Programming heterogeneous, accelerator-based multicore machines: a runtime's perspective

Raymond Namyst
University of Bordeaux

ORAP
March 25th, 2013

RUNTIME
INRIA Group
INRIA Bordeaux Sud-Ouest

The RUNTIME Team

High Performance Runtime Systems for Parallel Architectures

- Mid-size research group
  - 10 permanent researchers
  - 5 engineers
  - 8 PhD students

- Part of
  - INRIA Bordeaux – Sud-Ouest Research Center
  - LaBRI, Computer Science Lab at University of Bordeaux 1

Understanding the evolution of parallel machines

The end of Moore's law?

- The end of single thread performance increase
  - Clock rate is no longer increasing
  - Thermal dissipation

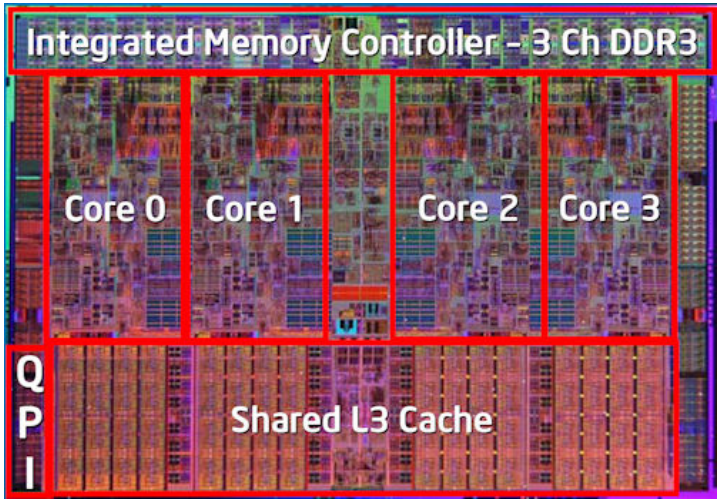
- Processor architecture is already very sophisticated
  - Prediction and Prefetching techniques achieve a very high percentage of success
  - Actually, processor complexity is decreasing!

- Question: What kind of circuits should we better add on a chip?

Understanding the evolution of parallel machines

Welcome to the multicore era

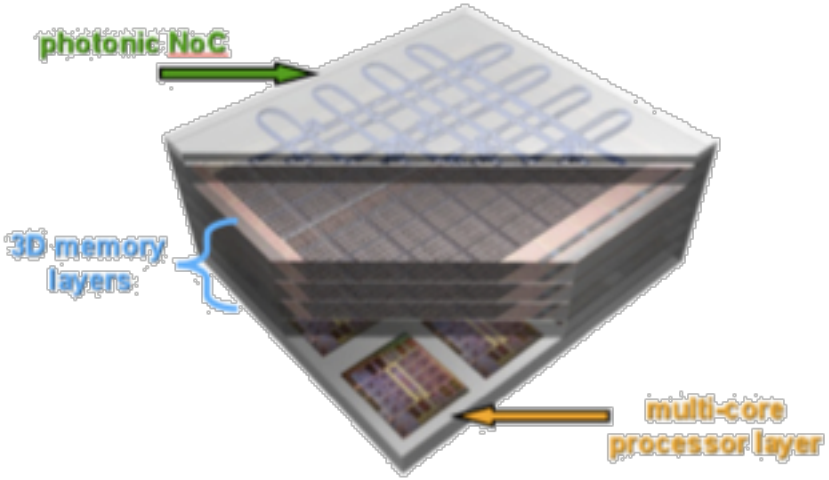
- Answer: Multicore chips
  - Several cores instead of one processor
- Back to complex memory hierarchies
  - Shared caches
    - Organization is vendor-dependent
  - NUMA penalties
- Clusters can no longer be considered as “flat sets of processors”

