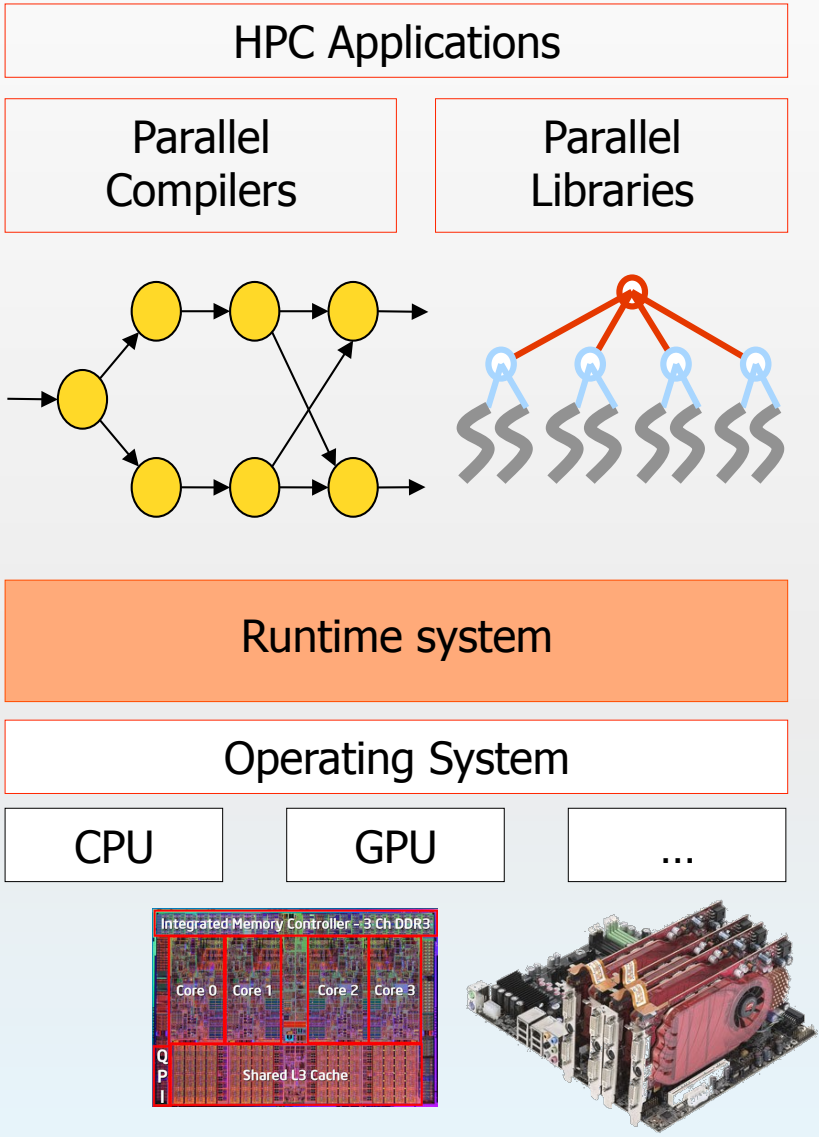
Overview of research activities

Toward "portability of performance"

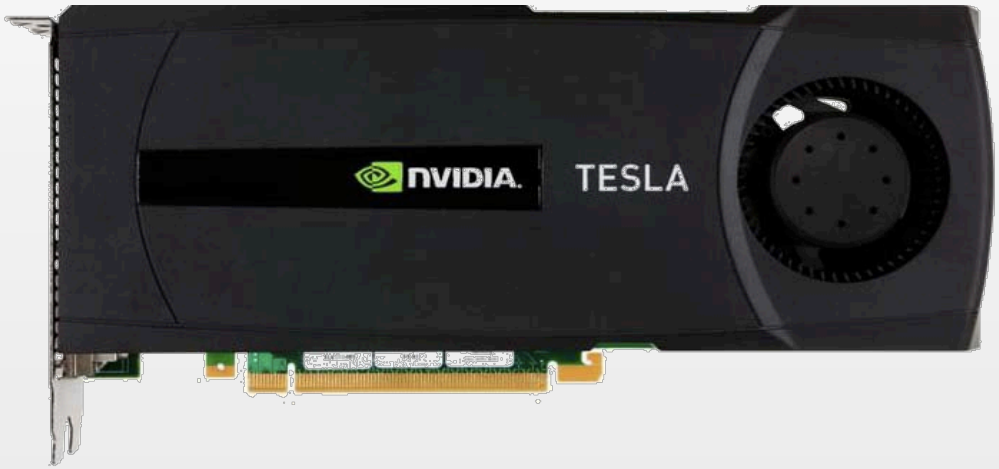
- Do dynamically what can't be done statically
  - Understand evolution of architectures
  - Enable new programming models
  - Put intelligence into the runtime!
- Exploiting shared memory machines
  - Thread scheduling over hierarchical multicore architectures
    - OpenMP
  - Task scheduling over accelerator-based 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 + …



Heterogeneous computing is here

And portable programming is getting harder…

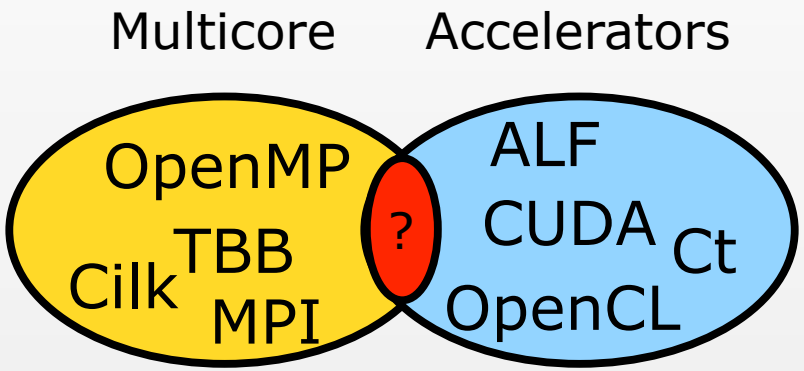
- GPU are the *new kids on the block*
  - Very powerful data-parallel accelerators
  - Specific instruction set
  - No hardware memory consistency
- Clusters featuring accelerators are already heading the Top500 list
  - Tianhe-1A (#1)
  - Nebulae (#3)
  - Tsubame 2.0 (#5)
  - Roadrunner (#?)
- Using GPUs as "side accelerators" is not enough
  - GPU = first class citizens

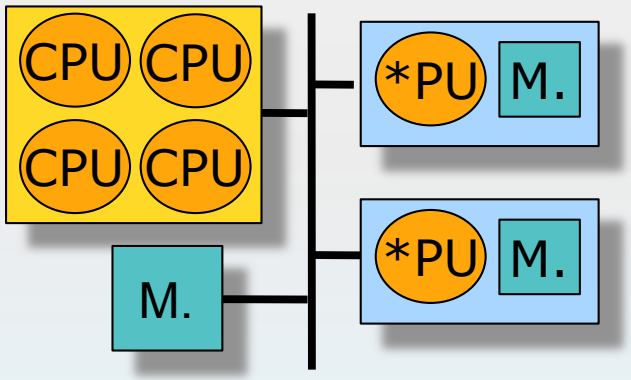


Heterogeneous computing is here

How shall we program heterogeneous clusters?

