StarPU: a runtime system for multiGPU multicore machines

Raymond Namyst
RUNTIME group, INRIA Bordeaux

Journées du Groupe Calcul
Lyon, November 2010

The RUNTIME Team

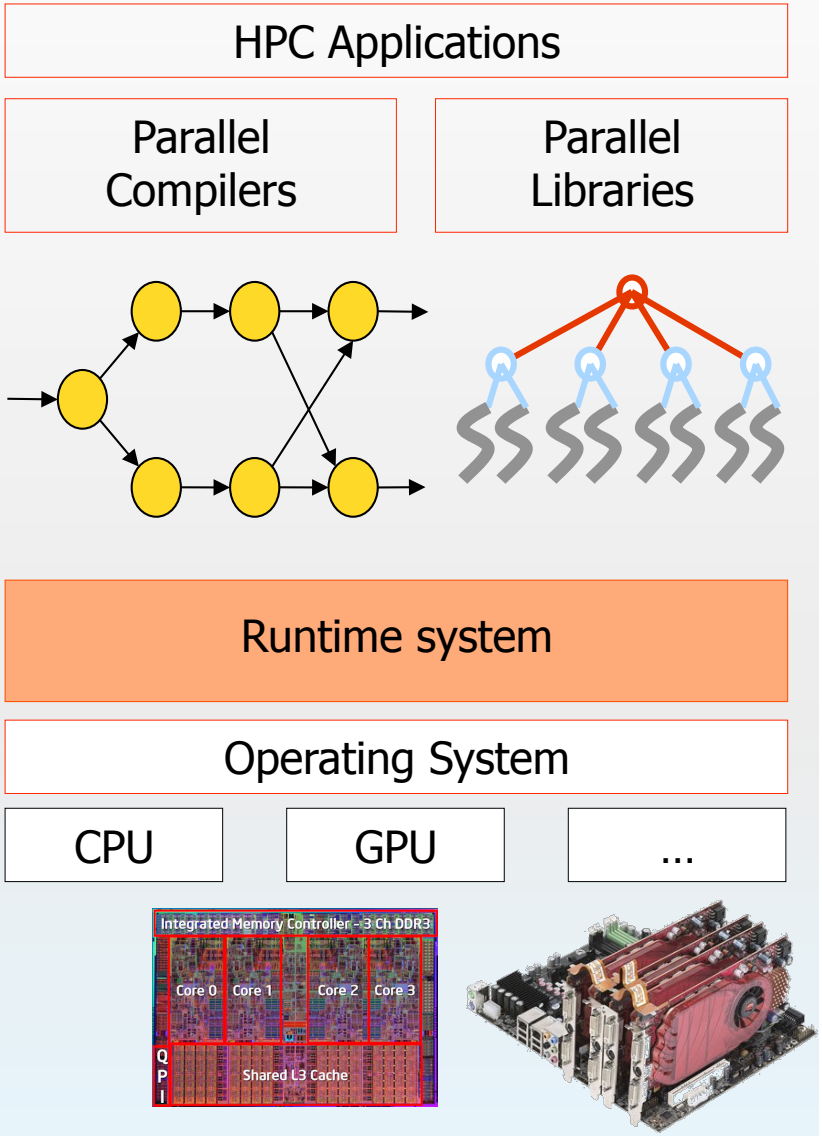
High Performance Runtime Systems for Parallel Architectures

- Mid-size research group
  - 9 permanent researchers
  - 5 engineers
  - ~10 PhD students
- Part of
  - INRIA Bordeaux – Sud-Ouest Research Center
  - LaBRI, Computer Science Lab at University of Bordeaux 1

Overview of research activities

Toward "portability of performance"

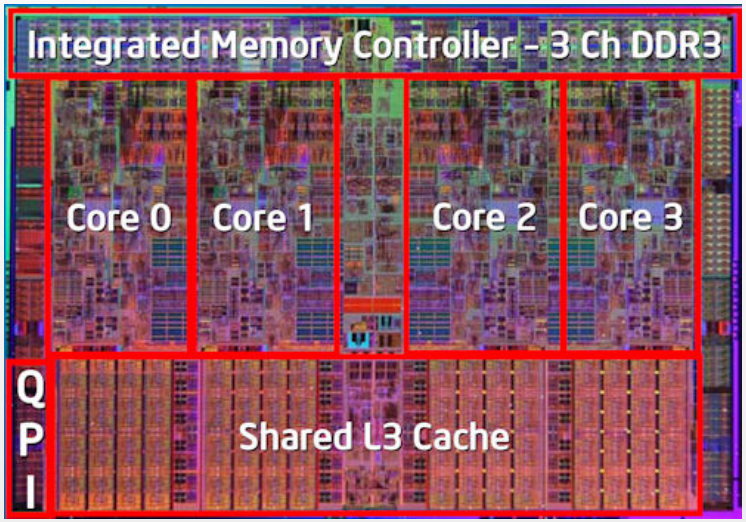
- Do dynamically what can't be done statically
  - Understand evolution of architectures
  - Enable new programming models
  - Put intelligence into the runtime!
- Exploiting shared memory machines
  - Thread scheduling over hierarchical multicore architectures
    - OpenMP
  - Task scheduling over accelerator-based machines
- Communication over high speed networks
  - Multicore-aware communication engines
  - Multithreaded MPI implementations
- Integration of multithreading and communication
  - Runtime support for hybrid programming
    - MPI + OpenMP + CUDA + TBB + …



Evolution of multiprocessor architecture

Multicore is a solid trend

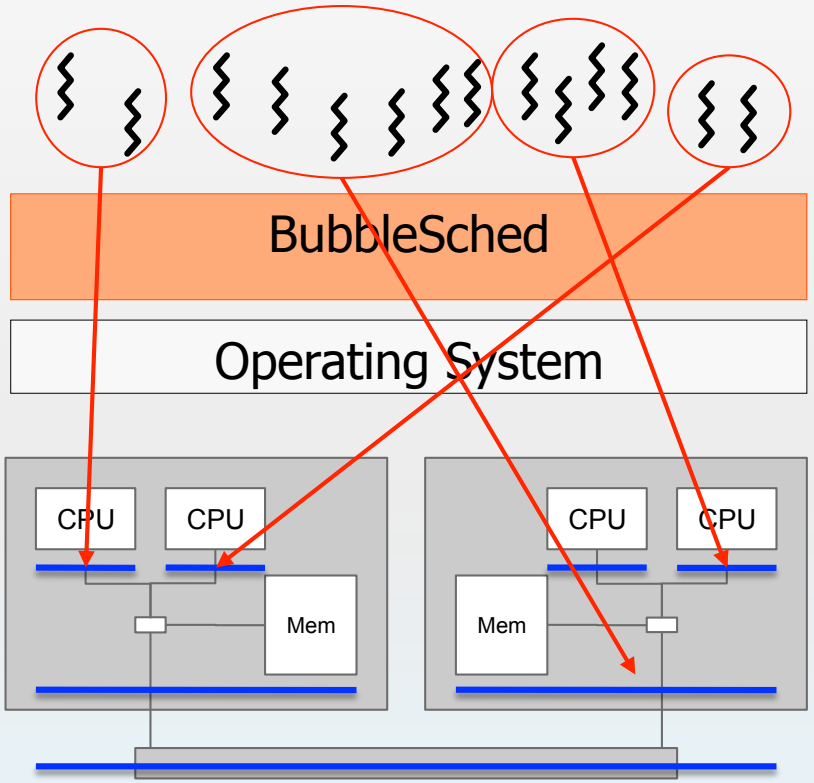
- Multicore chips
  - Architects' answer to the question: "What circuits should we add on a die?"
    - No point in adding new predicators or other intelligent units…
  - Back to complex memory hierarchies
    - Shared caches
    - NUMA factors
  - Clusters can no longer be considered as "flat sets of processors"



Thread Scheduling over Multicore Machines

Scheduling structured sets of threads

- The Bubble Scheduling concept
  - Capturing application's structure with nested bubbles
  - Scheduling = dynamic mapping trees of threads onto a tree of cores
  - Designing portable NUMA-aware scheduling policies
    - Focus on algorithmic issues

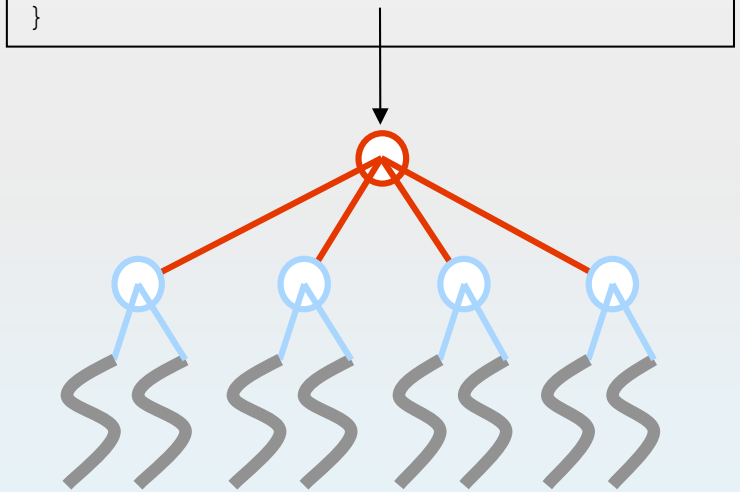


Thread Scheduling over Multicore Machines

The ForestGOMP OpenMP environment

- Extension to GNU OpenMP
  - Binary compliant with existing applications
- Designing multicore-friendly programs with OpenMP
  - Parallel sections generate bubbles
  - Nested parallelism is welcome!
- Composability
  - Challenge = autotuning the number of threads per parallel region

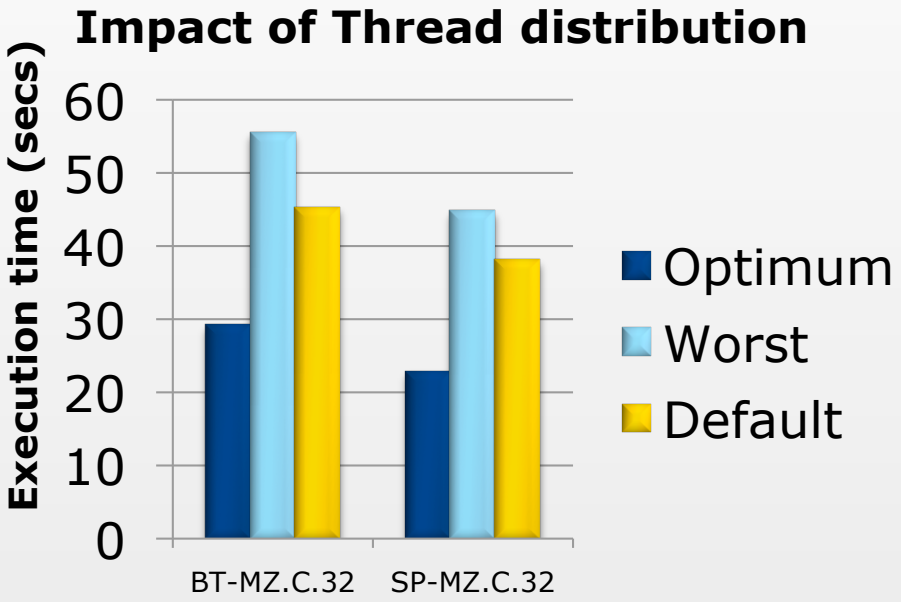
```
void work()
{
  ...

#pragma omp parallel for
  for (int i=0; i<MAX; i++)
    {
      ...
#pragma omp parallel for
num_threads (2)
      for (int k=0; k<MAX; k++)
        ...
    }
}
```