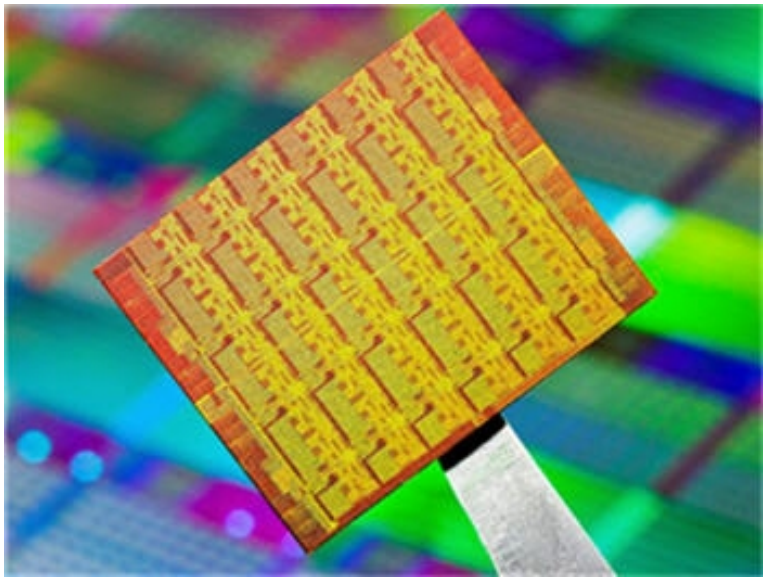


Understanding the evolution of parallel machines

Multicore is a solid trend

- More performance = more cores
  - Toward embarrassingly parallel machines?
- Designing scalable multicore architectures
  - 3D stacked memory
  - Non-coherent cache architectures
    - Intel SCC
    - IBM Cell/BE





What Programming Models for such machines?

Widely used, standard programming models

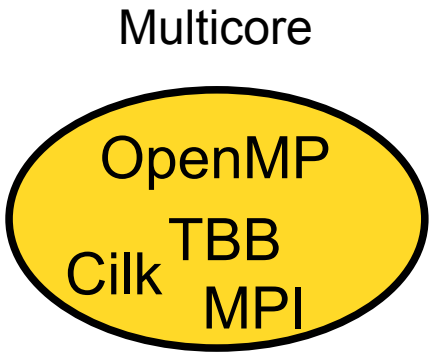
- MPI
  - Communication Interface
  - Scalable implementations exist already
    - Was actually designed with scalability in mind
    - Makes programmers “think” scalable algorithms
  - NUMA awareness?
  - Memory consumption

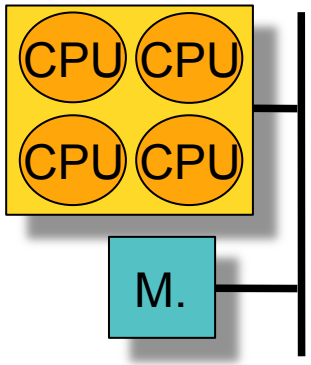
- OpenMP
  - Directive-based, incremental parallelization
  - Shared-memory model
    - Well suited to symmetric machines
  - Portability wrt #cores
  - NUMA awareness?

The Quest for Programming Models

Dealing with multicore machines

- Several efforts aim at making MPI and OpenMP multicore-ready
  - MPI
    - NUMA-aware buffer management
    - Efficient collective operations
  - OpenMP
    - Scheduling in a NUMA context (memory affinity, work stealing)
    - Memory management (page migration)