- The ~~hard~~ hybrid way
  - Combine different paradigms by hand
    - MPI + {OpenMP/TBB/???} + {CUDA/OpenCL}
  - Portability is hard to achieve
    - Work distribution depends on #GPU & #CPU per node…
      - Tools such as S-GPU may help!
    - Needs aggressive autotuning
  - Currently used for building parallel numerical kernels
    - MAGMA, D-PLASMA, FFT kernels





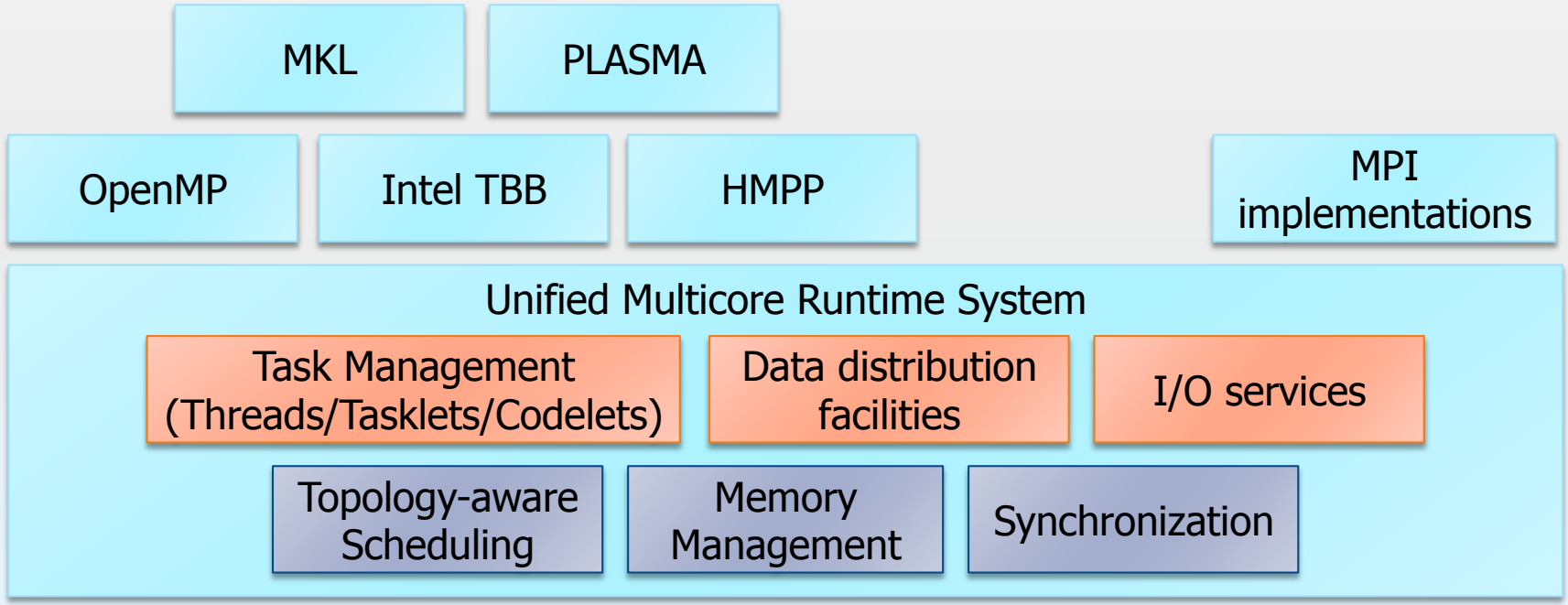
Heterogeneous computing is here

Mixing different paradigms leads to several issues

- Semantics issues
  - MPI and OpenMP don't mix easily
    - E.g. MPI communication inside parallel regions
    - Higher-level abstractions would help!
      - Think about domain-decomposition algorithms
- Resource allocation issues
  - Can we really use several hybrid parallel kernels simultaneously?
    - Ever tried to mix OpenMP and MKL?
    - Could be helpful in order to exploit millions of cores
  - It's all about composability
    - Probably the biggest challenge for runtime systems
      - Hybridization will mostly be indirect (linking libraries)
  - And with composability come a lot of related issues
    - Need for autotuning / scheduling hints

Runtime systems enabling composability
Background

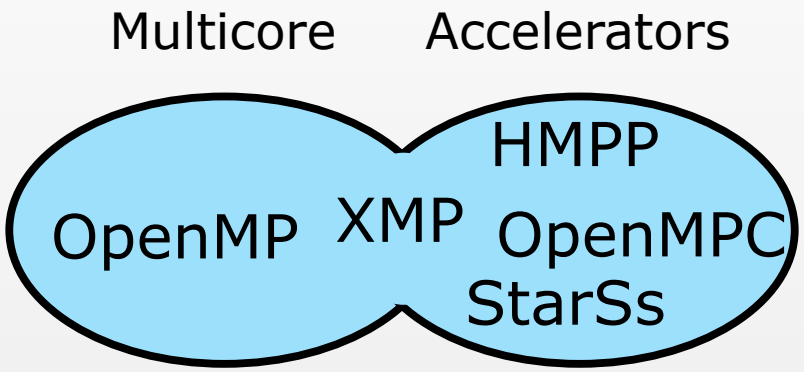
- So far, we've been working on providing a common runtime system for
  - MPI + (OpenMP)* = multiple OpenMP kernels mixed inside an MPI application
- Main features
  - Hierarchical thread scheduling (with potential oversubscription)
  - Topology-aware, adaptive parallelism
    - Give more cores to regions that scale better!
- Towards a common, unified runtime system?

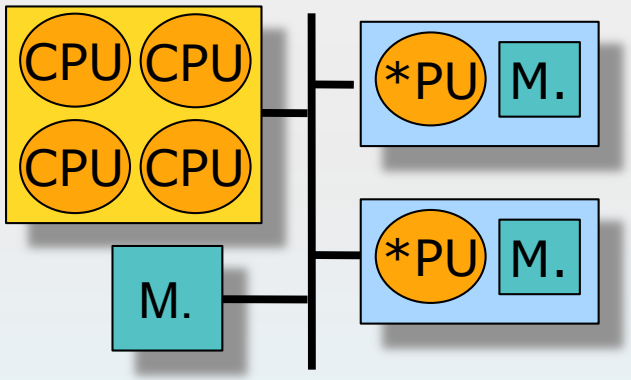


Heterogeneous computing is here (cont'd)

How shall we program heterogeneous clusters?