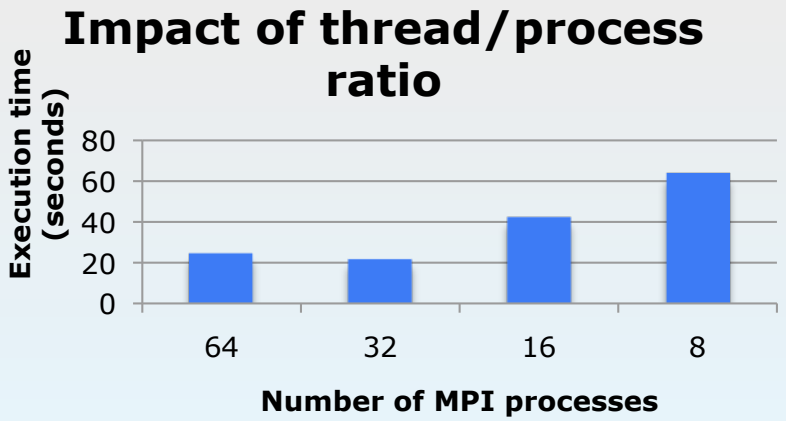


Mixing OpenMP with MPI

It makes sense even on shared-memory machines

- MPI should fit the underlying topology
  - HWLOC library [with OpenMPI group]
- Experimental platform for hybrid applications
  - Topology-aware process allocation
  - Customizable core/process ratio
  - # of OpenMP tasks independent from # of cores
    - OMP_NUM_THREADS ignored

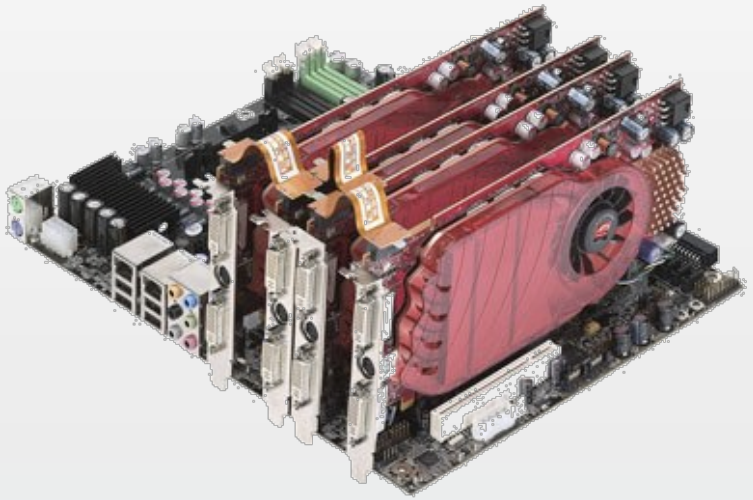




Recent evolution of hardware

Towards multi-GPU clusters

- GPU are the *new kids on the block*
  - Very powerful data-parallel accelerators
  - Specific instruction set
  - No hardware memory consistency
- Other chips already feature specialized harware
  - IBM Cell/BE
    - 1 PPU + 8 SPUs
  - Intel ~~Larrabee~~ MIC
    - 48-core with SIMD units
- Are we happy with that?
  - No, but it's probably unavoidable!

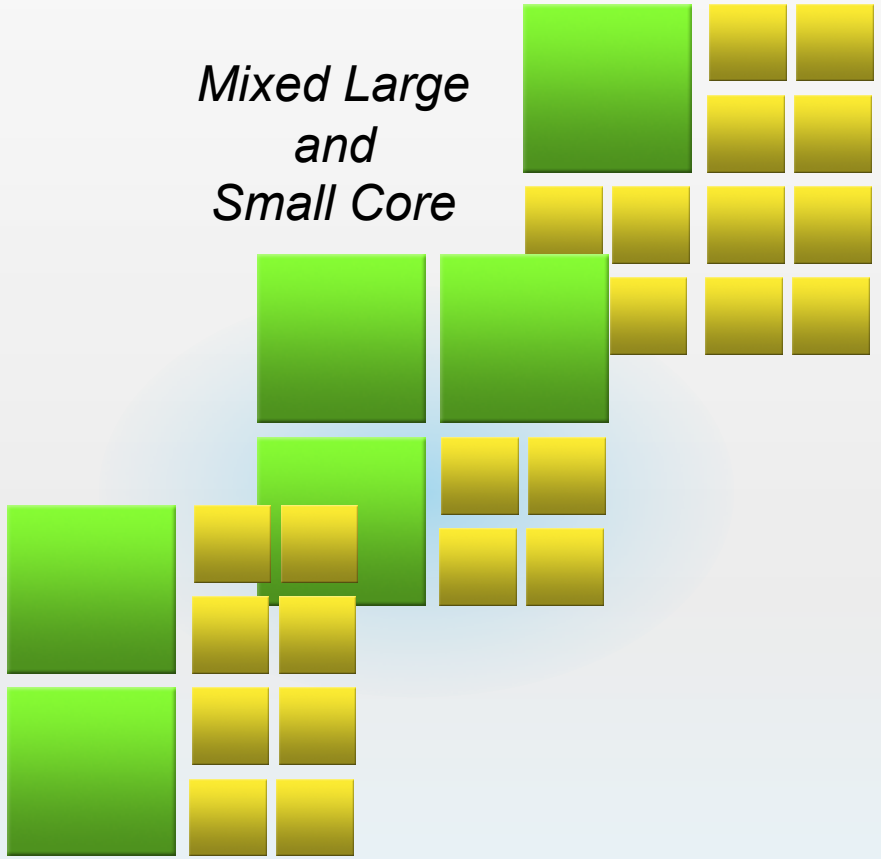


Future evolution of hardware

Heterogeneity is a also solid trend

- One interpretation of "Amdalh's law"
  - We will always need powerful, general purpose cores to speed up sequential parts of our applications!
- "Future processors will be a mix of general purpose and specialized cores"

  [anonymous source]



Programming environments

Programming the hard way

- Software Development Kits and Hardware Specific Languages
  - "Stay close to the hardware and get good performance"
    - Low-level abstractions
  - Compilers generate code for accelerator device
- Examples
  - Nvidia's CUDA
    - *Compute Unified Device Architecture*)
  - ATI Stream
    - Previously *Brook* and *Close-To-Metal*
  - IBM Cell SDK
  - OpenCL

Programming environments

Are we forced to use such low-level tools?

- Higher-level libraries are available
  - Generic libraries
    - Intel CT
  - Well-known computation kernels
    - BLAS routines
      - e.g. CUBLAS
    - FFT kernels
- Implementations are continuously enhanced
  - High Efficiency
- Limitations
  - Data must usually fit accelerators memory
  - Multi-GPU configurations not yet supported

Programming environments
High-Level Languages and Tools

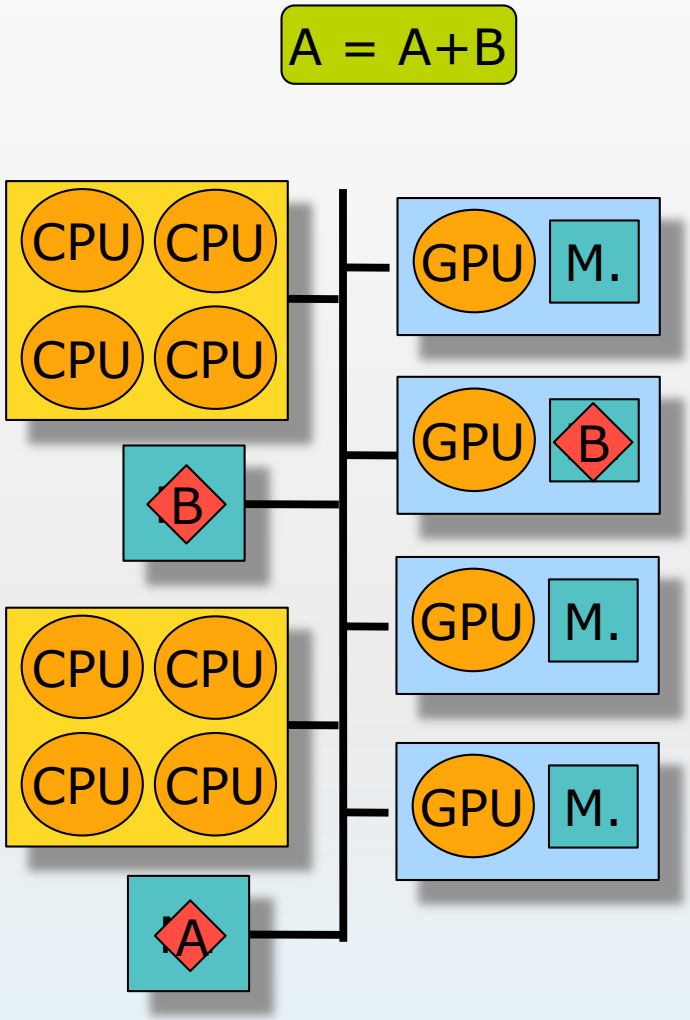
- Directive-based languages for offloading tasks to accelerators
  - Idea: use simpler directives… and better compilers!
    - HMPP (Caps Enterprise)
    - GPU SuperScalar (Barcelona Supercomputing Center)

```
#pragma omp task inout(C[BS][BS])
void matmul( float *A, float *B, float *C) {
// regular implementation
}
#pragma omp target device(cuda) implements(matmul)
copy_in(A[BS][BS] , B[BS][BS] , C[BS][BS])
copy_out(C[BS][BS])
void matmul cuda ( float *A, float *B, float *C) {
// optimized kernel for cuda
}
```