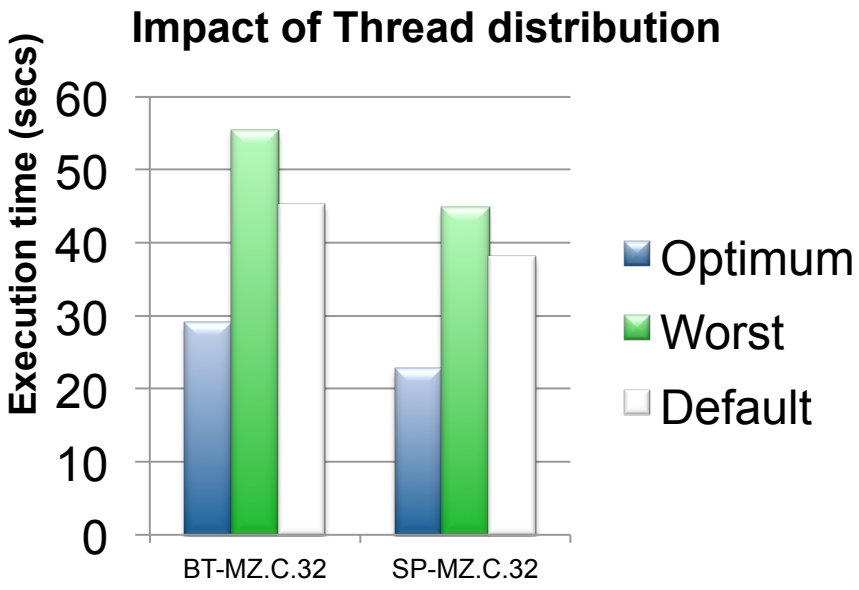


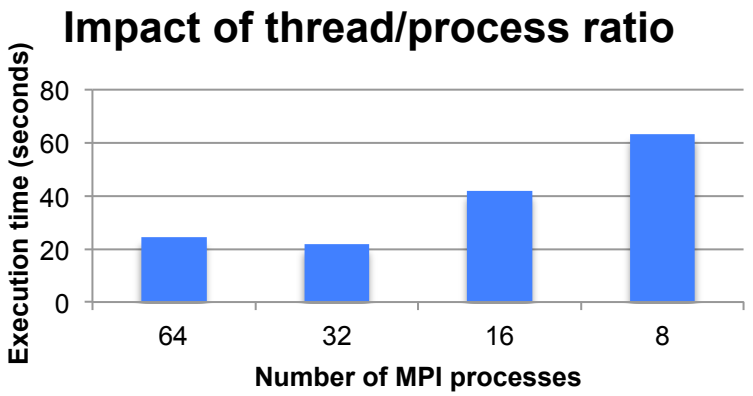


Mixing OpenMP with MPI

It makes sense even on shared-memory machines

- MPI address spaces must fit the underlying topology
- Experimental platforms for tuning hybrid applications
  - Topology-aware process allocation
  - Customizable core/process ratio
  - # of OpenMP tasks independent from # of cores
- MPC software (CEA/DAM)

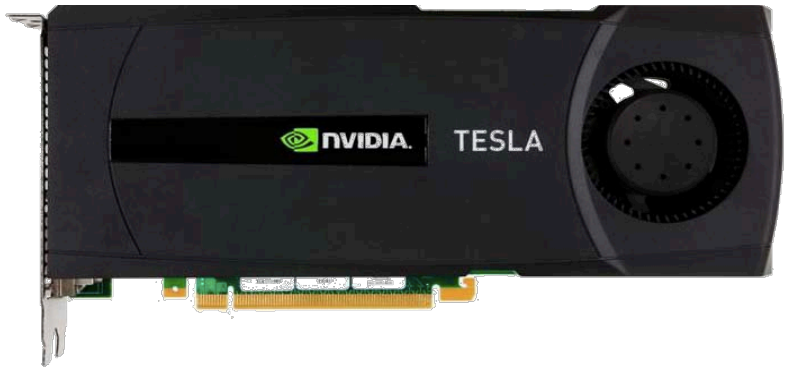




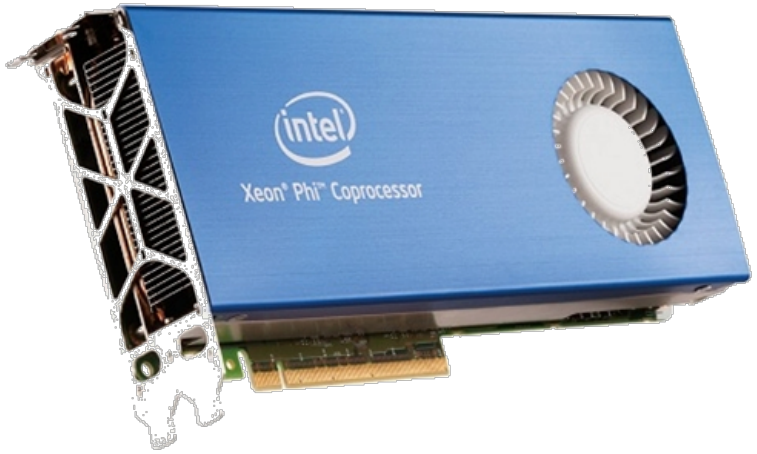
Then came heterogeneous computing

And it seems to be a solid trend…

- Accelerators
  - Nvidia & AMD GPUs
    - De facto adoption
    - Concrete success stories
      Speedups > 50
  - Intel MIC
  - ARM Mali



- Accelerators get more integrated
  - Intel Ivy Bridge, Haswell
  - AMD APUs



- They all raise the same issues
  - Cost of moving data
    - Even for so-called APUs
  - Programming model

The Quest for Programming Models

How shall we program hybrid machines?