- The uniform way
  - Use a single (or a combination of) high—level programming language to deal with network + multicore + accelerators
  - Increasing number of directive-based languages
    - Use simple directives… and good compilers!
      - XcalableMP
        - PGAS approach
      - HMPP, OpenMPC, OpenMP 4.0
        - Generate CUDA from OpenMP code
      - StarSs
  - Much better potential for *composability…*
    - If compiler is clever!





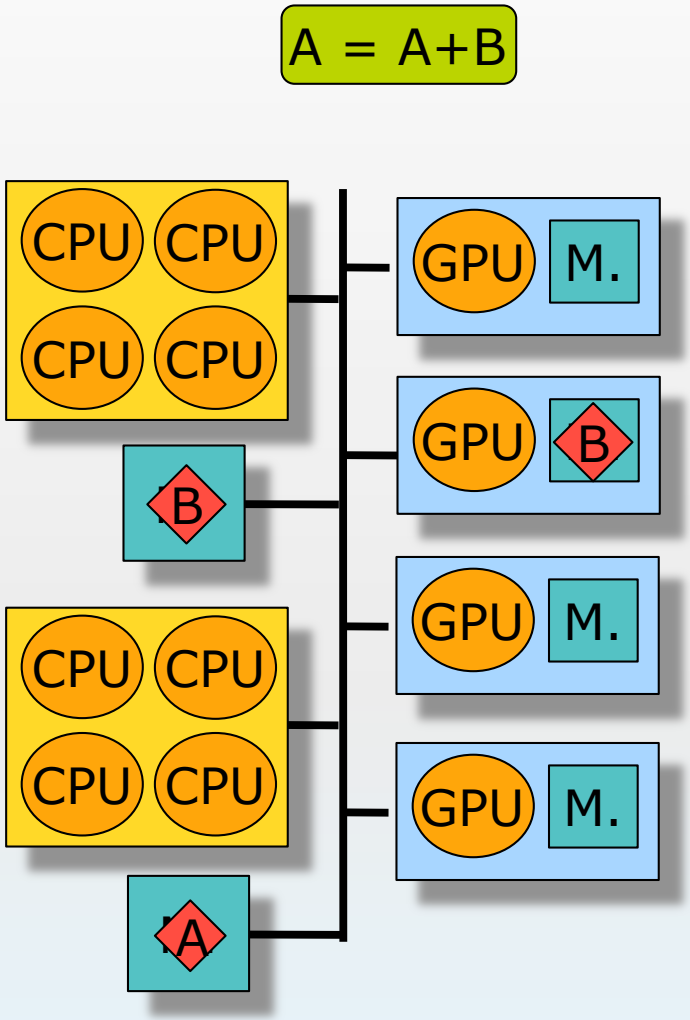
We need new runtime systems!
Leveraging CUDA/OpenCL

- Kernels need to exploit GPUs AND CPUs simultaneously
- Kernels need to run simultaneously
- Kernels need to accommodate to a variable number of processing units

Overview of StarPU

A runtime system for heterogeneous architectures

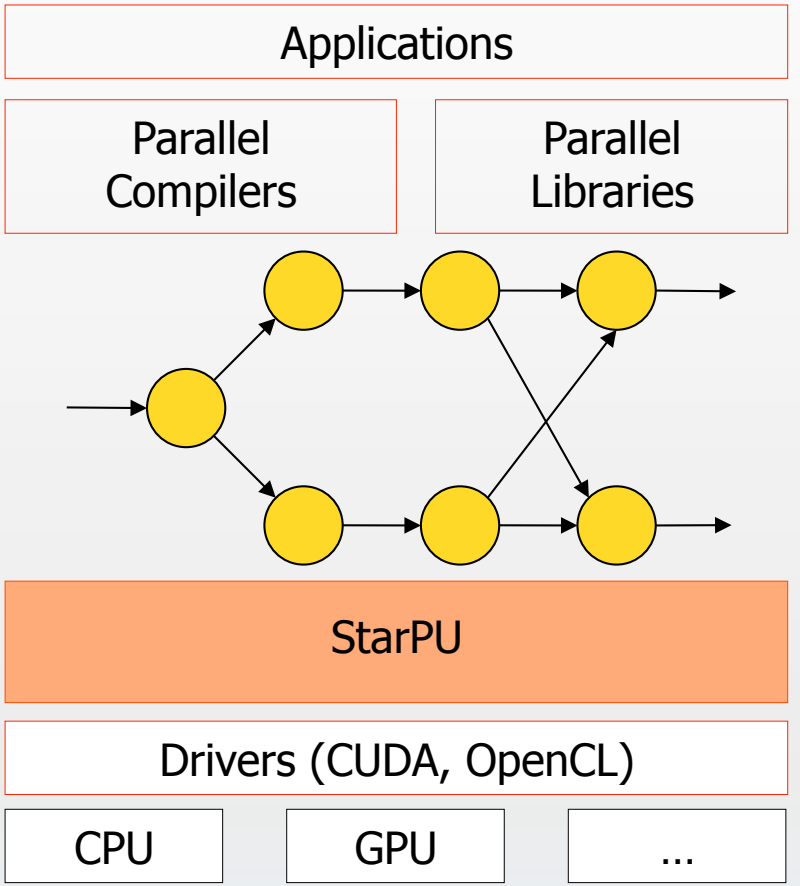
- Rational
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous processing units
  - Avoid unnecessary data transfers between accelerators
    - Software VSM for heterogeneous machines



