Towards a common runtime system for multicore machines, accelerators and clusters?

Pr. Raymond Namyst
RUNTIME group, INRIA Bordeaux

WPSE 2010

The RUNTIME Team

High Performance Runtime Systems for Parallel Architectures

- Main research directions
  - Exploiting shared memory machines
    - Thread scheduling over hierarchical multicore architectures
    - 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
- Typical client software
  - OpenMP, MPI

Main plot

A matter of composability

- 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

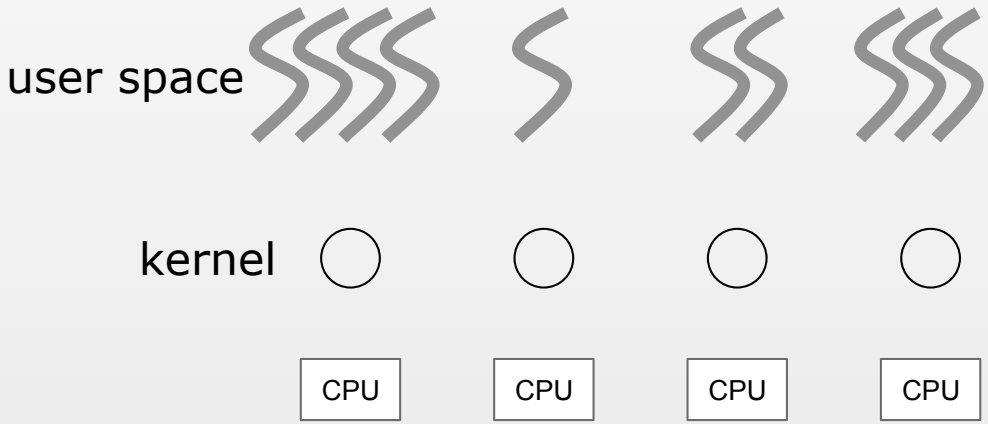
The early days of clusters

- High Speed Networks
  - Many different hardware and protocols
- Few processors per node
  - Uniform memory access times

Thread Scheduling over Multiprocessors

Processor virtualization

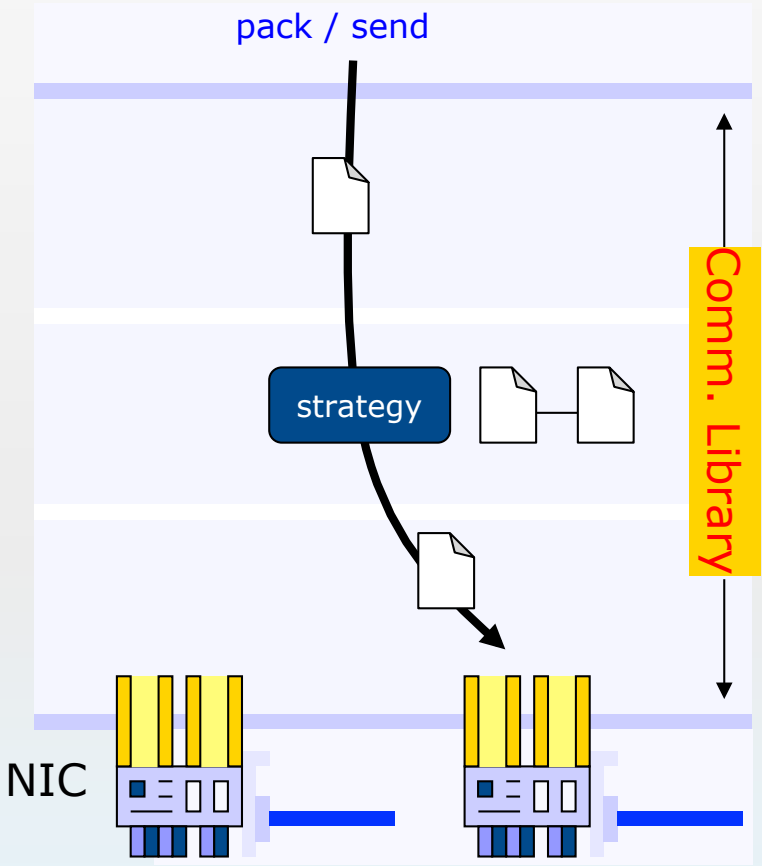
- *"Let the application use an arbitrary number of threads!"*
- Two-level N:M multithreading
  - One kernel thread per processor
  - Many user-level threads per kernel thread
  - *Scheduler Activation*-like extension
- Specifically designed for HPC
  - A lot of unuseful POSIX features are optional
  - Polling facilities are available for I/O management software
- Efficient
  - Performance of basic operations approx. 10x faster than NPTL



Communication over high-speed networks

The traditional way

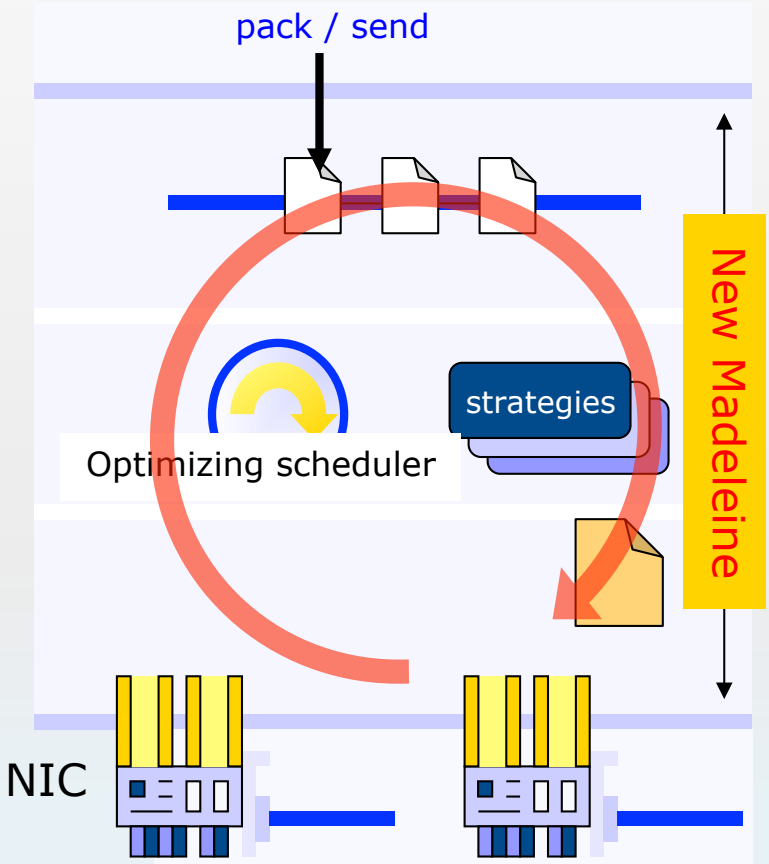
- Target = high speed networks
  - Myrinet, Quadrics, IB, SCI
- Traditional communication libraries focus on raw performance
  - Latency, bandwidth
- We can still improve MPI!
  - Deal with
    - Non-trivial communication schemes
    - Complex message layout
      - datatypes
  - Take advantage of today's multicore architectures