Overview of StarPU

A runtime system for heterogeneous architectures

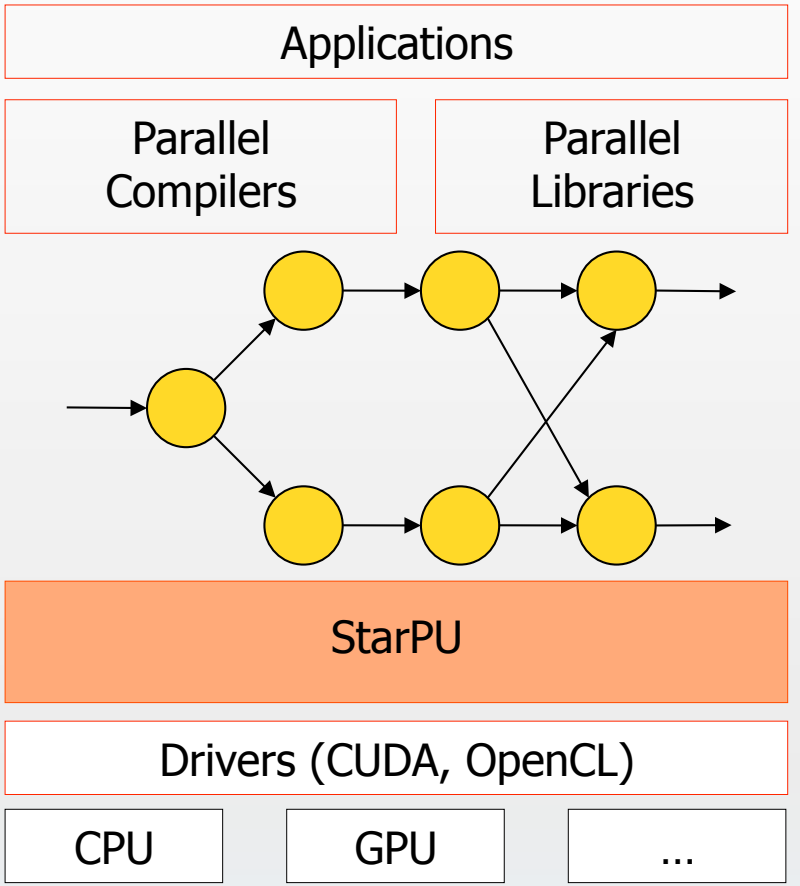
- Rational
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous cores
  - Avoid unnecessary data transfers between accelerators
    - Software VSM for heterogeneous machines



Overview of StarPU

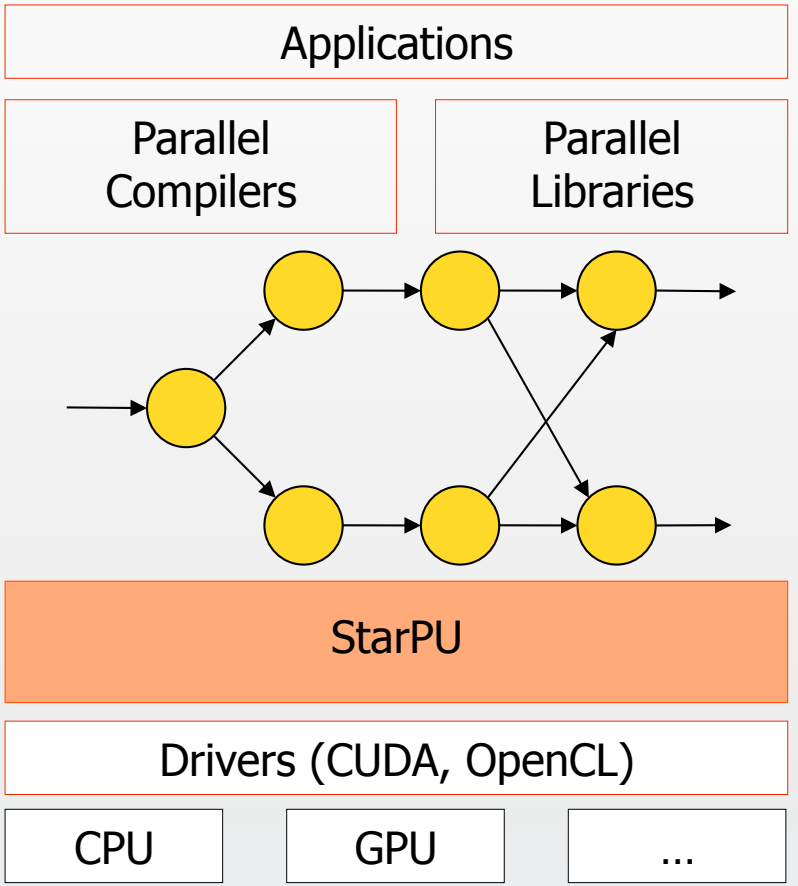
Maximizing PU occupancy, minimizing data transfers

- Ideas
  - Accept tasks that may have multiple implementations
    - Together with potential inter-dependencies
      - Leads to a dynamic acyclic graph of tasks
  - Provide a high-level data management layer
    - Application should only describe
      - which data may be accessed by tasks
      - How data may be divided



Memory Management
Automating data transfers

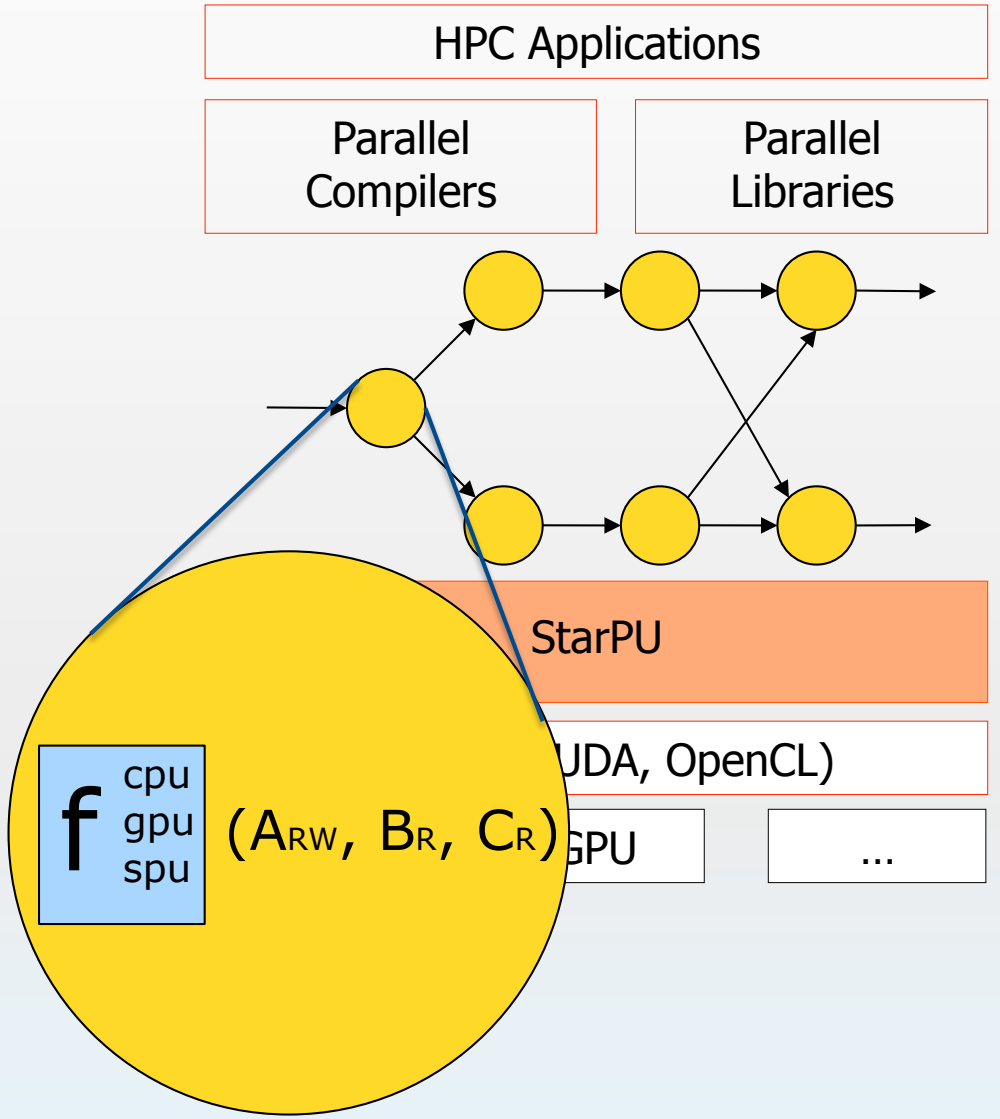
- StarPU provides a Virtual Shared Memory subsystem
  - Weak consistency
    - Explicit data fetch
  - Replication
    - MSI protocol
  - Single writer
    - Except for specific, "accumulation data"
  - High-level API
    - Partitioning filters
- Input & output of tasks = reference to VSM data



Tasks scheduling

Dealing with heterogeneous hardware accelerators

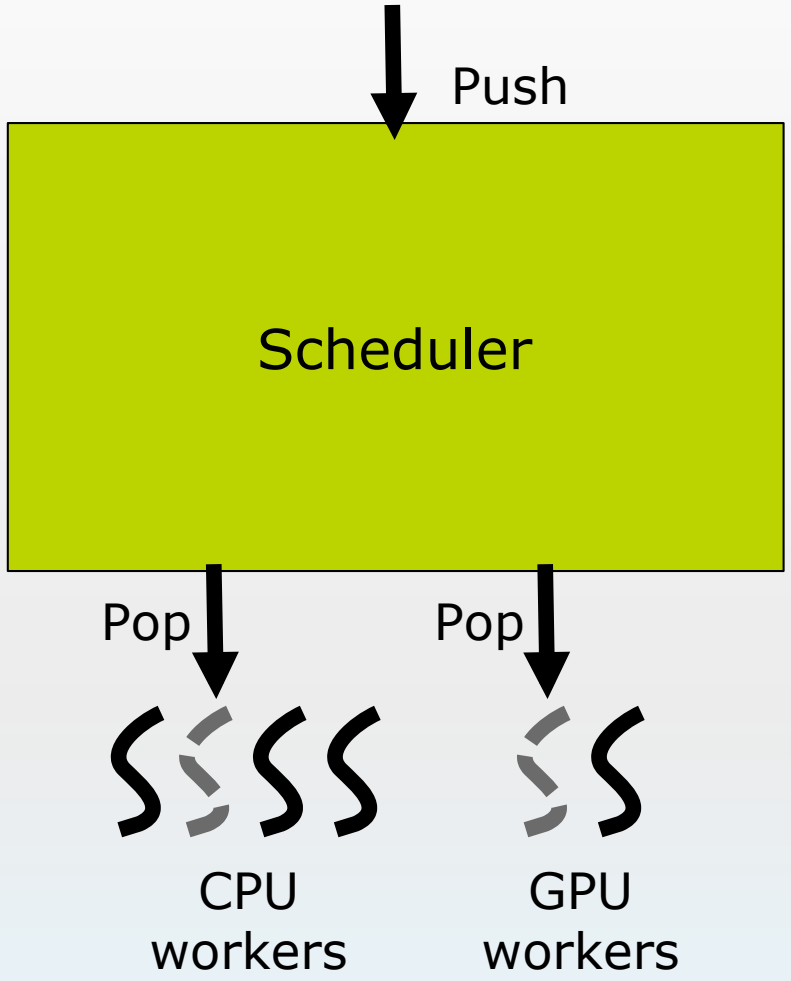
- Tasks =
  - Data input & output
  - Dependencies with other tasks
  - Multiple implementations
    - E.g. CUDA + CPU implementation
  - Scheduling hints
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



Tasks scheduling

How does it work?