Overview of StarPU

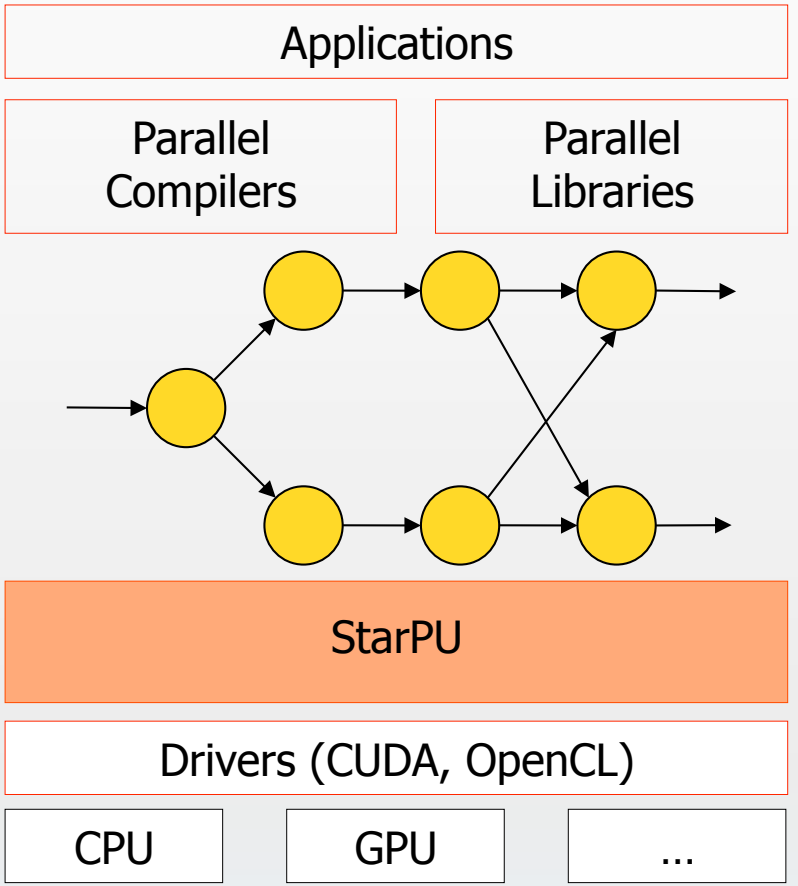
Maximizing PU occupancy, minimizing data transfers

- 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

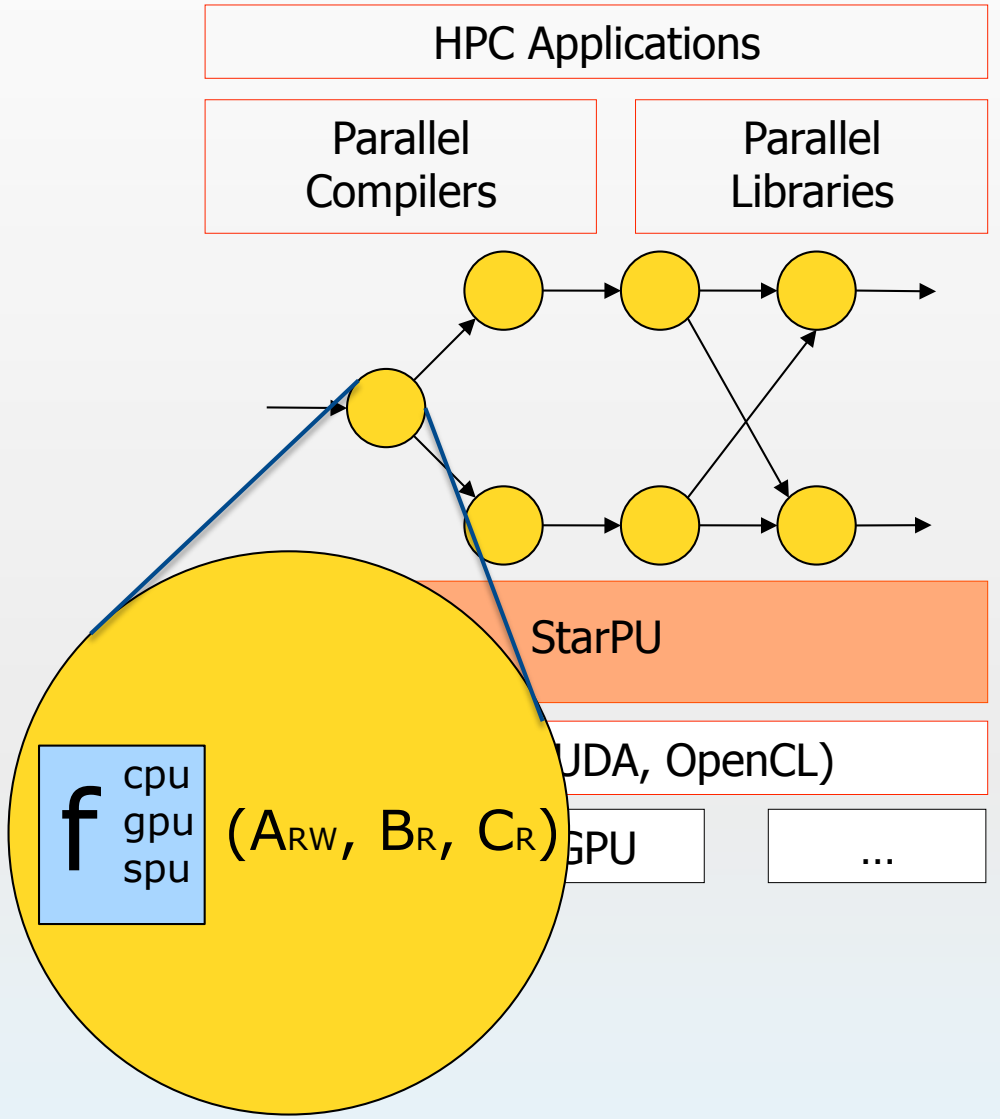
- StarPU provides a Virtual Shared Memory subsystem
  - Weak consistency
    - Explicit data fetch
  - Replication
    - MSI protocol
  - Single writer
    - Except for specific, "accumulation data"
  - High-level API
    - Partitioning filters
- Input & output of tasks = reference to VSM data