Communication over high-speed networks

Using threads to build an active communication engine

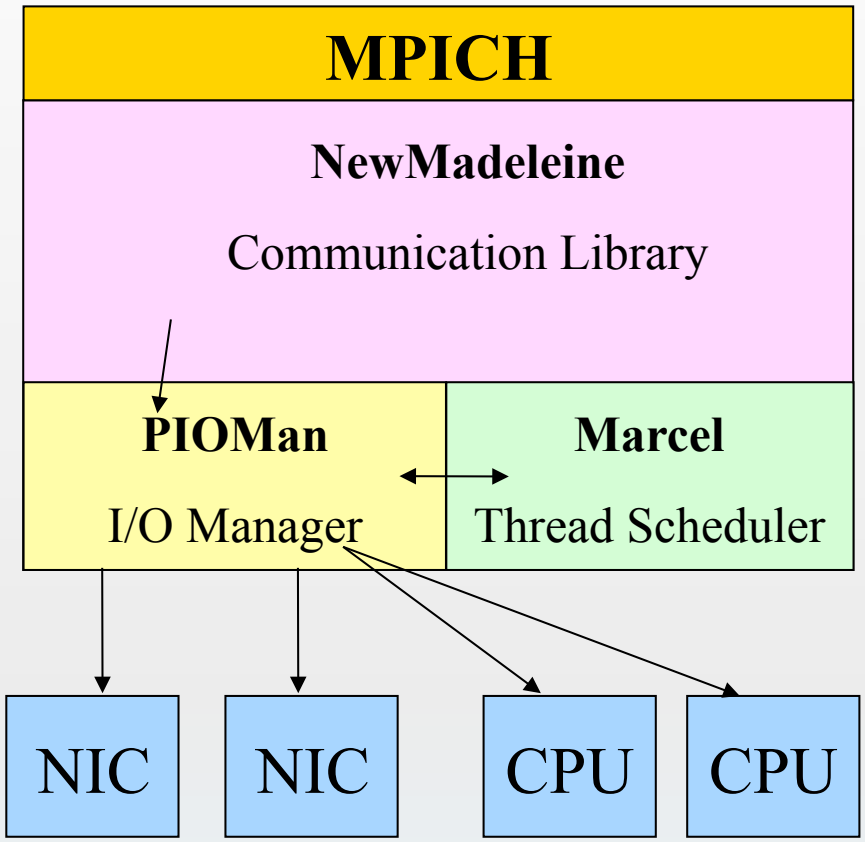
- Just-in-time Optimization
  - Separation of application activity / state of network links
  - Optimizing scheduler driven by NIC activity
  - Opportunistic scheduling of data segments
- The NewMadeleine communication engine
  - Multithreaded
  - Strategies = plug-ins



Communication over high-speed networks

Using threads to improve communication

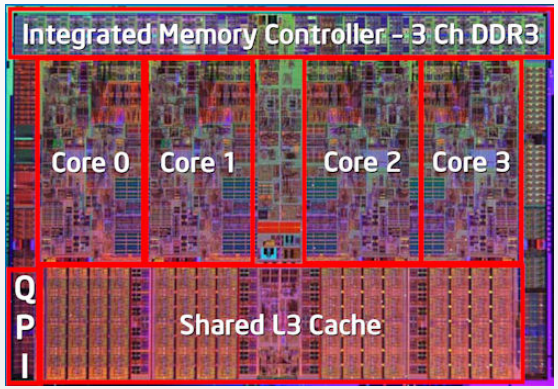
- Opportunistic, aggressive optimizing strategies
  - Out-of-order transfers, coalescing, …
- Heterogeneous multirail networks
  - Accurate splitting of data segment to feed available network links
  - MPICH/NewMadeleine [With ANL]
- Towards fully parallel communication engines
  - Everything is a task
  - Offloading on idle cores
  - Best-effort optimization algorithms
  - [with University of Tokyo]

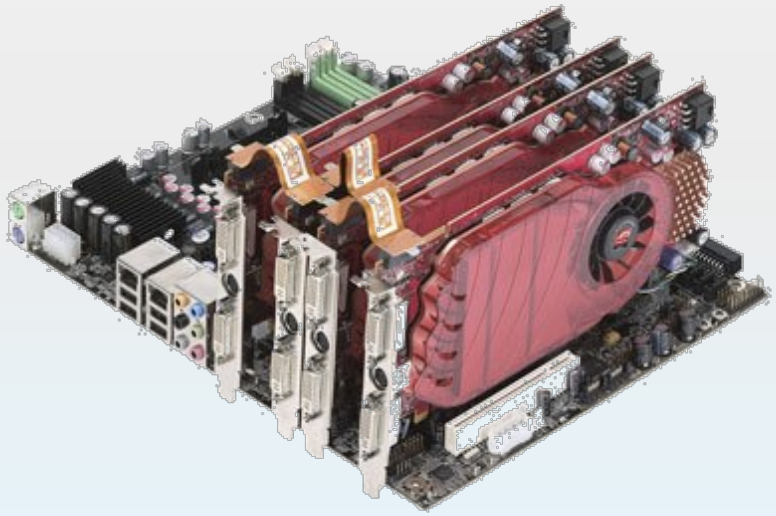


Recent evolution of hardware

Towards multi-GPU clusters

- Networks
  - No fundamental change
- Multicore chips
  - Back to complex memory hierarchies
    - Shared caches
    - NUMA factors
  - Clusters can no longer be considered as "flat sets of processors"
- Accelerators
  - Very powerful SIMD accelerators
  - Specific instruction set
  - No hardware memory consistency