- Use well-known libraries when available
  - BLAS routines, FFT kernels, stencils
- Use directive-based languages
  - OpenACC, HMPP, OpenMP 4.0
  - They can save you from:
    - Learning which type of memory coalescing is supported by a given card
    - Thinking about setting some arrays dimensions to 17 instead of 16
    - Searching for tradeoffs between cluster occupancy and register usage
- Otherwise, fall back to CUDA or OpenCL…

The Quest for Programming Models

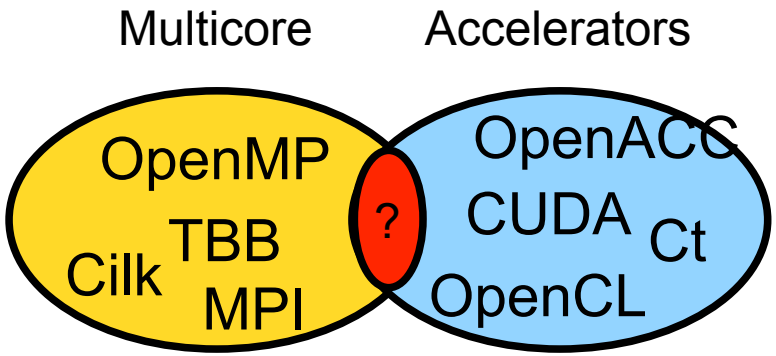
How shall we program hybrid machines?

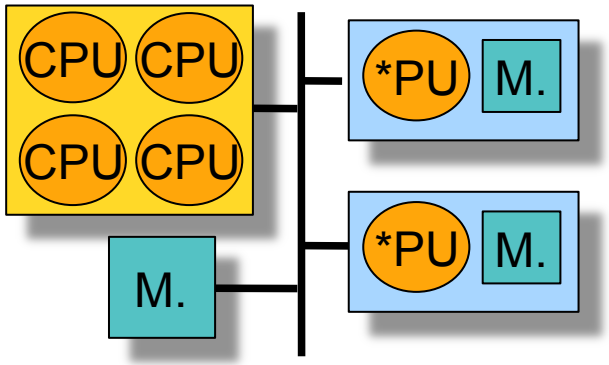
- Use well-known libraries when available
  - BLAS routines, FFT kernels, stencils
- Use directive-based languages
  - OpenACC, HMPP, OpenMP 4.0
  - They can save you from:
    - Learning what kind of memory coalescing is supported by a given card
    - Thinking about setting some arrays dimensions to 17 instead of 16
    - Searching for tradeoffs between cluster occupancy and register usage
- Otherwise, ~~fall back to CUDA or OpenCL~~ think twice and give directive-based languages a second chance!

The Quest for Programming Models

How shall we program heterogeneous clusters?

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

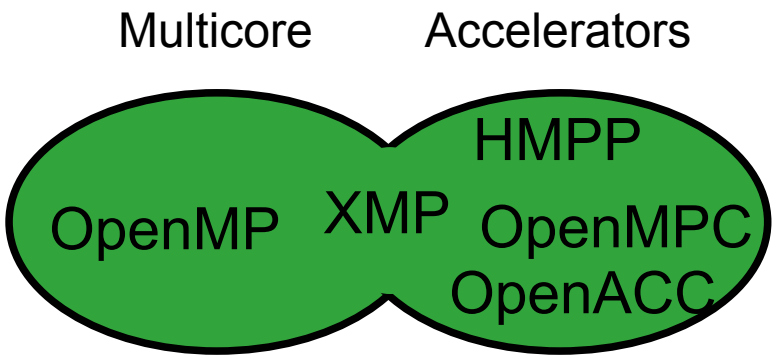


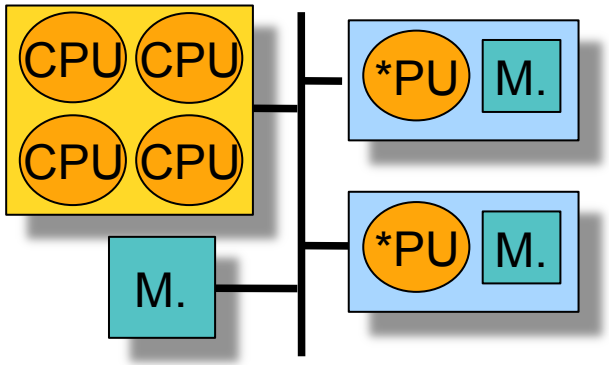


The Quest for Programming Models

How shall we program heterogeneous clusters?