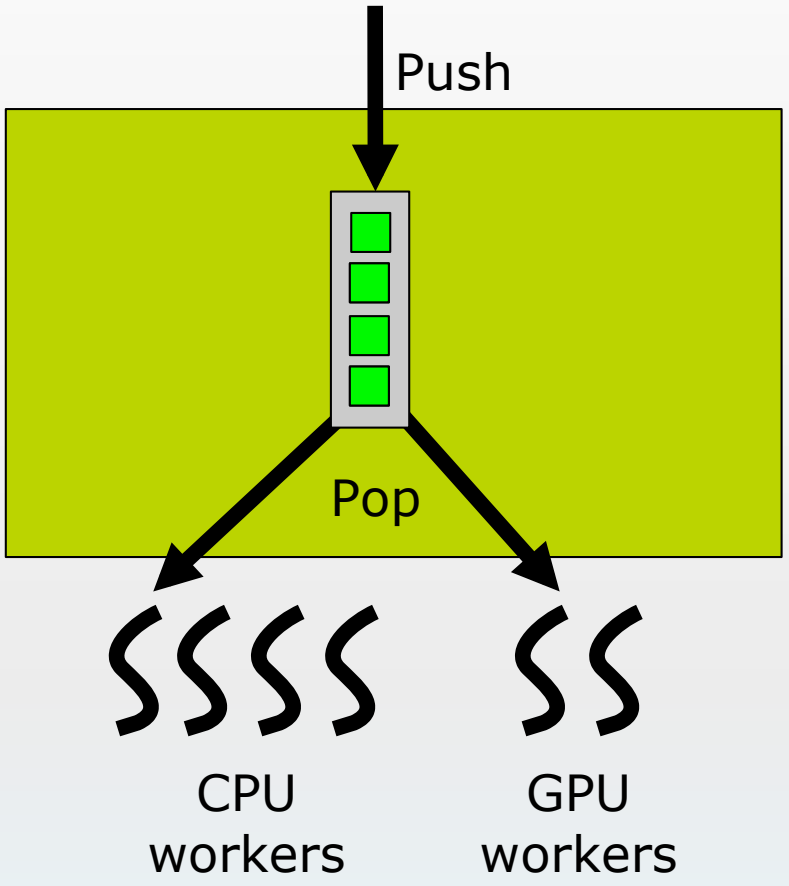
- When a task is submitted, it first goes into a pool of "frozen tasks" until all dependencies are met
- Then, the task is "pushed" to the scheduler
- Idle processing units actively poll for work ("pop")
- What happens inside the scheduler is… up to you!



Tasks scheduling

Developing your own scheduler

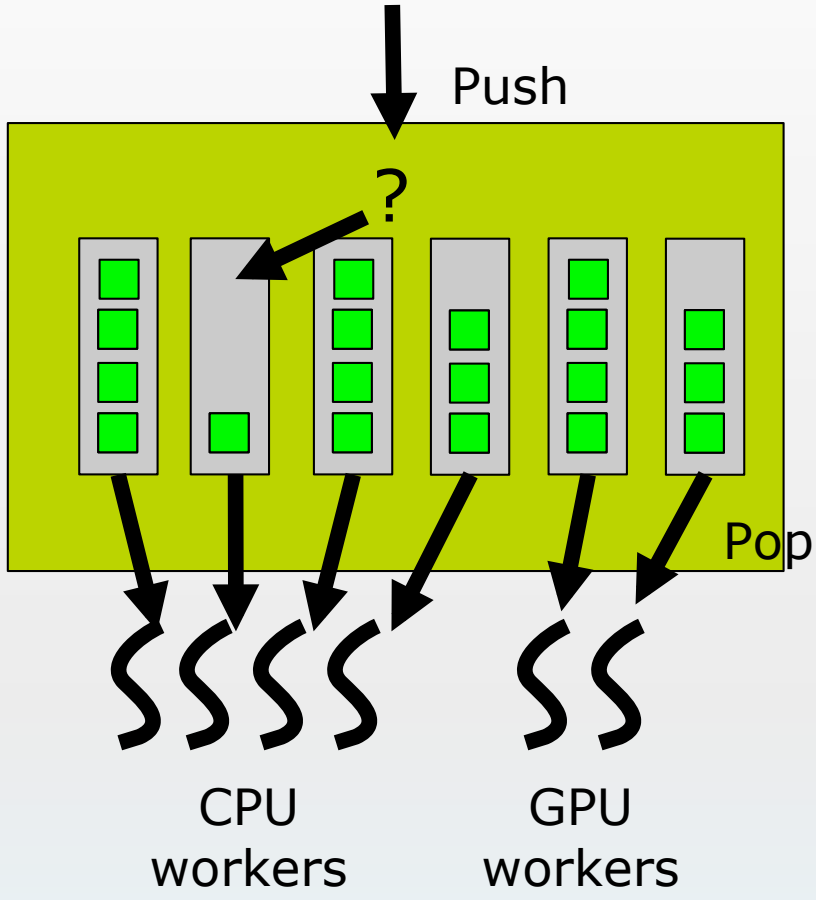
- Queue based scheduler
  - Each worker « pops » task in a specific queue
- Implementing a strategy
  - Easy!
  - Select queue topology
  - Implement « pop » and « push »
    - Priority tasks
    - Work stealing
    - Performance models, …
- Scheduling algorithms testbed



Tasks scheduling

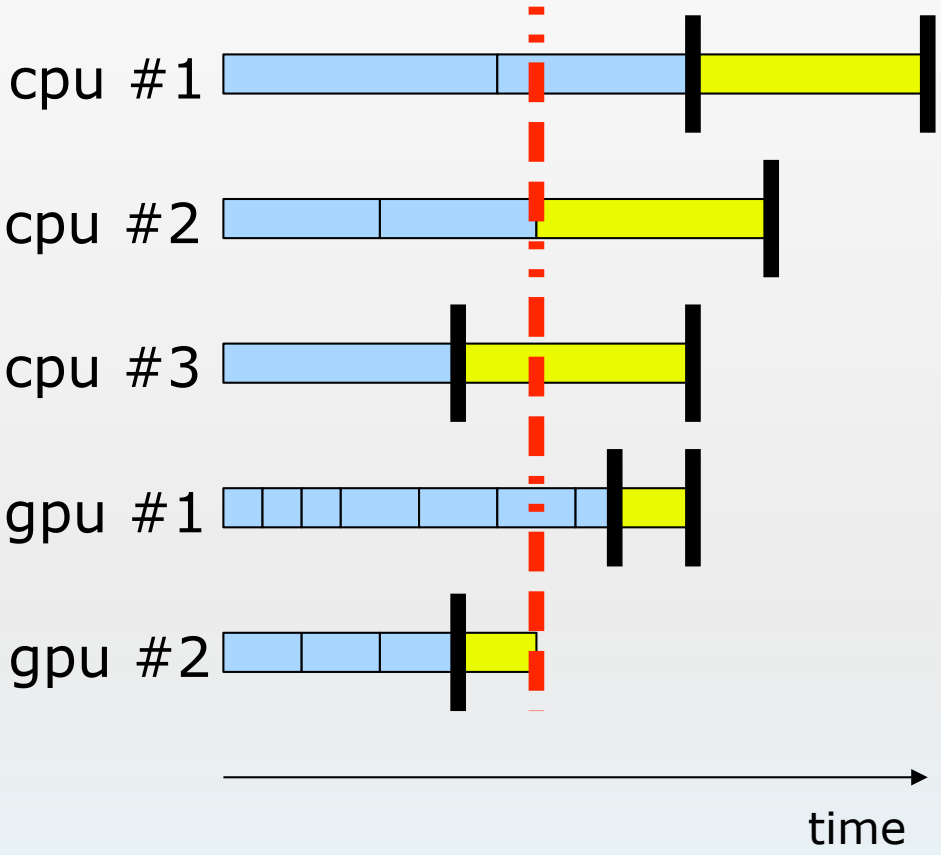
Developing your own scheduler

- Queue based scheduler
  - Each worker « pops » task in a specific queue
- Implementing a strategy
  - Easy!
  - Select queue topology
  - Implement « pop » and « push »
    - Priority tasks
    - Work stealing
    - Performance models, …
- Scheduling algorithms testbed



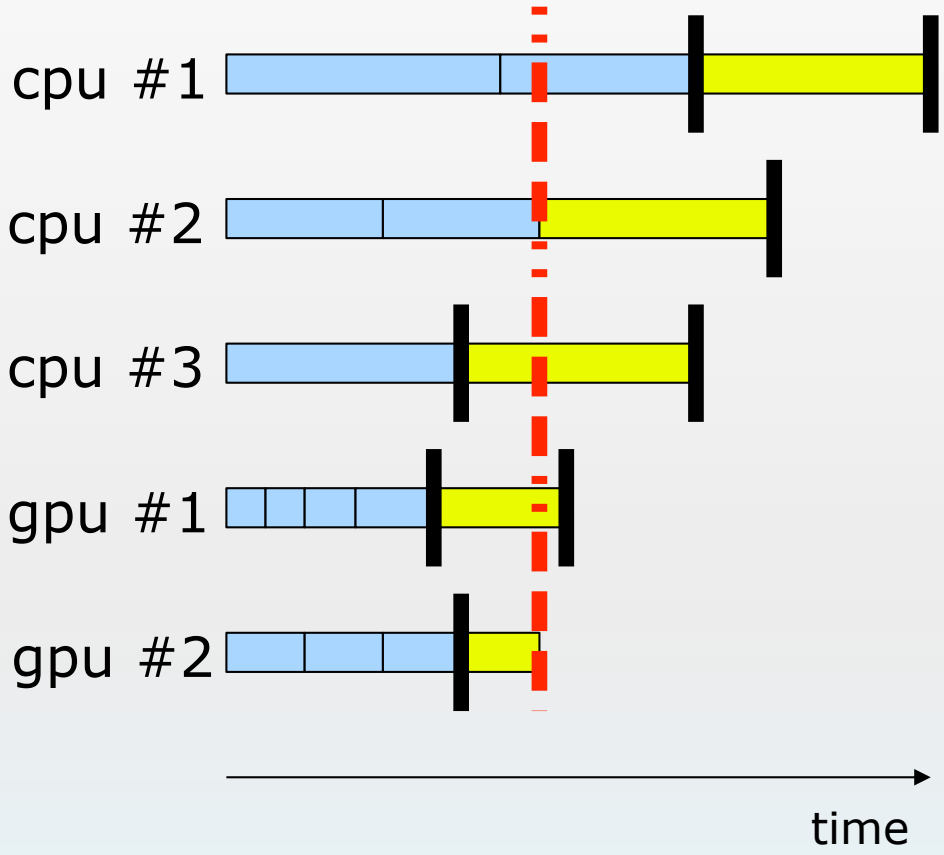
Dealing with heterogeneous architectures
Performance prediction