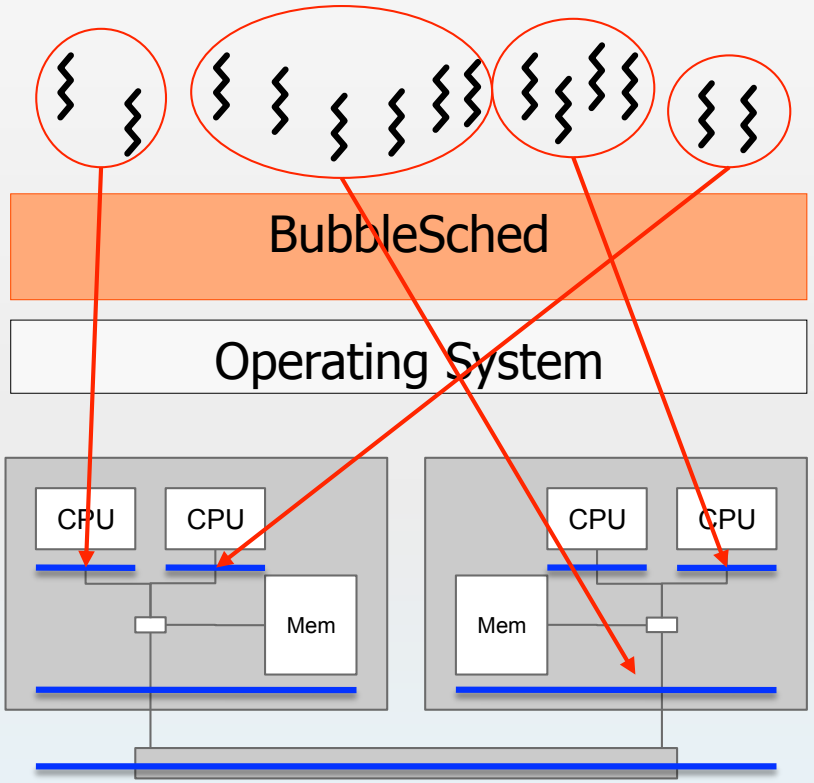




Thread Scheduling over Multicore Machines

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

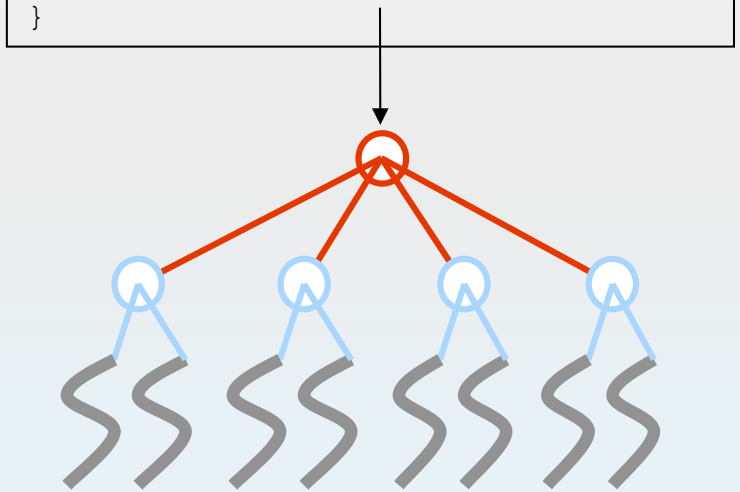


Thread Scheduling over Multicore Machines

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

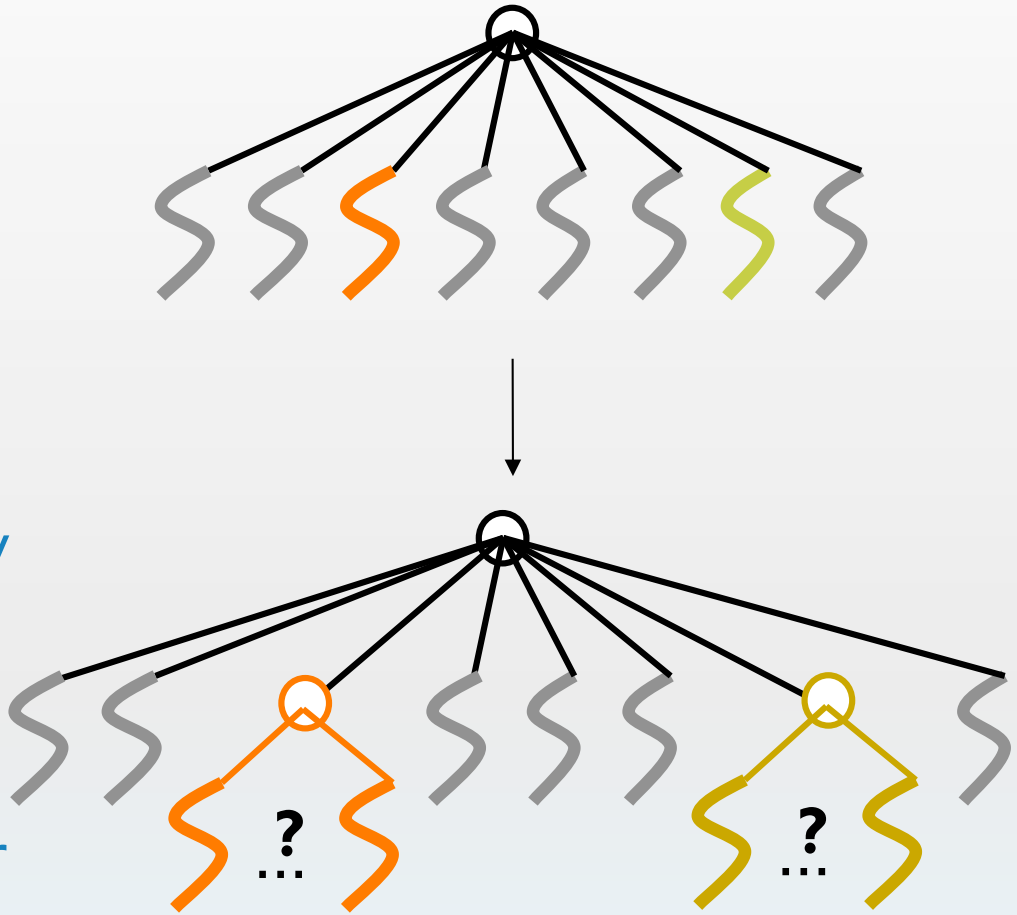
```
void work()
{
  ...

#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++)
        ...
    }
}
```



Towards adaptive parallel regions

Ongoing work

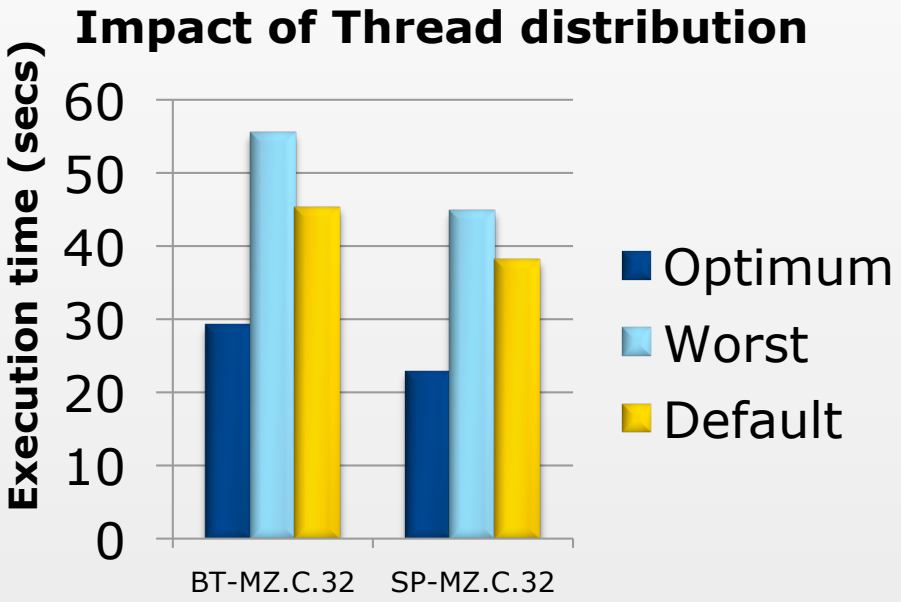
- How to deal with dynamic, concurrent parallelism?
  - E.g. concurrent calls to multithreaded BLAS
- Adaptive solution
  - Get rid of OMP_NUM_THREADS
  - Use performance history table
    - Hardware counters
  - Give more cores to regions that scale better