Tasks scheduling

Dealing with heterogeneous hardware accelerators

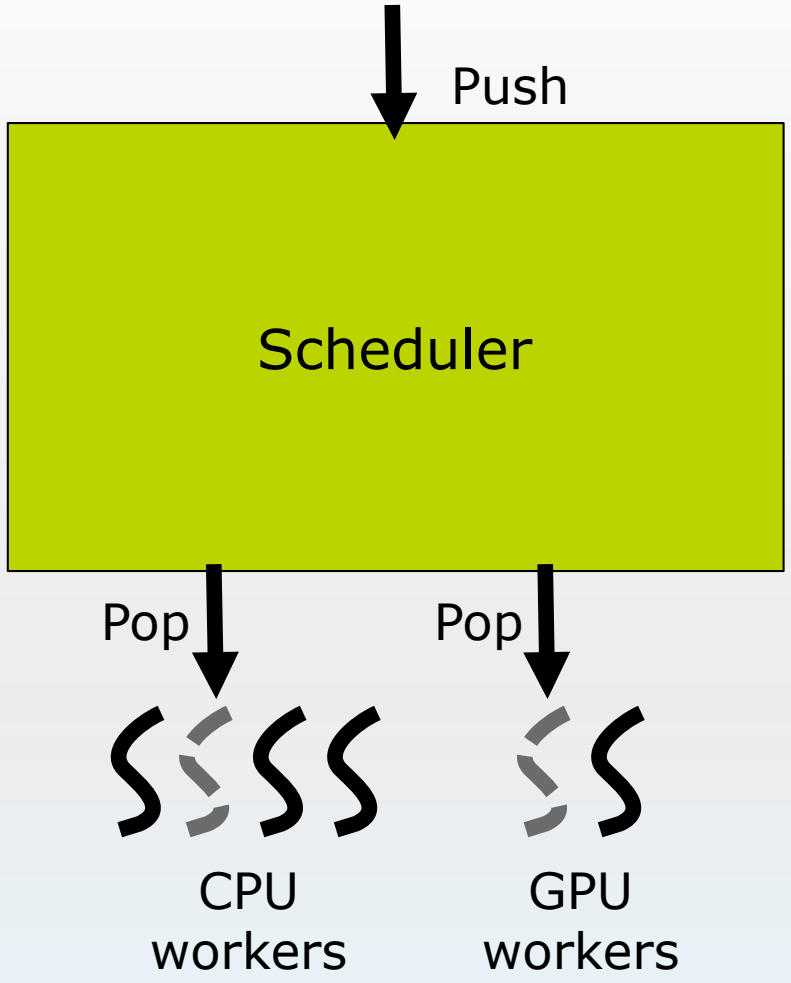
- Tasks =
  - Data input & output
  - Dependencies with other tasks
  - Multiple implementations
    - E.g. CUDA + CPU implementation
  - Scheduling hints
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



Tasks scheduling

How does it work?

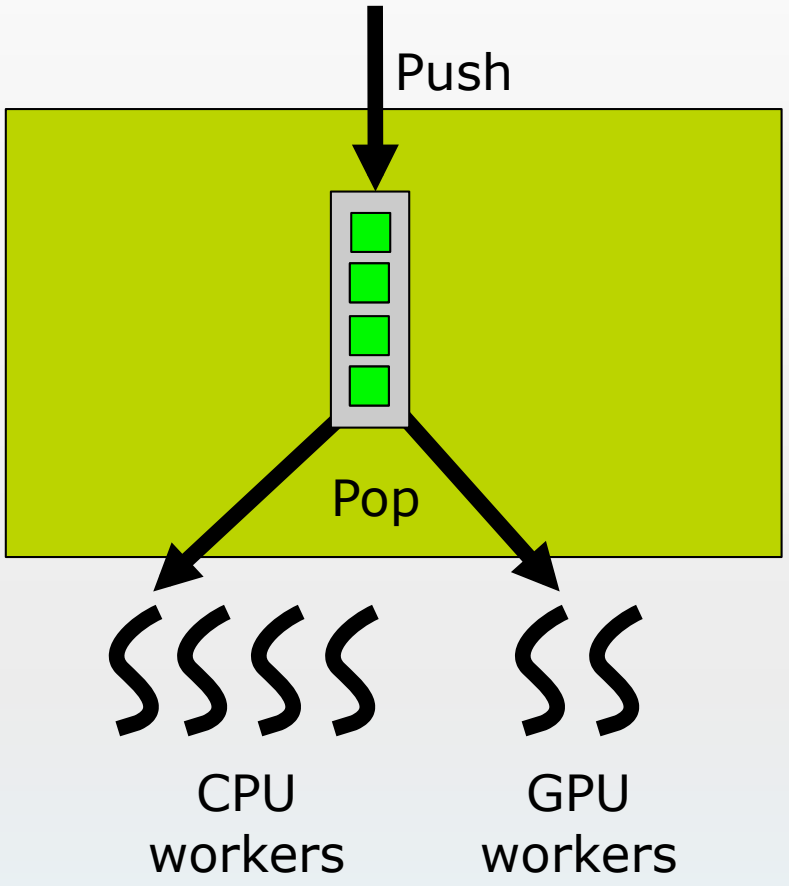
- When a task is submitted, it first goes into a pool of "frozen tasks" until all dependencies are met
- Then, the task is "pushed" to the scheduler
- Idle processing units actively poll for work ("pop")
- What happens inside the scheduler is… up to you!



Tasks scheduling

Developing your own scheduler

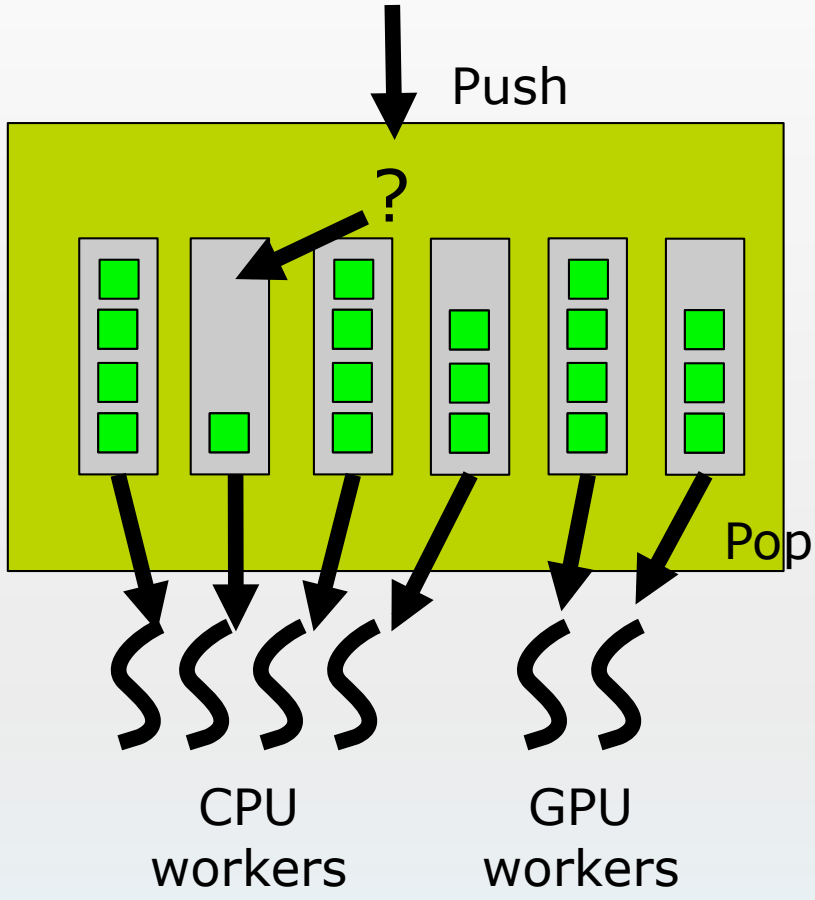
- Queue based scheduler
  - Each worker « pops » task in a specific queue
- Implementing a strategy
  - Easy!
  - Select queue topology
  - Implement « pop » and « push »
    - Priority tasks
    - Work stealing
    - Performance models, …
- Scheduling algorithms testbed