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
- Can be used to improve scheduling
  - E.g. Heterogeneous Earliest Finish Time



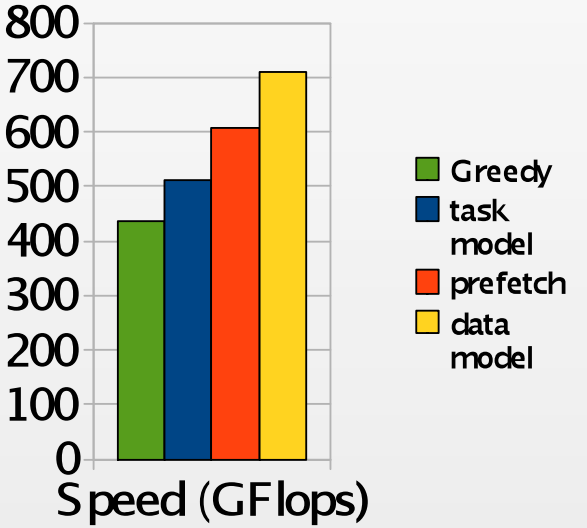
Dealing with heterogeneous architectures
Performance prediction

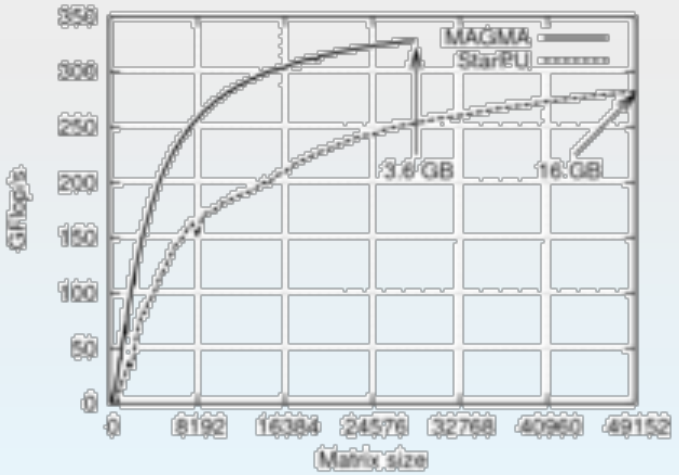
- Data transfer time estimation
  - Sampling based on off-line calibration
- Can be used to
  - Better estimate overall exec time
  - Minimize data movements



Dealing with heterogeneous architectures
Performance

- On the influence of the scheduling policy
  - LU decomposition
    - 8 CPUs (Nehalem) + 3 GPUs (FX5800)
    - 80% of work goes on GPUs, 20% on CPUs
- StarPU exhibits good scalability *wrt*:
  - Problem size
  - Number of GPUs

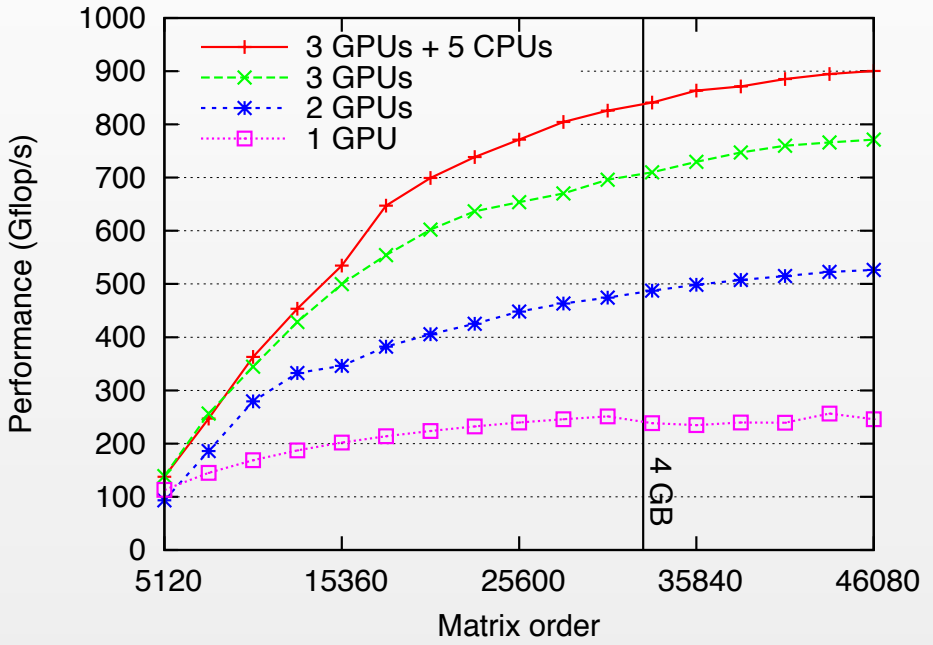


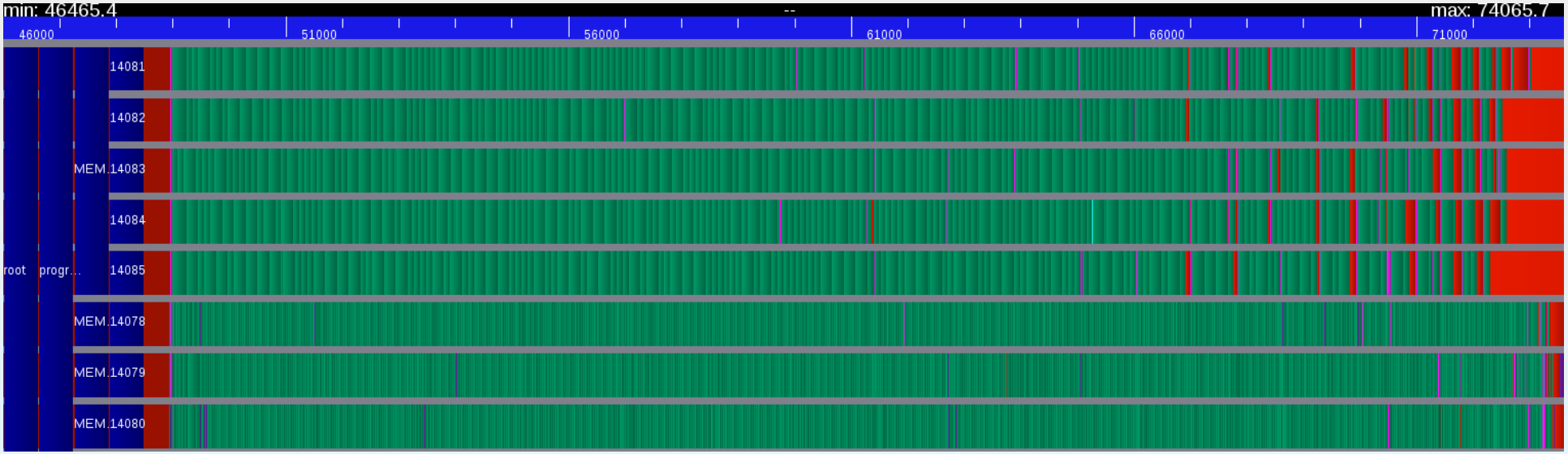


Dealing with heterogeneous architectures

Implementing PLASMA on top of StarPU

- With University of Tennessee & INRIA HiePACS
  - Cholesky decomposition
    - 5 CPUs (Nehalem) + 3 GPUs (FX5800)
    - Efficiency > 100%

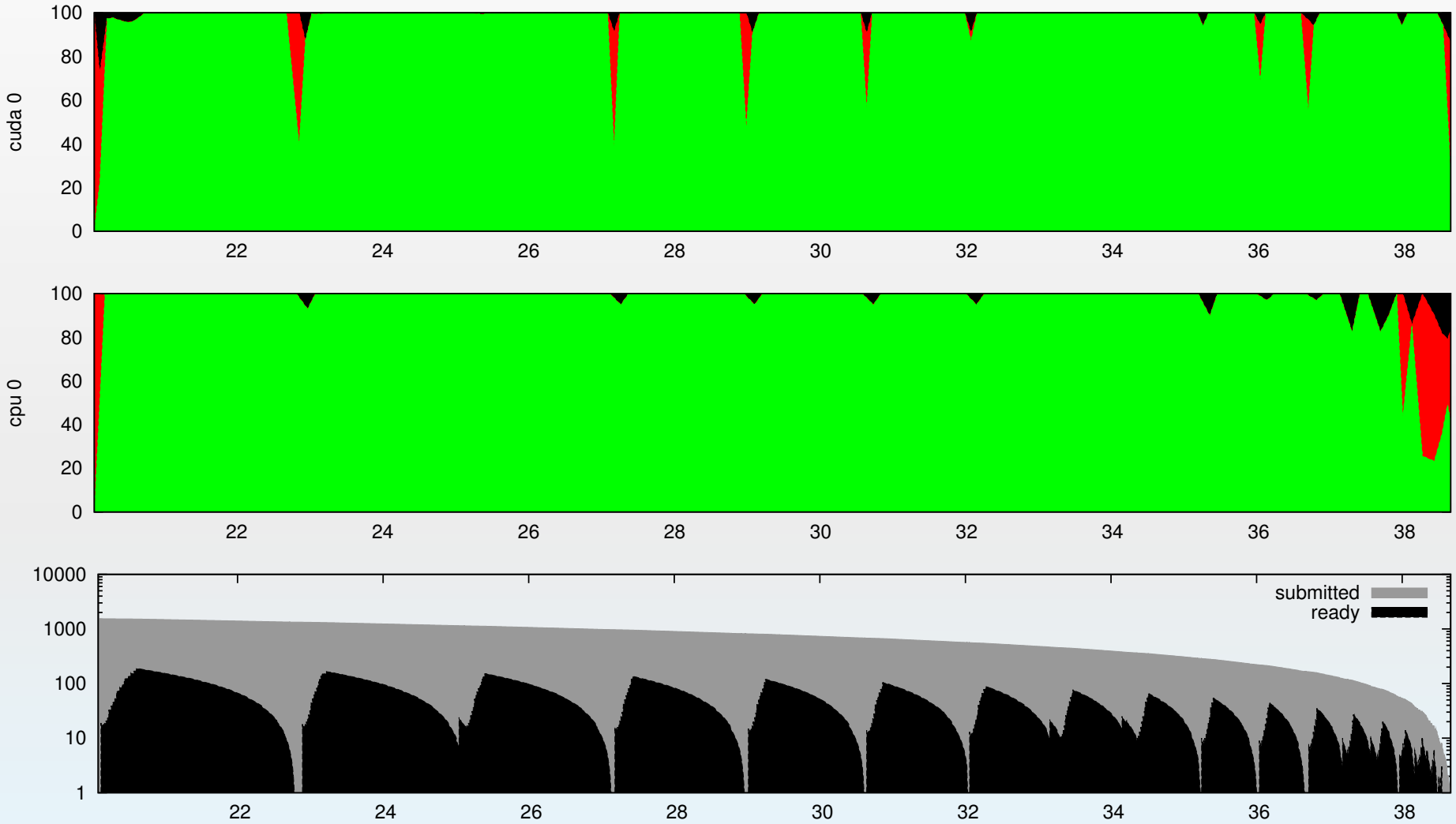




Performance feedback API

Online/offline performance analysis

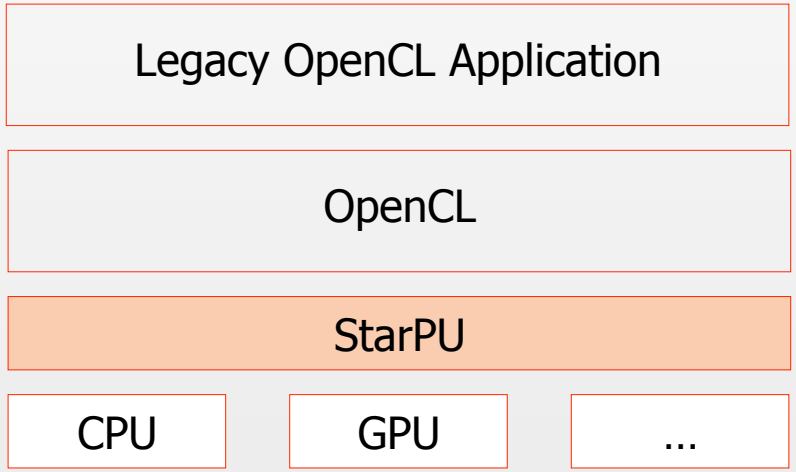
- “starpu_top”