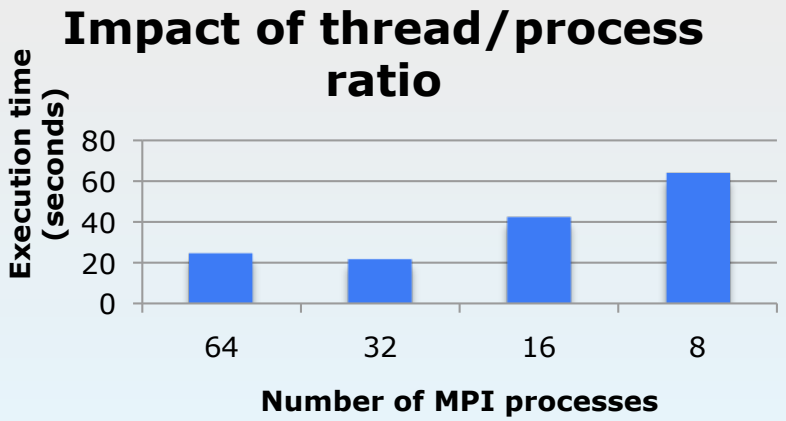


Mixing OpenMP with MPI

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

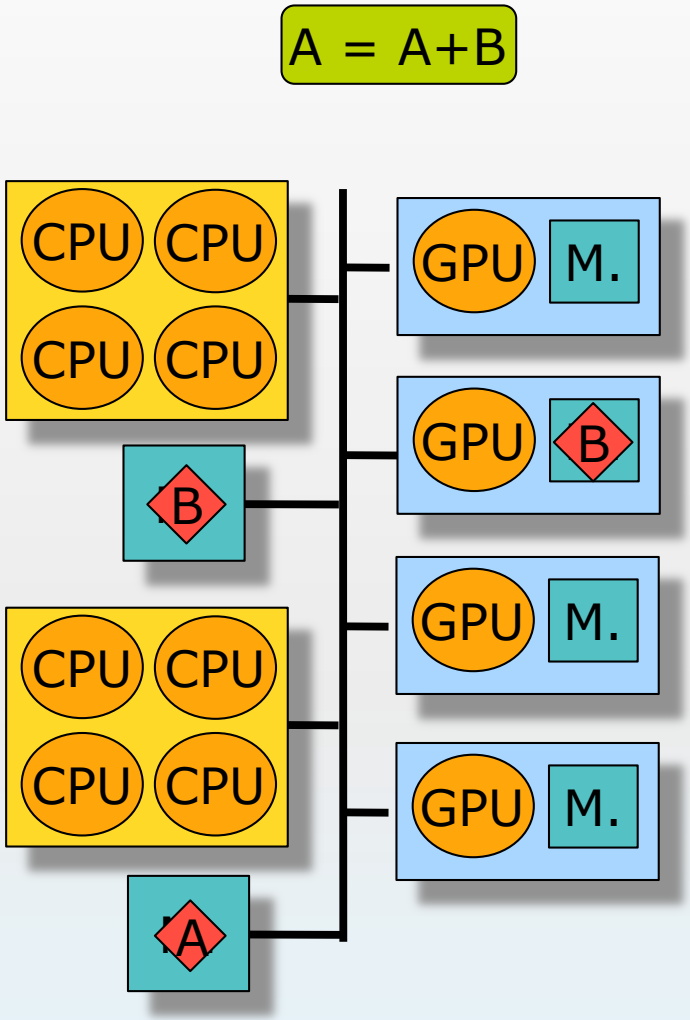




Dealing with heterogeneous architectures

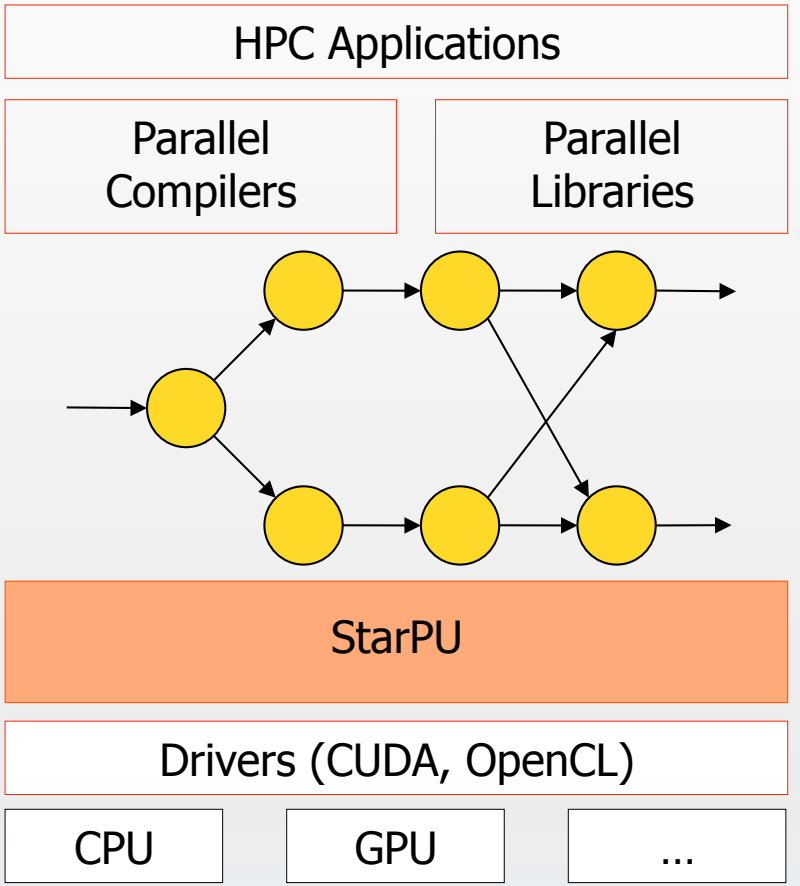
The StarPU runtime system

- CPU+GPU+SPU = *PU
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous cores
    - Scheduling ≠ offloading
  - Avoid unnecessary data transfers between accelerators
    - Software DSM for heterogeneous machines



Dealing with heterogeneous architectures
Memory Management

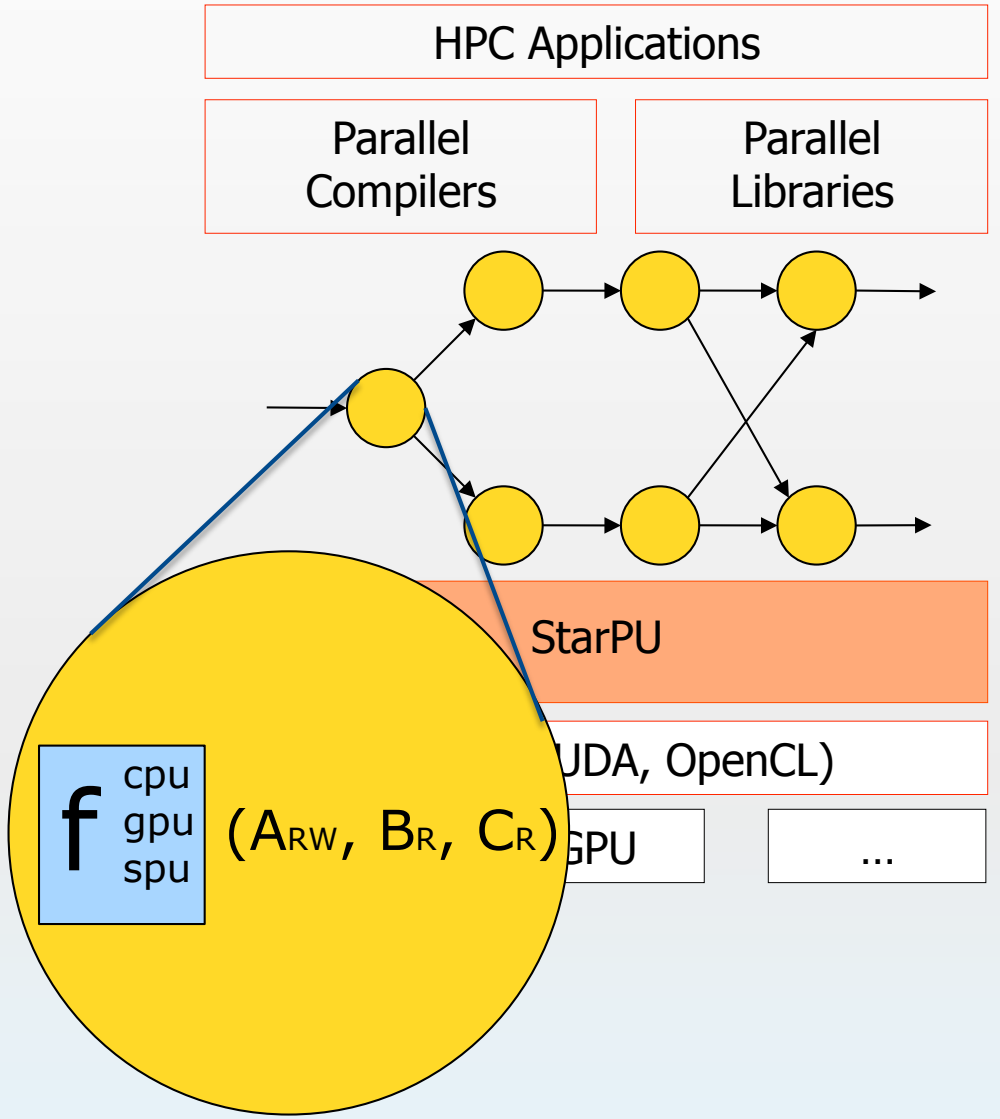
- StarPU provides a Virtual Shared Memory subsystem
  - Weak consistency
  - Replication
  - Single writer
  - High level API
    - Partitioning filters
- Input & ouput of tasks = reference to VSM data