Tasks scheduling

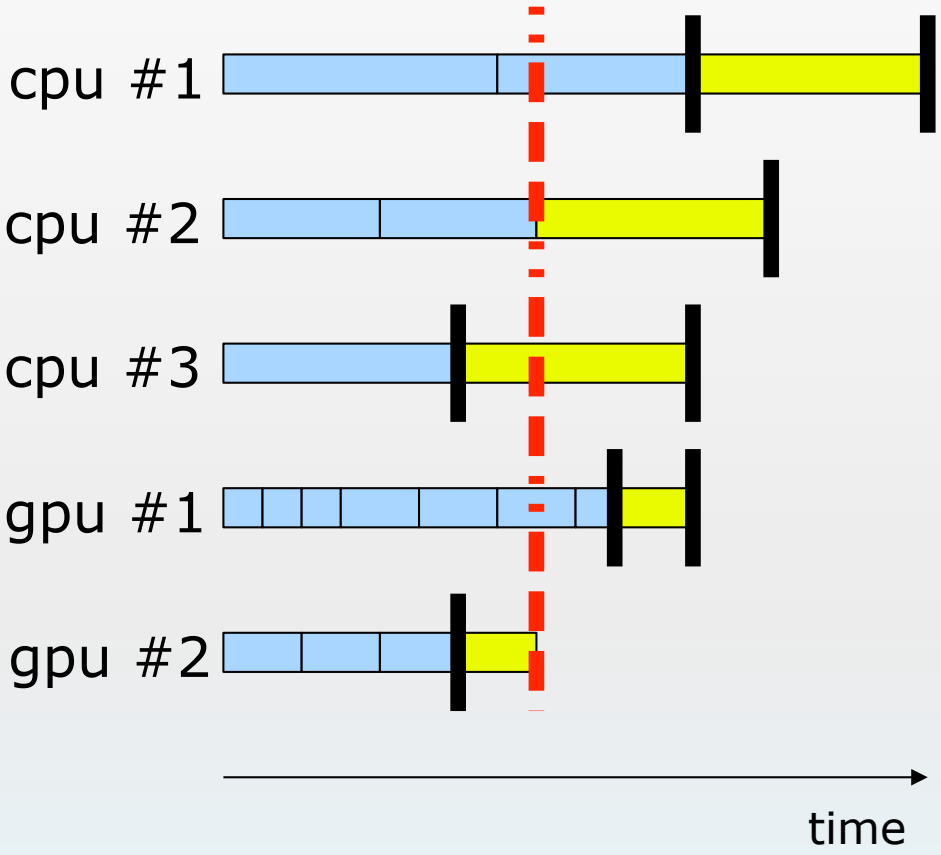
Developing your own scheduler

- 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



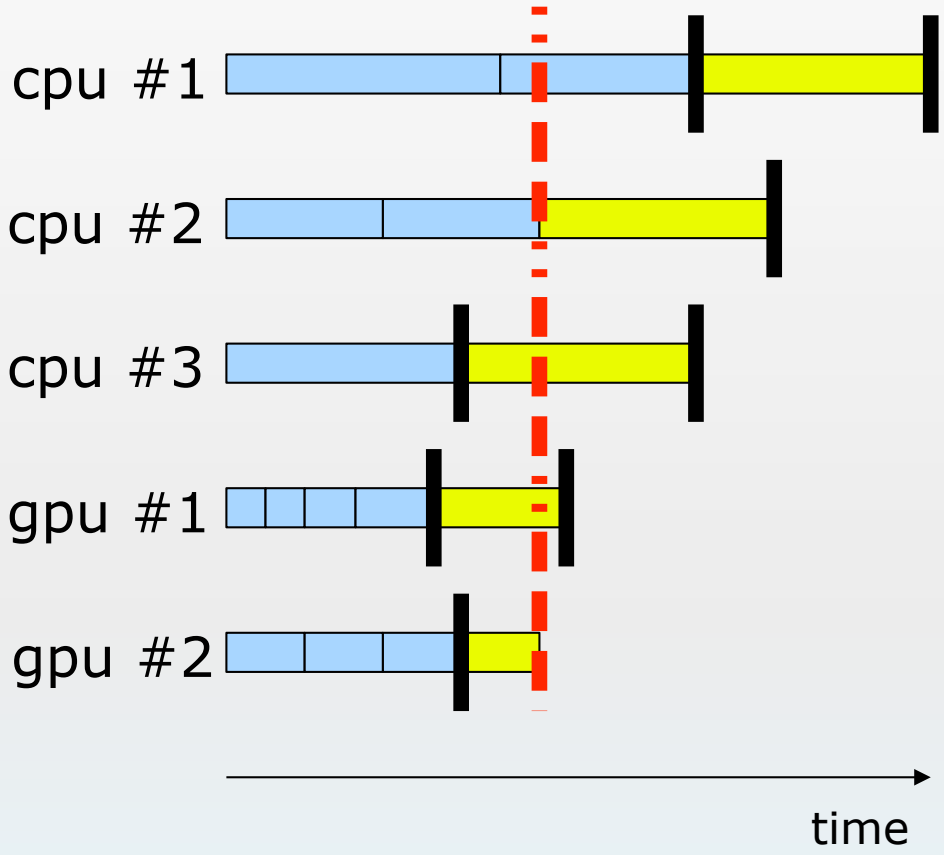
Dealing with heterogeneous architectures
Performance prediction

- 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



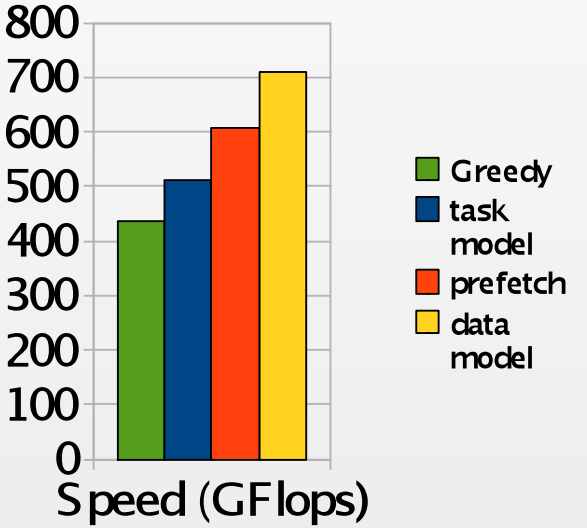
Dealing with heterogeneous architectures
Performance prediction