Using StarPU through a standard API

A StarPU driver for OpenCL (Sylvain Henry)

- Run legacy OpenCL codes on top of StarPU
  - OpenCL sees a number of starPU devices
- Performance issues
  - OpenCL kernels are “generic”
    - So they are likely to behave well only on a particular type of architecture



Moving to multi-GPU clusters

Putting it all together

- MPI + StarPU
  - StarPU is able to use GPUs and CPUs simultaneously
  - We just need to mix StarPU and MPI
  - Several applications
    - TPACF
    - LU decomposition
    - Stencil computation (e.g. Wave Propagation)
  - Experiments on the AC Cluster from NCSA
    - 4 GPU quad-core nodes

Using raw MPI+StarPU integration

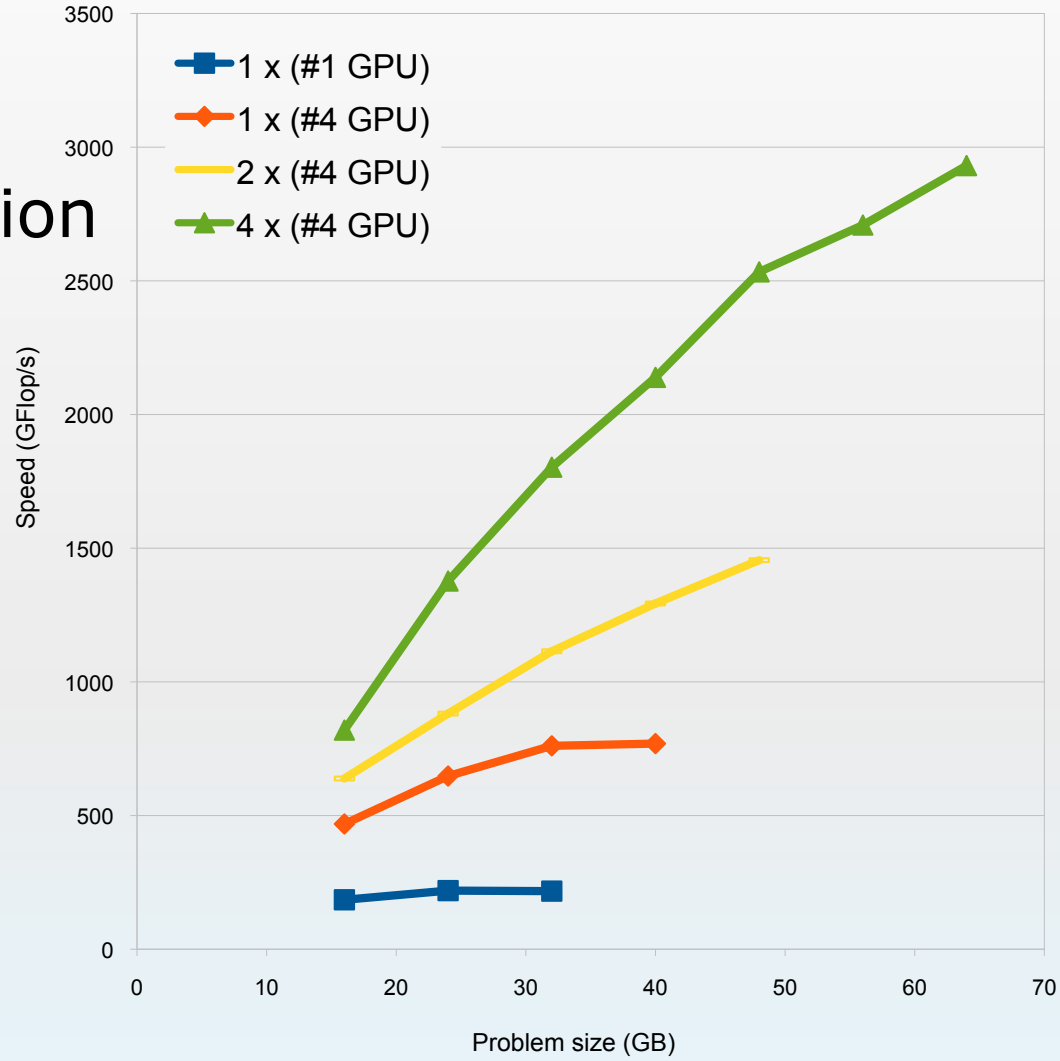
Without going to a full DSM system

- Keep MPI SPMD style
  - Static distribution of data (at the moment)
    - No load balancing between MPI processes
- StarPU scope limited to shared-memory nodes
- Inter-process data dependencies
  - MPI communications triggered by StarPU data availability
    - StarPU memory management system provides support
      - MPI datatypes

LU with MPI+StarPU

Performance

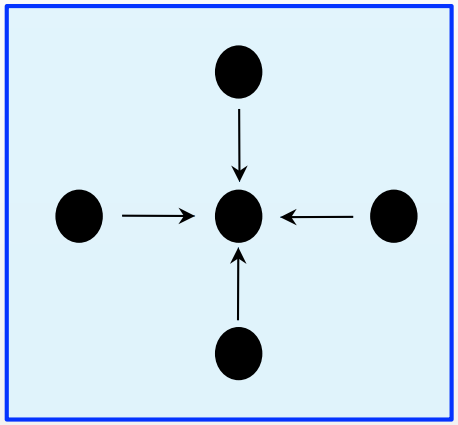
- LU decomposition
  - MPI + multi-GPU
- MPI Cyclic-distribution
  - ~ SCALAPACK
  - No pivoting !
- Future work
  - Integrate into D-PLASMA

