. CPU workers

The ParaFormance™ Tool

- Automatically transforms programs
  - Under programmer control
- Integrated into Eclipse (CDT)
  - Widely used commercial development system
- Supports full C++(14) standard
- Simple and easy to use

Conclusions

University of St Andrews

- Lapedo can help programmers write applications for heterogeneous parallel systems
- Semi-automatic refactorings to introduce CPU and accelerator components into sequential or concurrent code
- Automatic generation of the code to offload computations to accelerators and for data transfers
- Excellent speedups for several use cases with minimal programmer effort
- More details will be provided in

V.Janjic, C. Brown and K. Hammond, **Hybrid-Skel: A Library of Hybrid Parallel Skeletons for Programming Heterogeneous Multi-Core/Many-Core Systems in Erlang**, PARCO 2015

REPHRASE

Lapedo/Rephrase Needs You!

University of St Andrews

- 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/rephrase-news

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

University of St Andrews

THANK YOU!

`http://rephrase-ict.eu`

@rephrase_eu