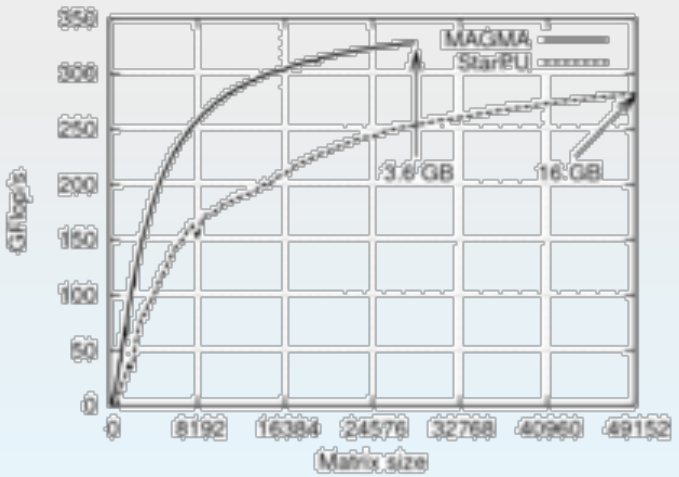
- Data transfer time estimation
  - Sampling based on off-line calibration
- Can be used to
  - Better estimate overall exec time
  - Minimize data movements



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

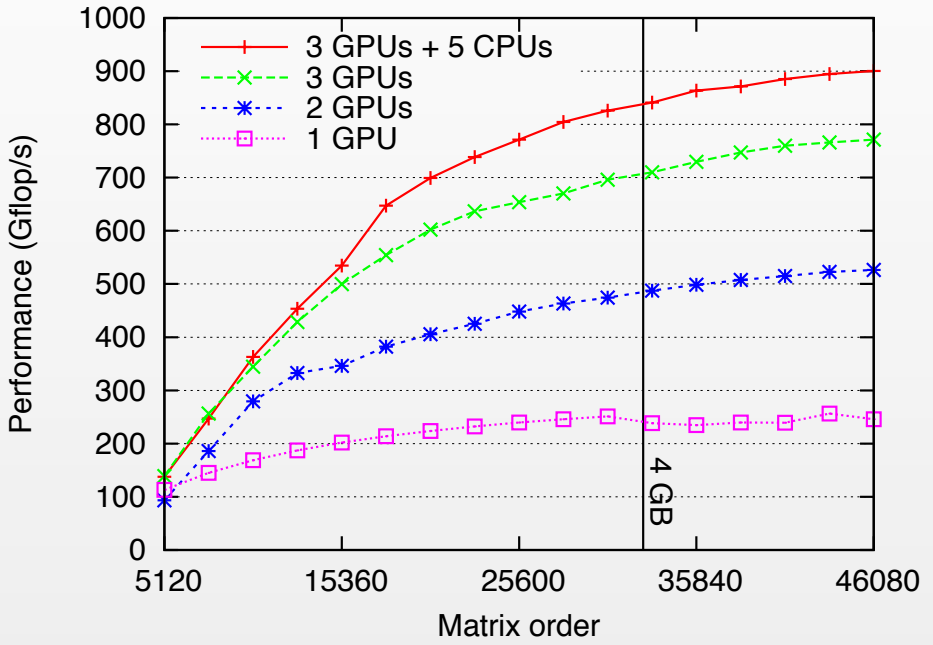


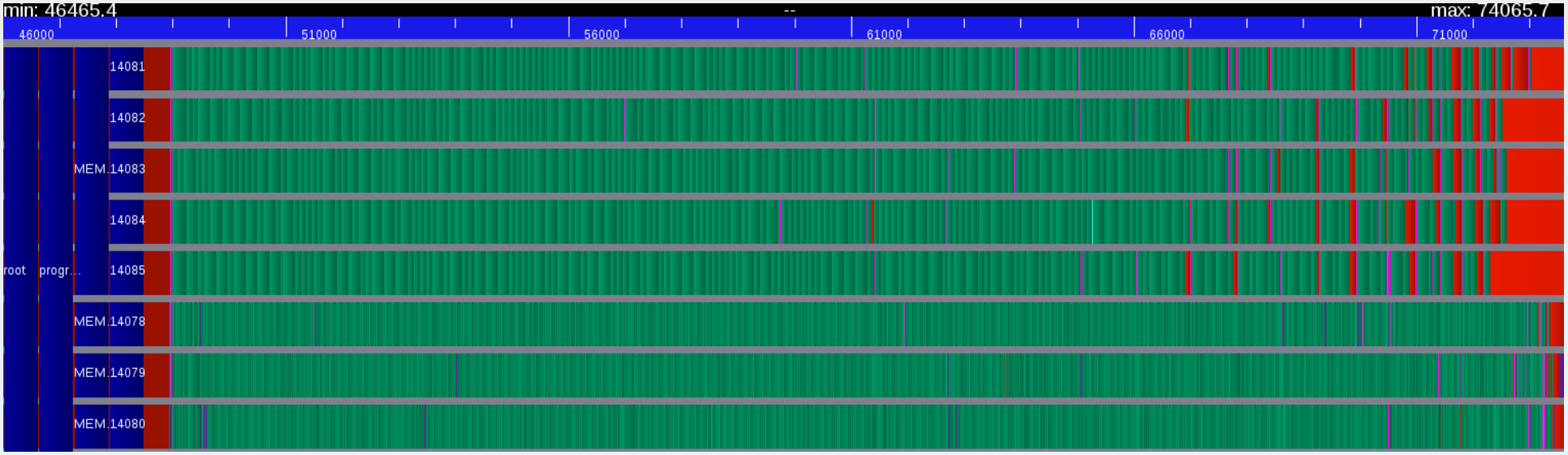


Dealing with heterogeneous architectures

Implementing MAGMA on top of StarPU

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

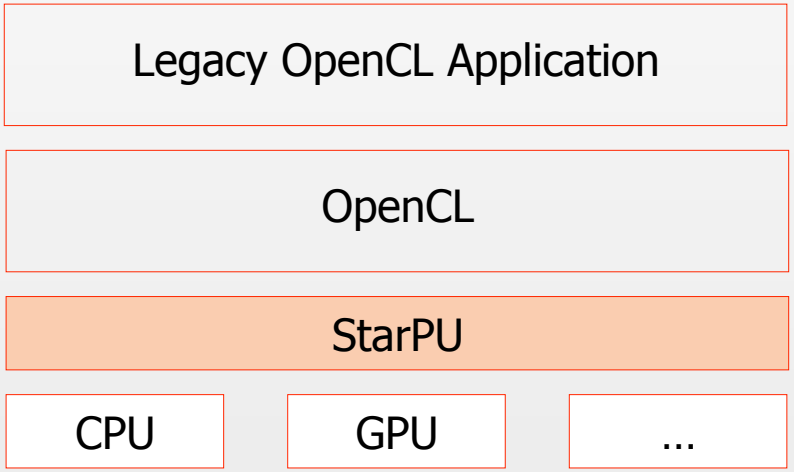




Using StarPU through a standard API

A StarPU driver for OpenCL

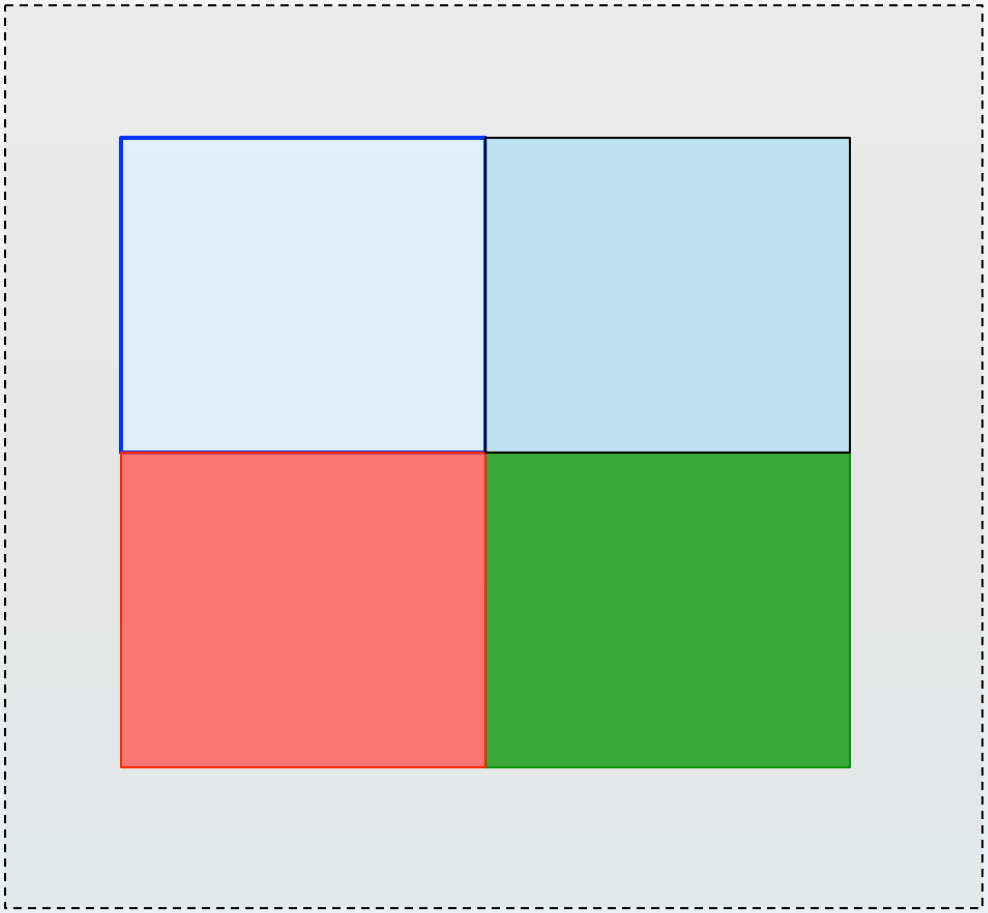
- Run legacy OpenCL codes on top of StarPU
  - OpenCL sees a number of starPU devices
- Performance limitations
  - Data transfers performed just-in-time
  - Data replication not managed by StarPU
- Ongoing work
  - We propose light extensions to OpenCL
    - Greatly improves flexibility when used
    - Regular OpenCL behavior if not extension is used



Integration with Multithreading