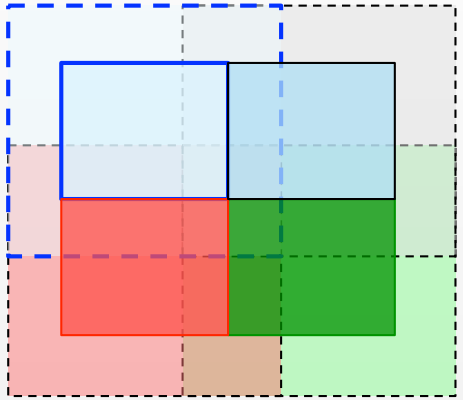


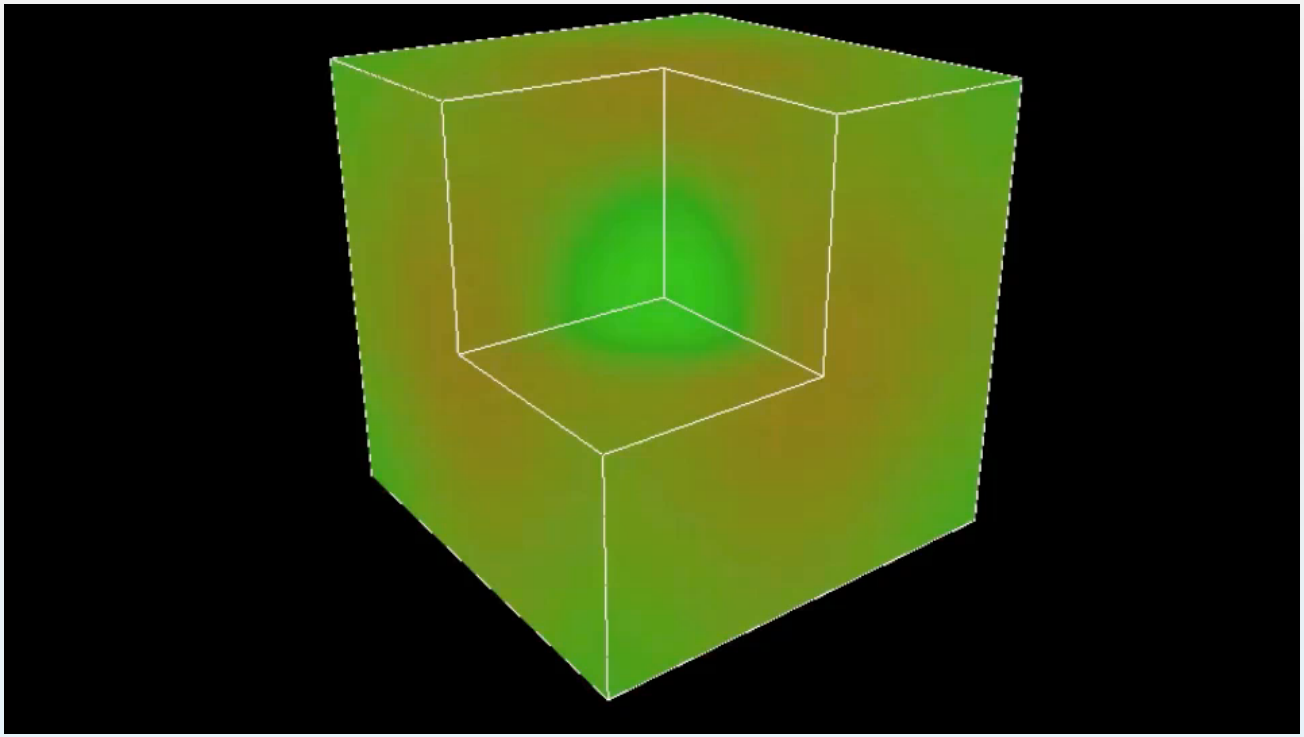
Wave propagation

Stencil computation

- It’s all about data movements
  - Prefetching
  - Asynchronism





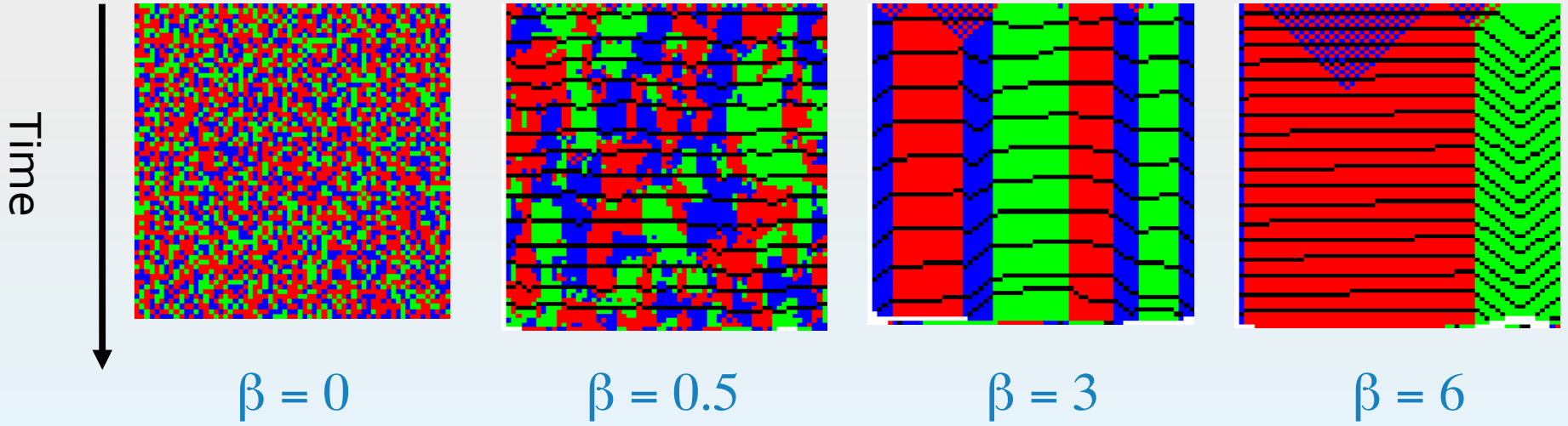


Wave propagation

Can a dynamic scheduler compete with a static approach?

- Load balancing *vs* data stability
  - We estimate the task cost as

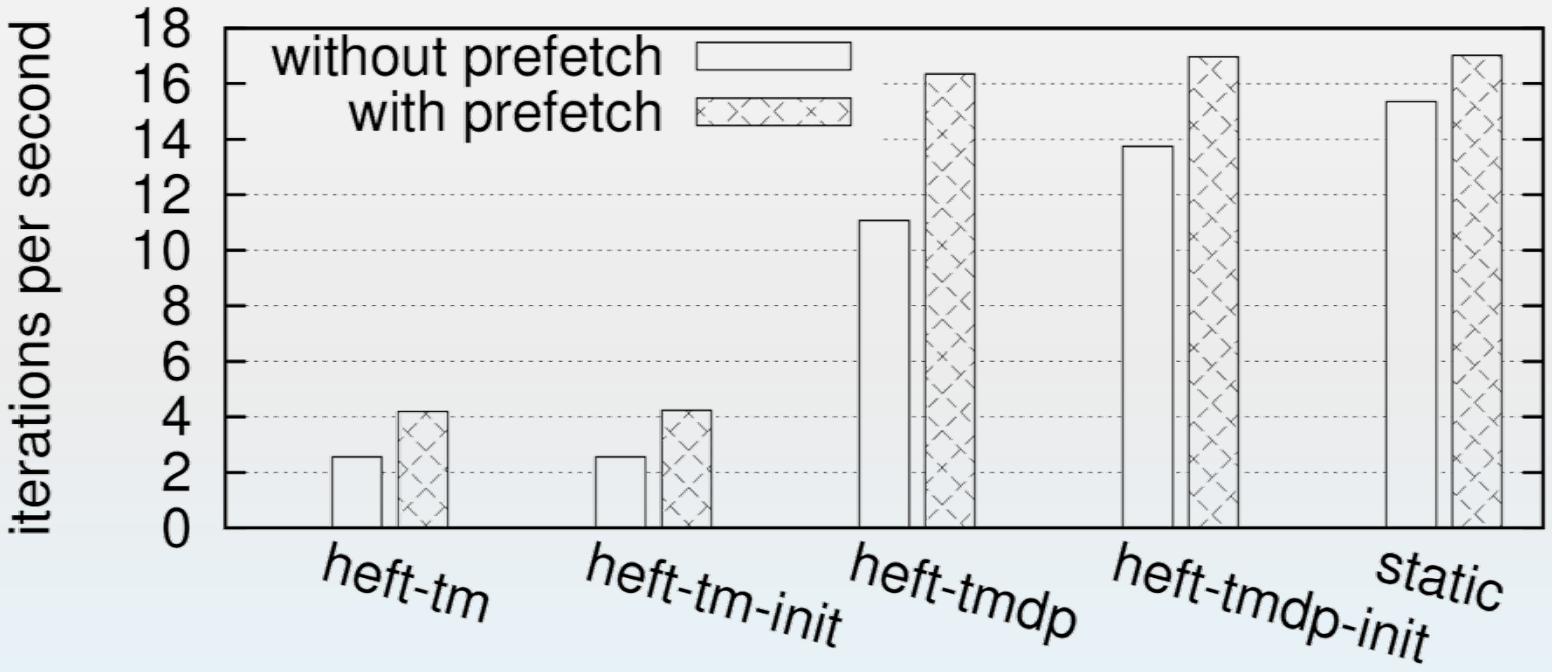
    α compute + β transfer
  - Problem size: 256 x 4096 x 4096, divided into 64 blocks
    - Task distribution (1 color per GPU)
    - Dynamic scheduling can lead to stable configurations



Wave propagation

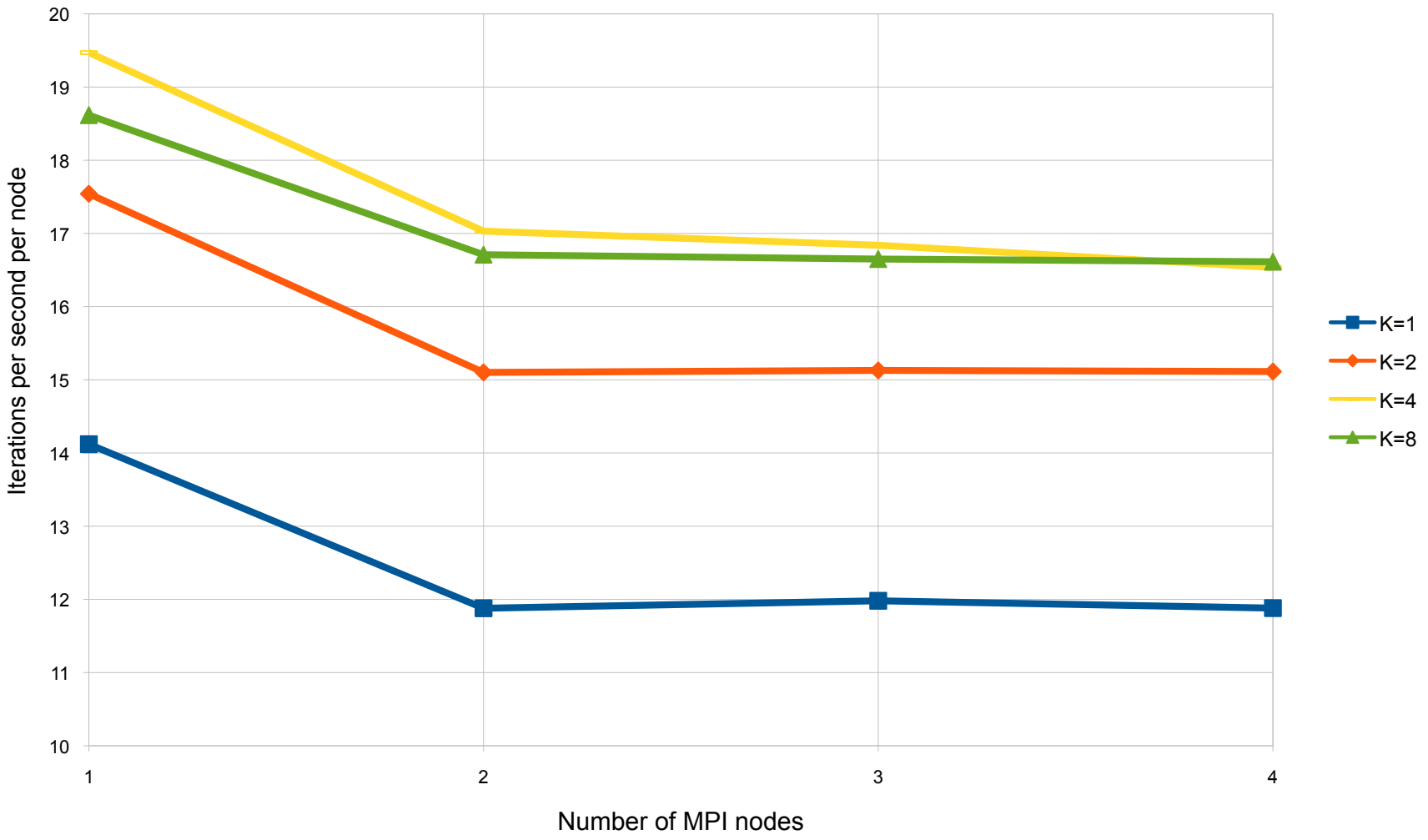
Performance

- Impact of scheduling policy
  - 3 GPUs (FX5800) – no CPU used
  - 256 x 4096 x 4096 : 64 blocks
  - Speed up = 2.7 (2 PCI 16x + 1 PCI 8x config)