Dealing with heterogeneous architectures

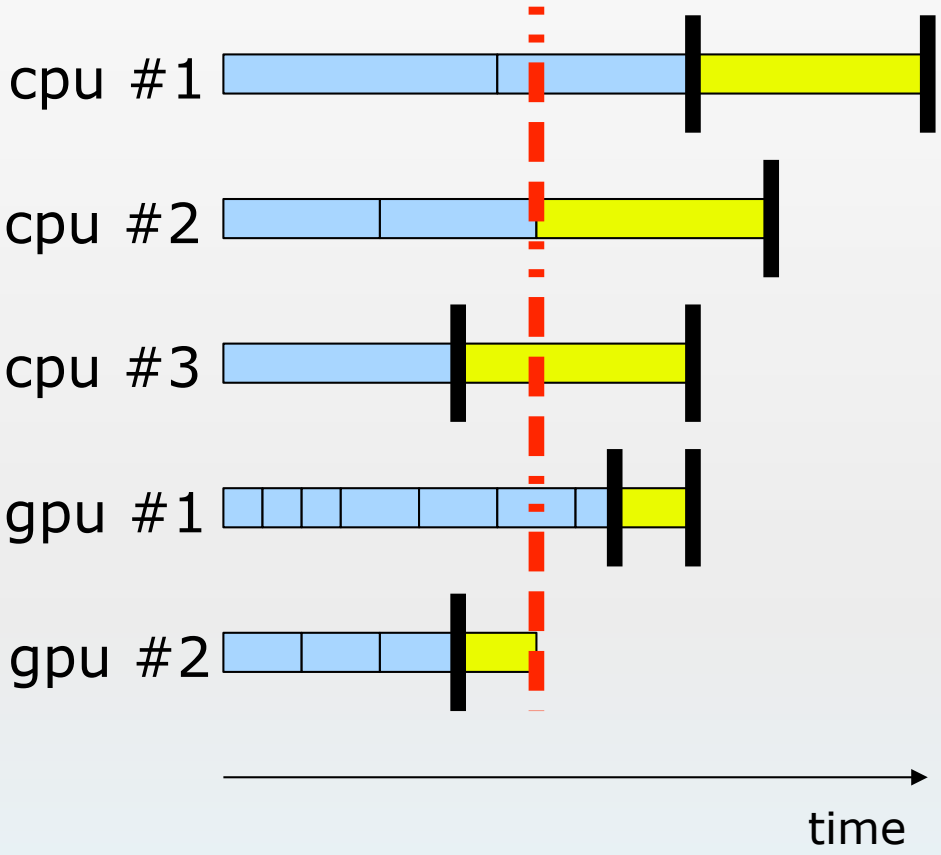
Task scheduling

- 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



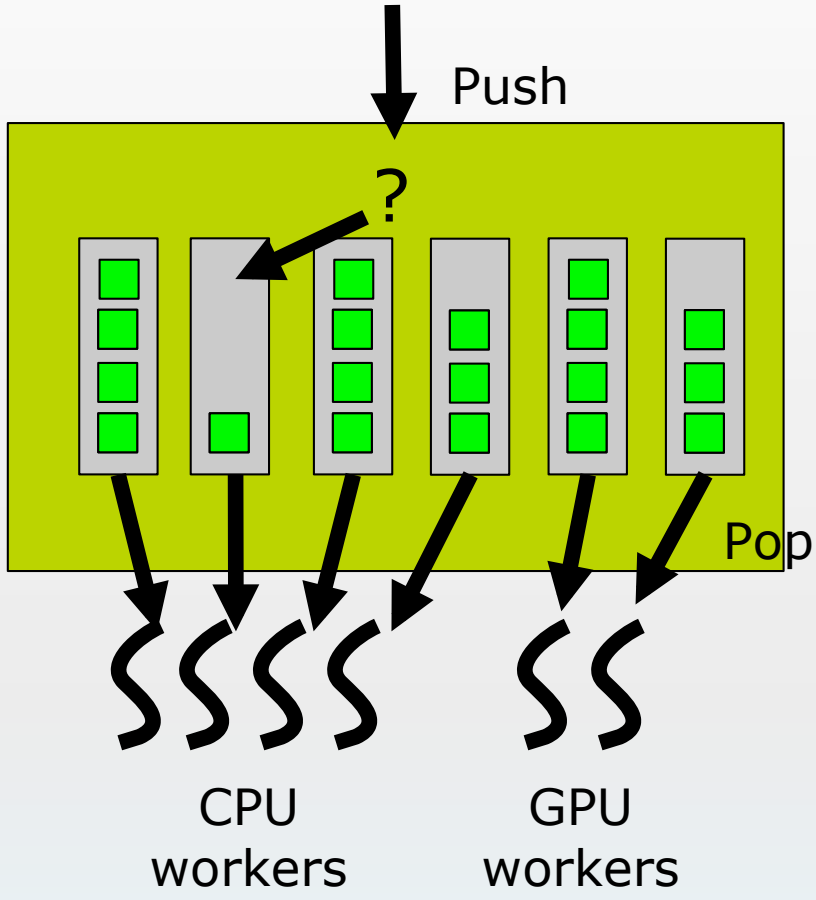
Dealing with heterogeneous architectures
Performance prediction

- 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



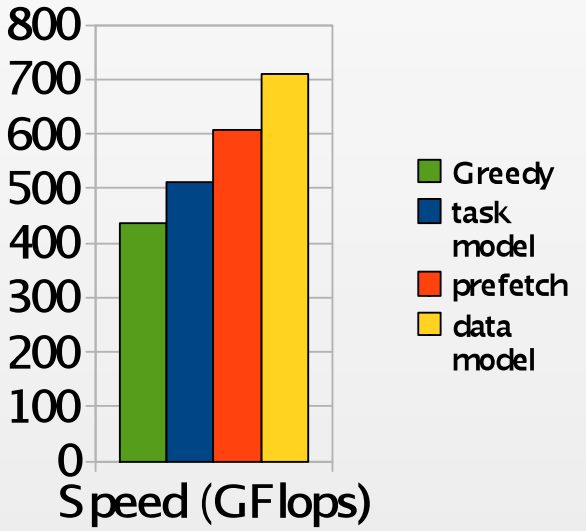
Dealing with heterogeneous architectures
Scheduler internals