Dealing with parallel StarPU tasks

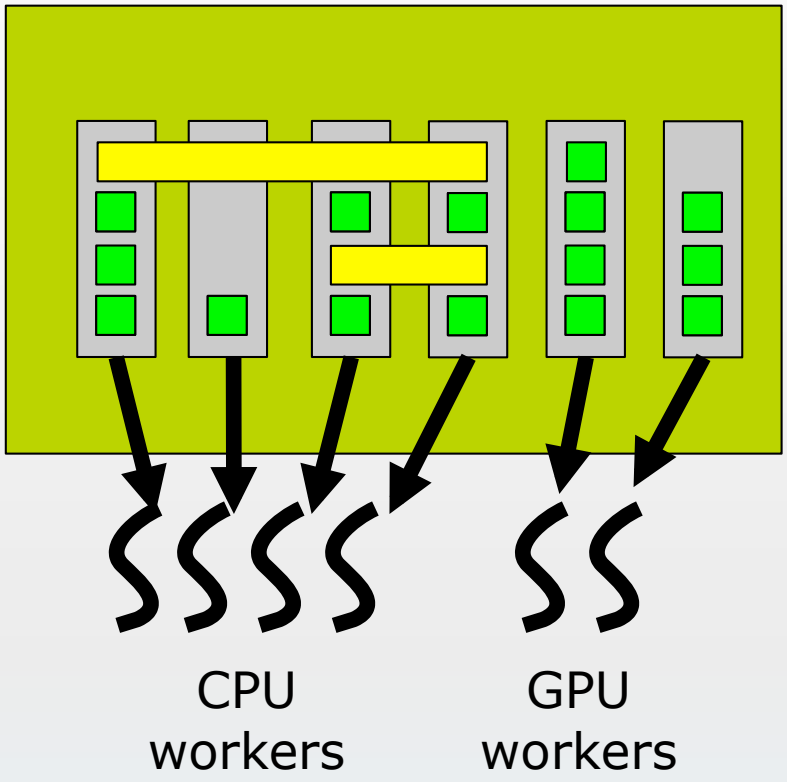
- StarPU + OpenMP/TBB/…
  - Many algorithms can take advantage of shared memory
  - We can't seriously "*taskify*" the world!
- The Stencil case
  - When neighbor tasks can be scheduled on a single node
    - Just use shared memory!
    - Hence an OpenMP stencil kernel



Integration with and Multithreading

Dealing with parallel StarPU tasks

- Current 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

- 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

- Propose “natural” extensions to OpenCL
  - Introduce more dynamicity
- Enhance cooperation between runtime systems and compilers
  - Granularity, runtime support for “divisible tasks”
  - Feedback for autotuning software
    - [PEPPHER European project]
- Demonstrate the relevance of StarPU in other frameworks
  - StarPU+OpenMP+MPI as a target for XcalableMP
    - French-Japanese ANR-JST FP3C project

Thank you!

- More information about StarPU

  http://runtime.bordeaux.inria.fr