Wave propagation

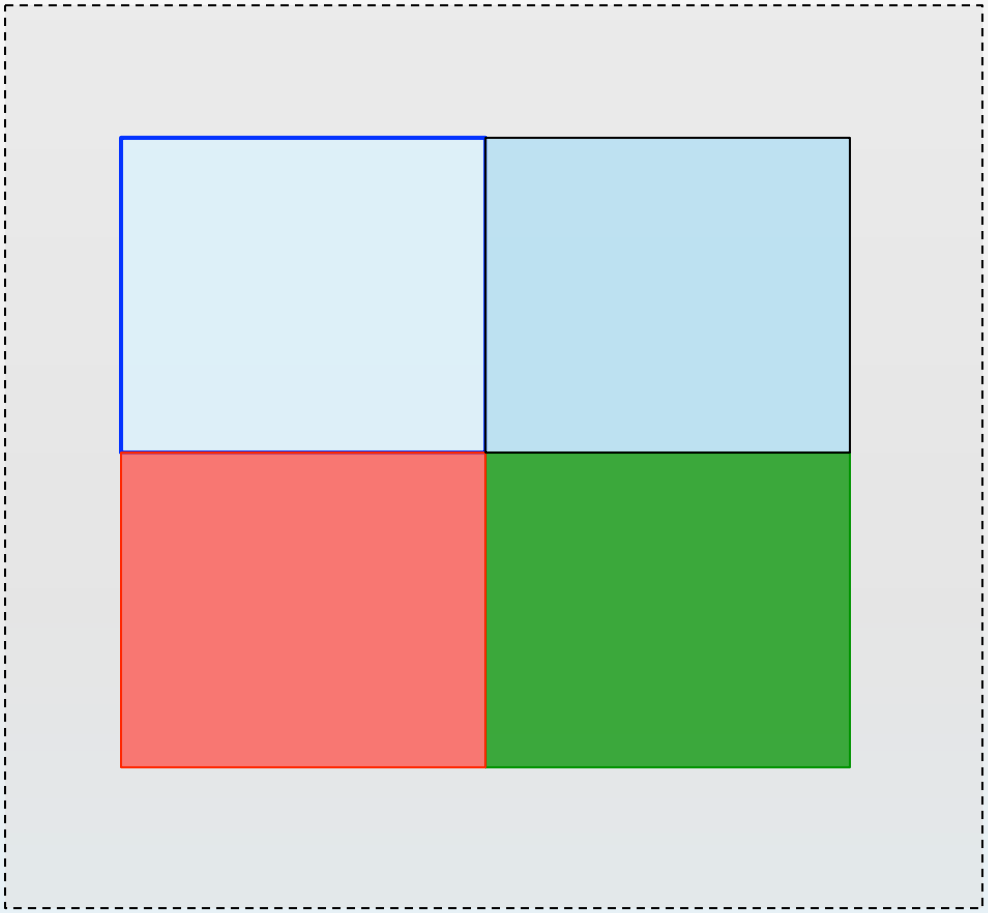
Behavior on several cluster nodes



Towards parallel tasks on CPUs

Going further

- MPI + StarPU + OpenMP
  - Many algorithms can take advantage of shared memory
  - We can't seriously "*taskify*" the world!
- The Stencil case
  - When neighbor tasks can be scheduled on a single node
    - Just use shared memory!
    - Hence an OpenMP stencil kernel



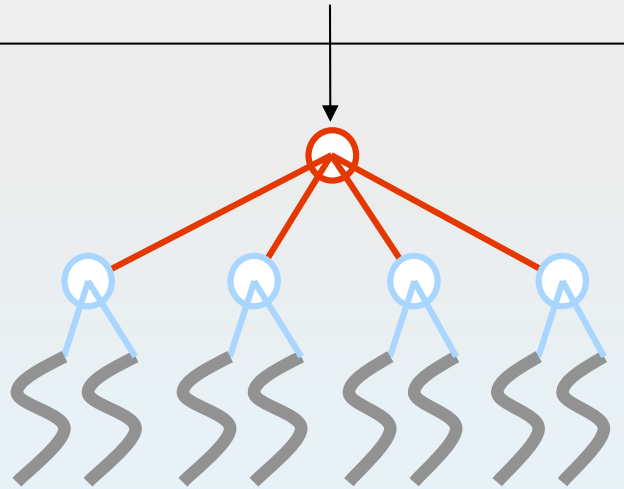
Integrating StarPU and Multithreading

How to deal with parallel tasks on multicore?

- Mixing StarPU with
  - OpenMP
  - Intel TBB
  - Pthreads
  - Etc.

- Raises the Composability issue
  - Challenge = autotuning the number of threads per parallel region

```
void work()
{
  ...

#pragma omp parallel for
  for (int i=0; i<MAX; i++)
    {
      ...
#pragma omp parallel for
num_threads (2)
      for (int k=0; k<MAX; k++)
        ...
    }
}
```



Main plot

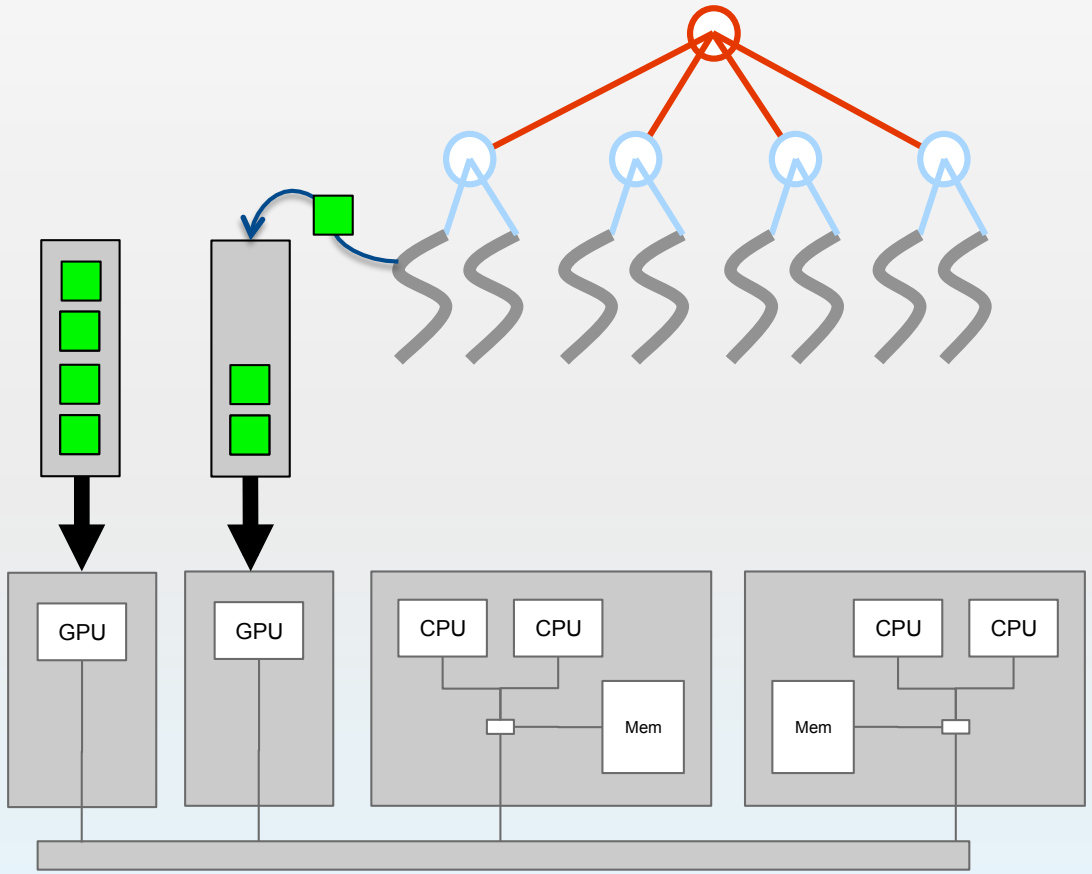
Composability is actually the biggest challenge

- Whatever your programming model, you need a runtime system able to handle communication, multitasking, I/O, etc.
  - It should also make it possible to mix different execution models
    - In Indirect Hybridization I trust!
- Up to now, we have designed separate *multithreaded* runtime systems for
  - Multicore machines
  - Accelerator
  - Clusters
- Can we easily put it all together?
  - Only a matter of using a common threads library?
  - Early experiments on multi-GPU clusters

Integrating StarPU and Multithreading

Integrating tasks and threads

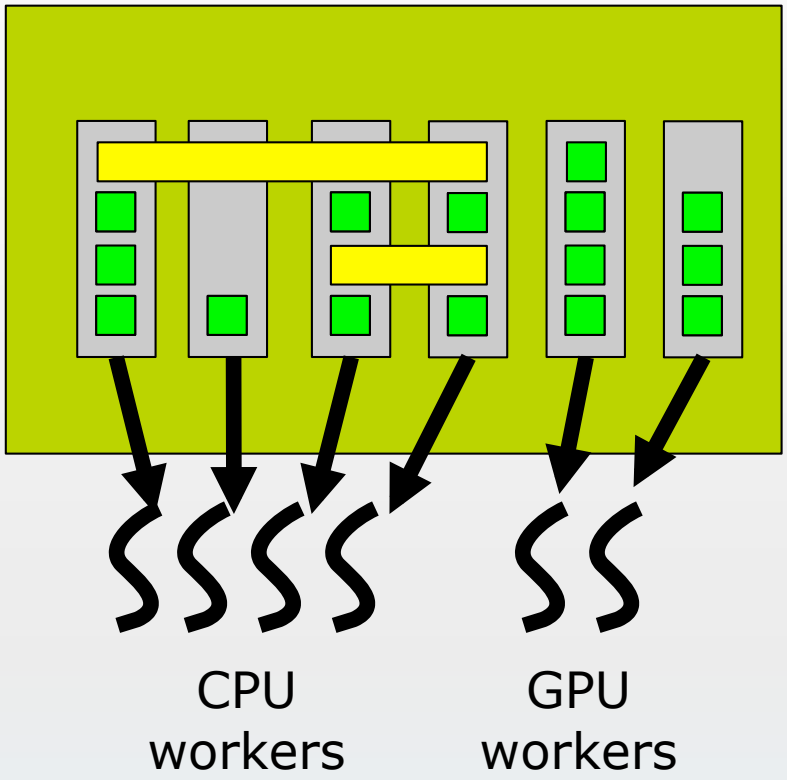
- First approach
  - Use an OpenMP main stream
    - Suggested (?) by recent parallel language extension proposals
      - E.g. Star SuperScalar (UPC Barcelona)
      - HMPP (CAPS Enterprise)
    - Implementing scheduling is difficult
      - Much more than a simple offloading approach…



Integrating StarPU and Multithreading

Integrating tasks and threads

- Alternate approach
  - Let StarPU spawn OpenMP tasks
    - Performance modeling would still be valid
    - Would also work with other tools
      - E.g. Intel TBB
    - How to find the appropriate granularity?
      - May depend on the concurrent tasks!
    - StarPU tasks = first class citizen
      - Need to bridge the gap with existing parallel languages



High-level integration

Generating StarPU code out of StarSs (Sylvain Gault)

- Experiments with
  - StarSs [UPC Barcelona]
- Writing StarSs +OpenMP code is easy
  - Platform for experimenting hybrid scheduling
    - OpenMP + StarPU

```
#pragma css task inout(v)
void scale_vector(float *v, float a, size_t n);

#pragma css target device(smp) implements (scale_vector)
void scale_vector_cpu(float *v, float a, size_t n) {
        int i;
        for (i = 0; i < n; i++)
                v[i] *= a;
}

int main(void)
{
        float v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        size_t vs = sizeof(v)/sizeof(*v);

#pragma css start

scale_vector(v, 4, vs);
...
```

Future work

- Bridge the gap with parallel languages
  - StarPU+OpenMP as a target for the StarSs language
    - Kernel generation
    - Data representation
  - StarPU+OpenMP+MPI as a target for XcalableMP?
- Enhance cooperation between runtime systems and compilers
  - Granularity, runtime support for “divisible tasks”
  - Feedback for autotuning software
  - [PEPPHER European project]

Thank you!

- More information about StarPU

  http://runtime.bordeaux.inria.fr