- 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

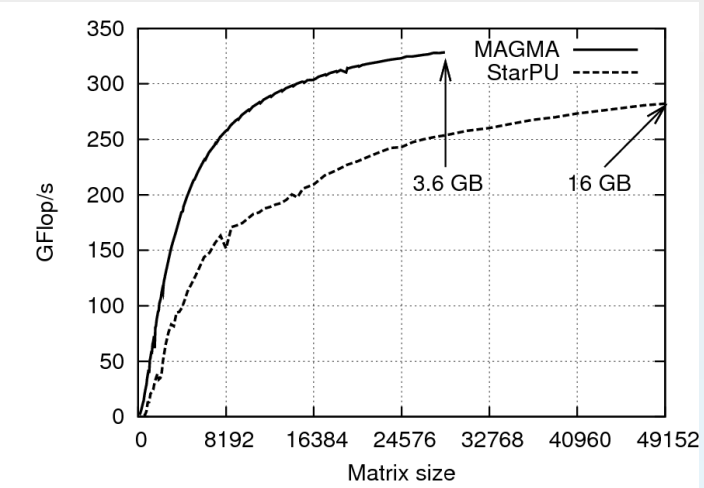


Dealing with heterogeneous architectures
Performance

- 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



Moving to multi-GPU clusters

Putting it all together

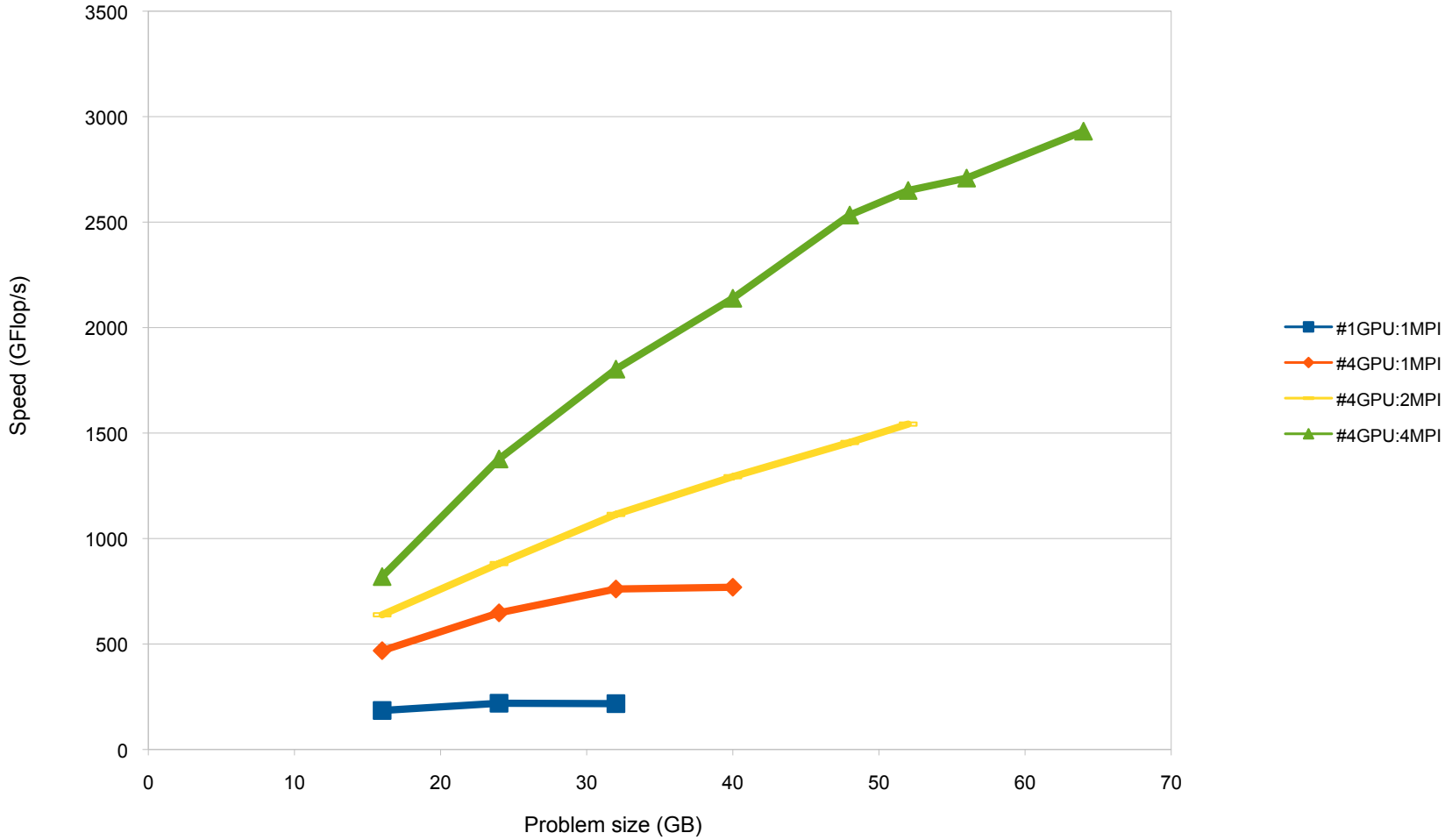
- MPI + StarPU
  - StarPU is able to use GPUs and CPUs simultaneously
  - We just need to mix StarPU and MPI
  - Two applications
    - LU decomposition
    - Stencil computation (e.g. Wave Propagation)
  - Experiments on the AC Cluster from NCSA
    - 4 GPU quad-core nodes

Using raw MPI+StarPU integration

- Keep MPI SPMD style
  - Static distribution of data
    - 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

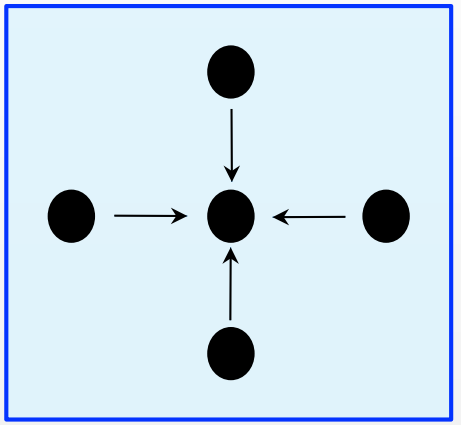
Performance

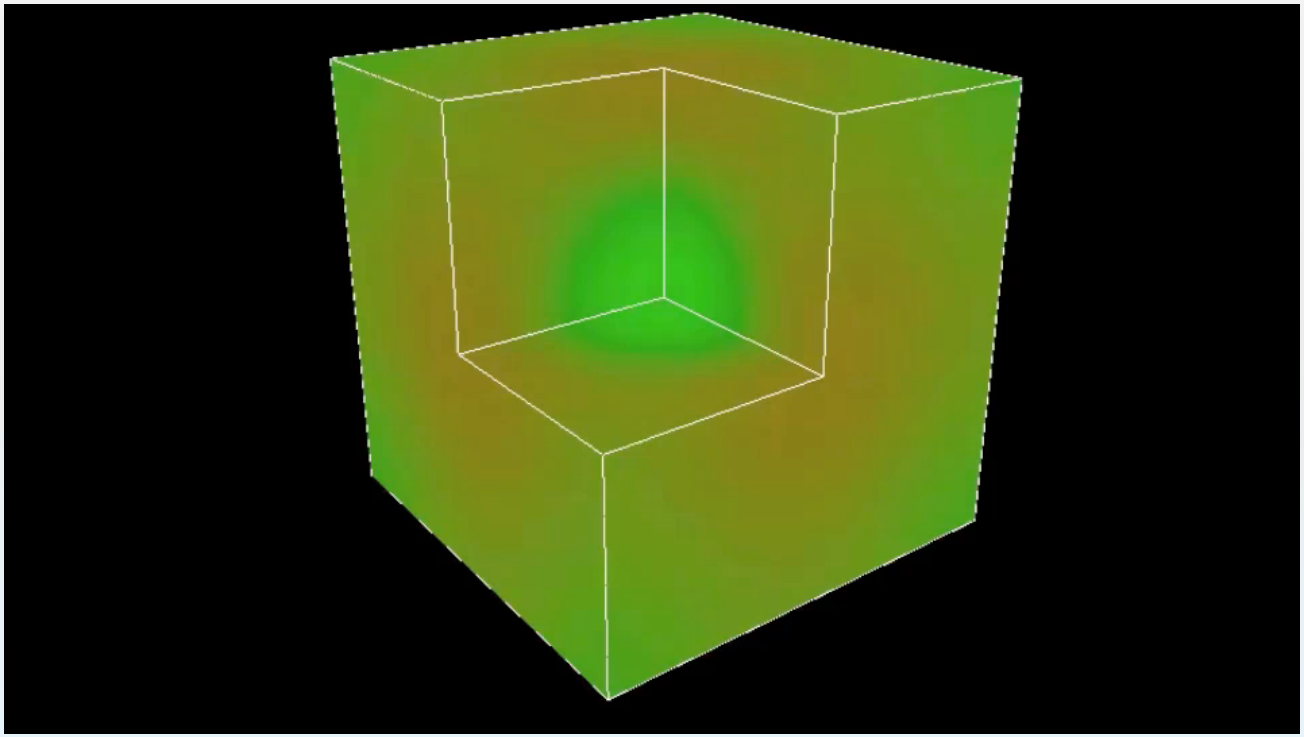


Wave propagation

Stencil computation

- It’s all about data movements
  - Prefetching
  - Asynchronism
  - [with UIUC/NCSA]