- The uniform way
  - Use high-level programming languages to deal with network + multicore + accelerators
  - Increasing number of directive-based languages
    - Use simple directives… and good compilers!
    - XcalableMP
      ANR-JST FP3C project
      PGAS approach
    - HMPP, OpenACC, OpenMP 4.0
  - Much better potential for *composability…*
    - If compiler is clever!





The role of runtime systems

Toward “portability of performance”

- Do dynamically what can’t be done statically
  - Load balance
  - React to hardware feedback
  - Autotuning, self-organization
- We need to put more intelligence into the runtime!

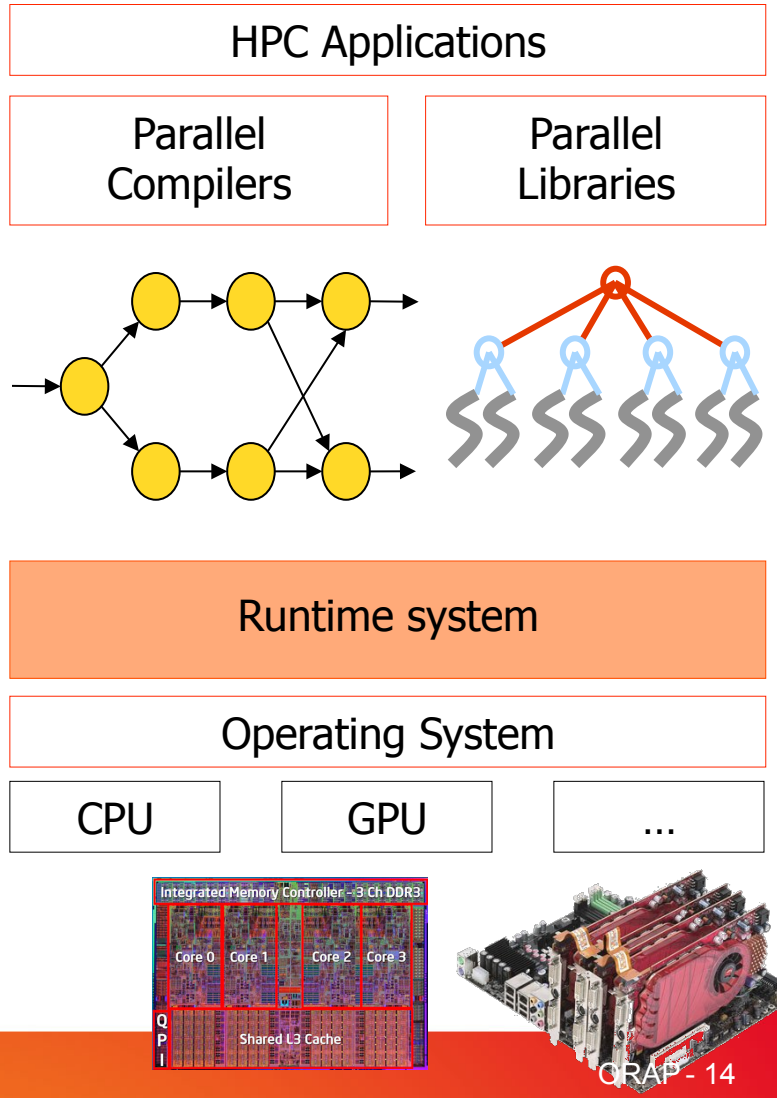
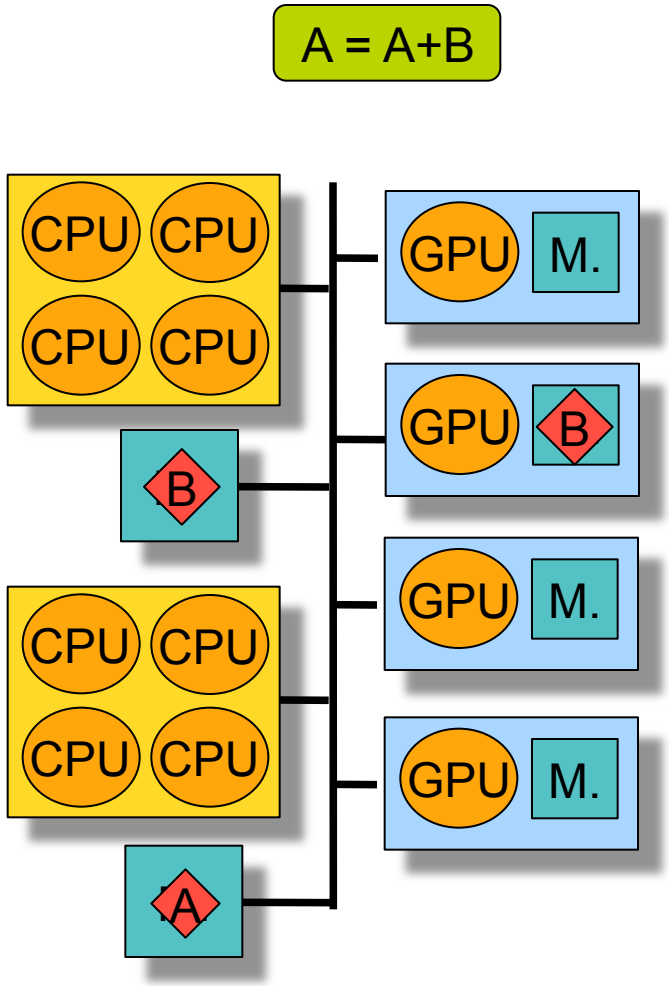