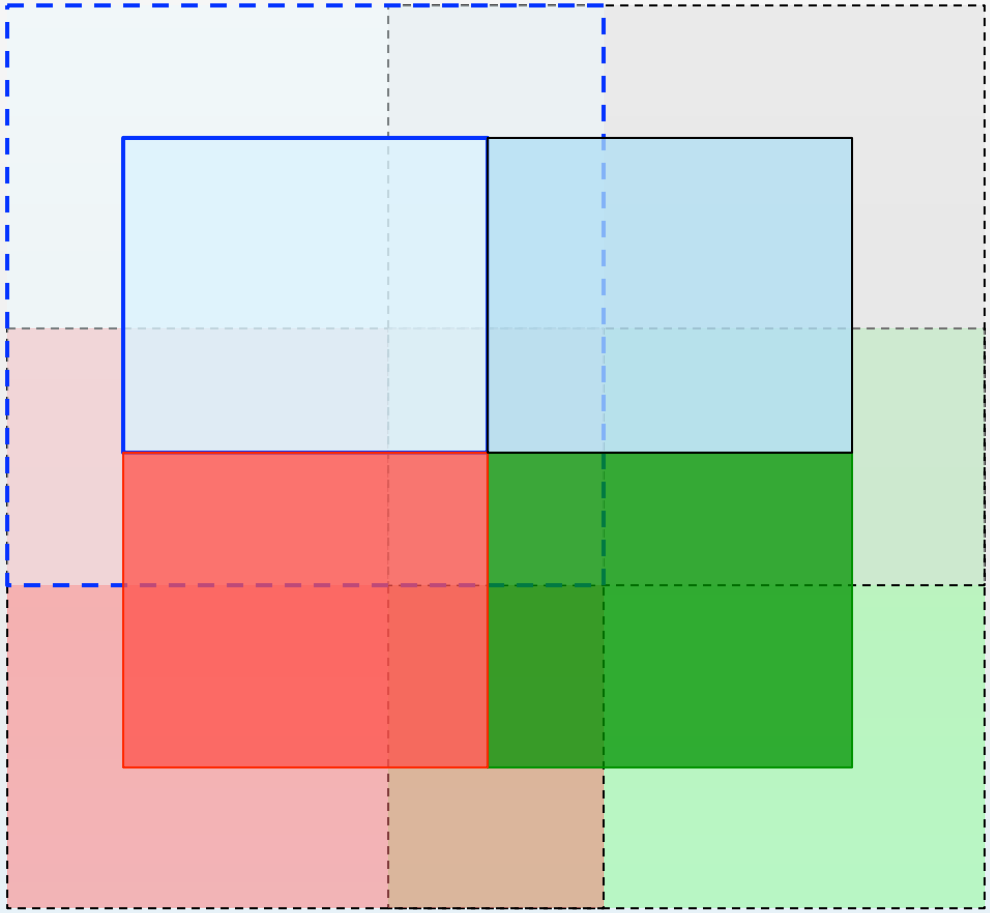




Wave propagation

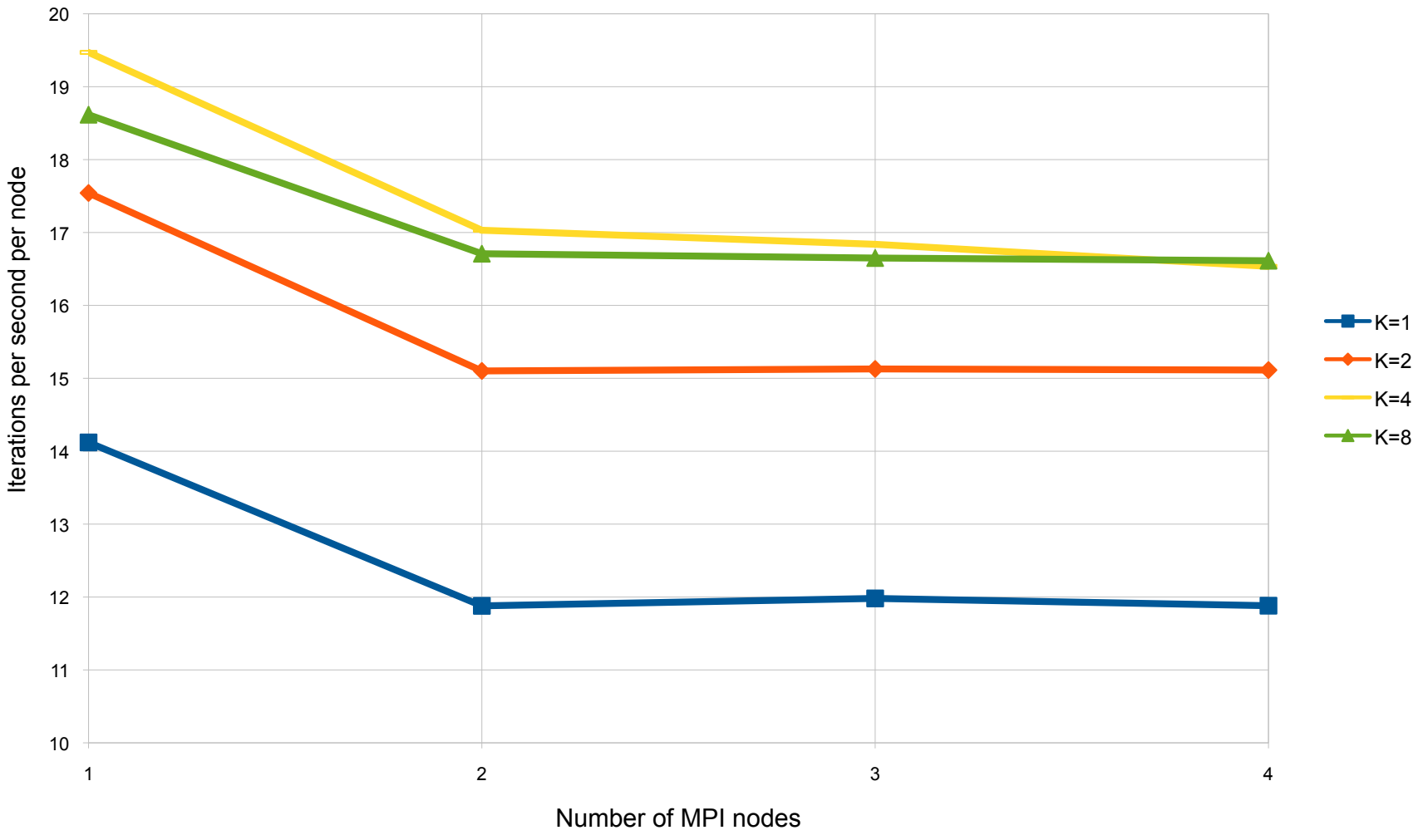
Using ghost cells

- Typical StarPU task decomposition
- Border thickness of k = k iterations without communication



Wave propagation

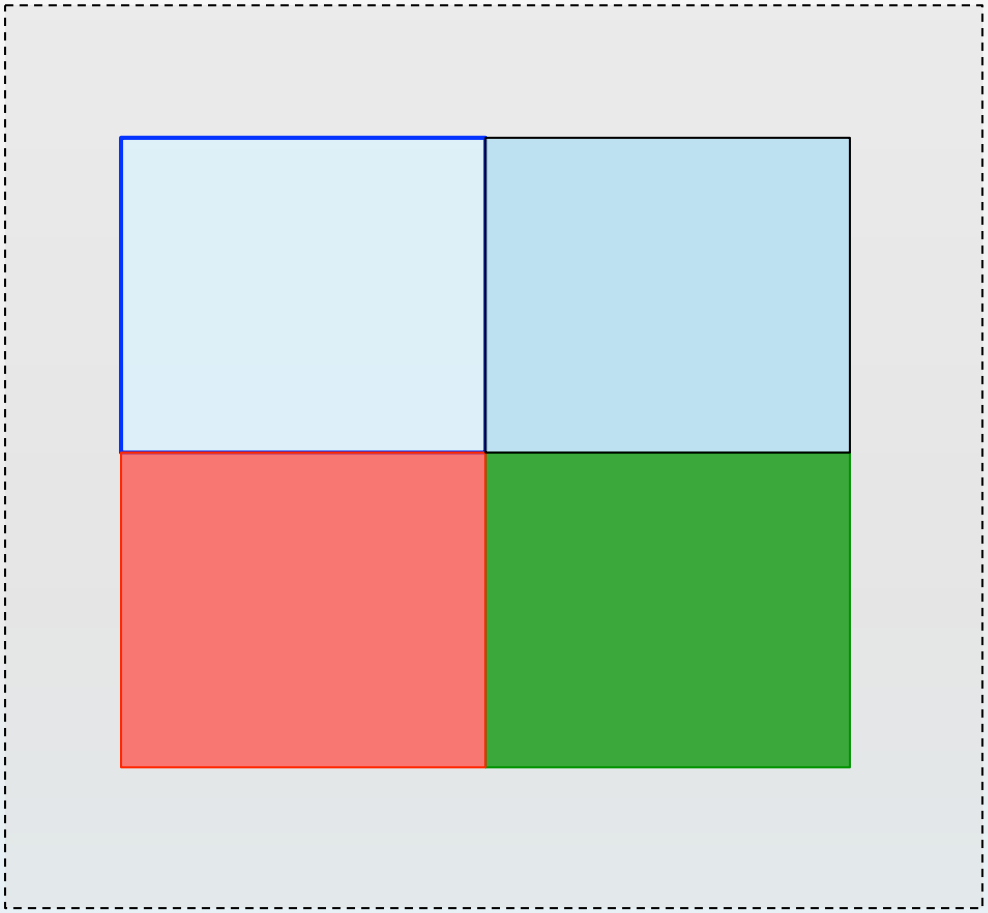
Performance



Towards parallel tasks on CPUs

Going further

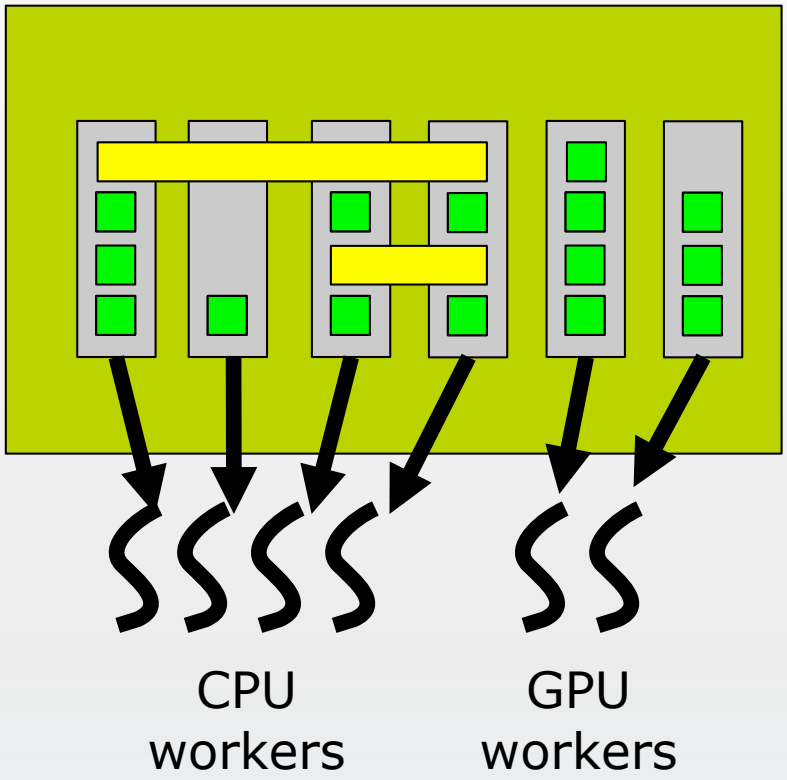
- MPI + StarPU + OpenMP
  - Many algorithms can take advantage of shared memory
  - We can't seriously "*taskify*" the world!
- The Stencil case
  - When neighbor tasks can be scheduled on a single node
    - Just use shared memory!
    - Hence an OpenMP stencil kernel



Open issues

Integrating tasks and threads

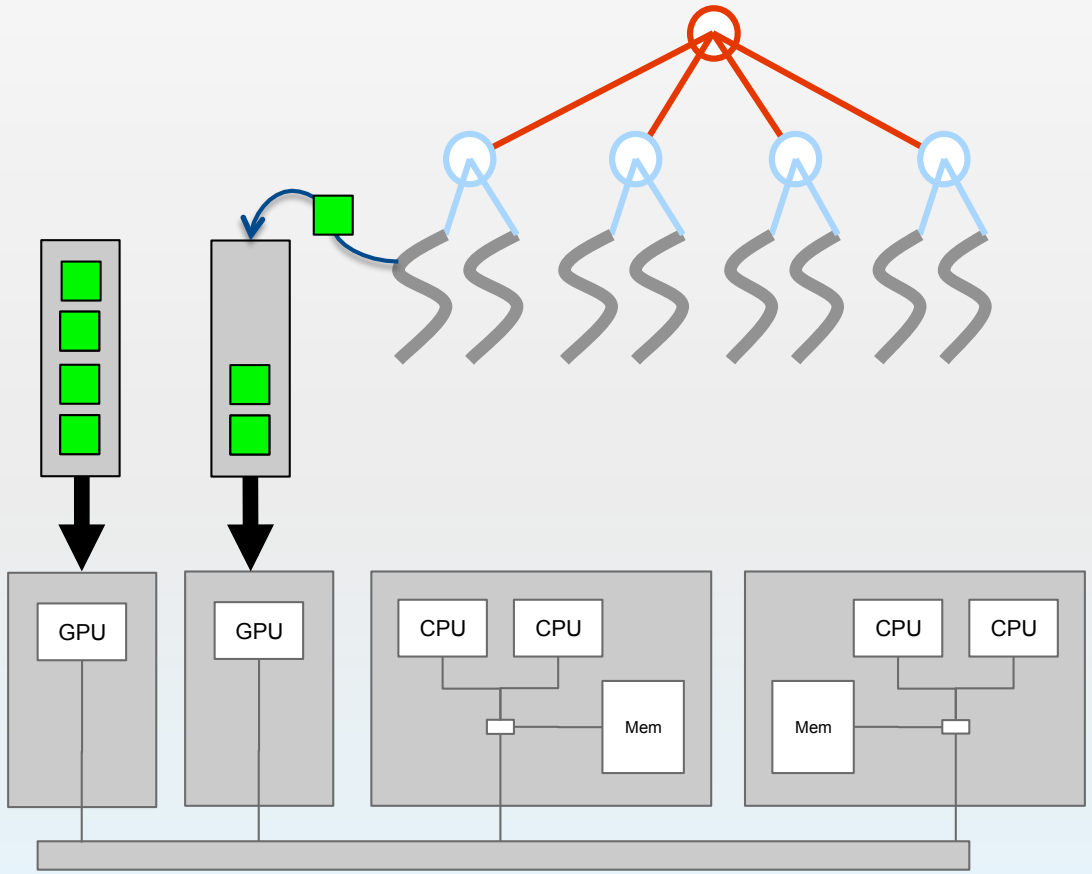
- First approach
  - Let StarPU spawn OpenMP tasks
    - Performance modeling would still be valid
    - Would also work with other tools
      - E.g. Intel TBB
    - How to find the appropriate granularity?
      - May depend on the concurrent tasks!
    - StarPU tasks = first class citizen
      -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



Open issues

Integrating tasks and threads

- Second 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 much more difficult
      - More than a simple offloading approach…



Future work

- More experiments with both solutions
- 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 Runtime

  http://runtime.bordeaux.inria.fr