Illustration of a widespread approach: StarPU

A runtime system for heterogeneous architectures

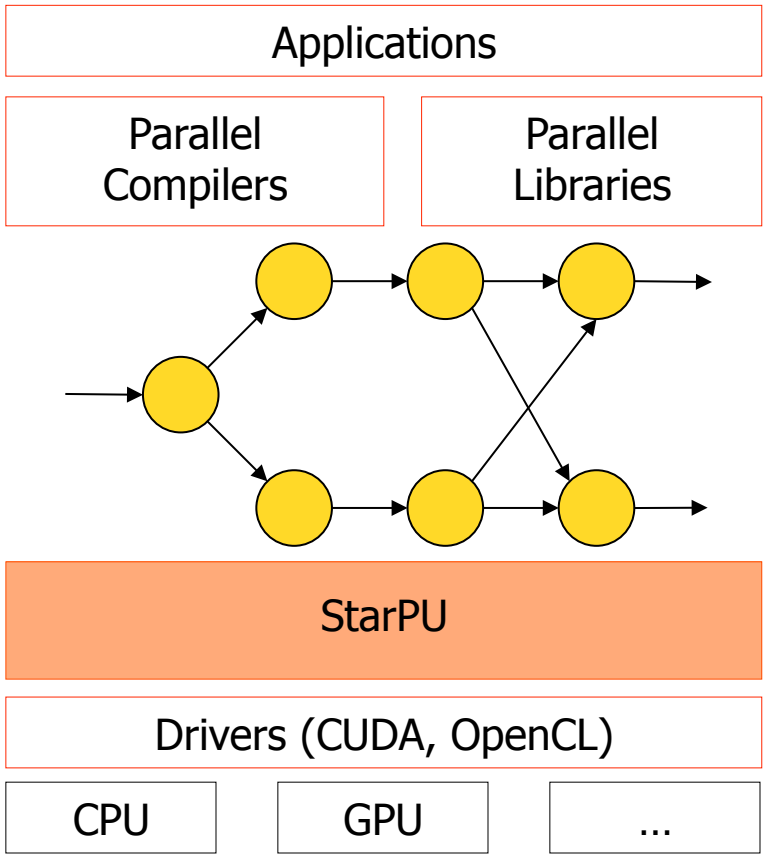
- Rational
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous processing units
  - Avoid unnecessary data transfers between accelerators
    - Software VSM for heterogeneous machines



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
      - Data-flow approach
  - Provide a high-level data management layer
    - Application should only describe
      - which data may be accessed by tasks
      - How data may be divided

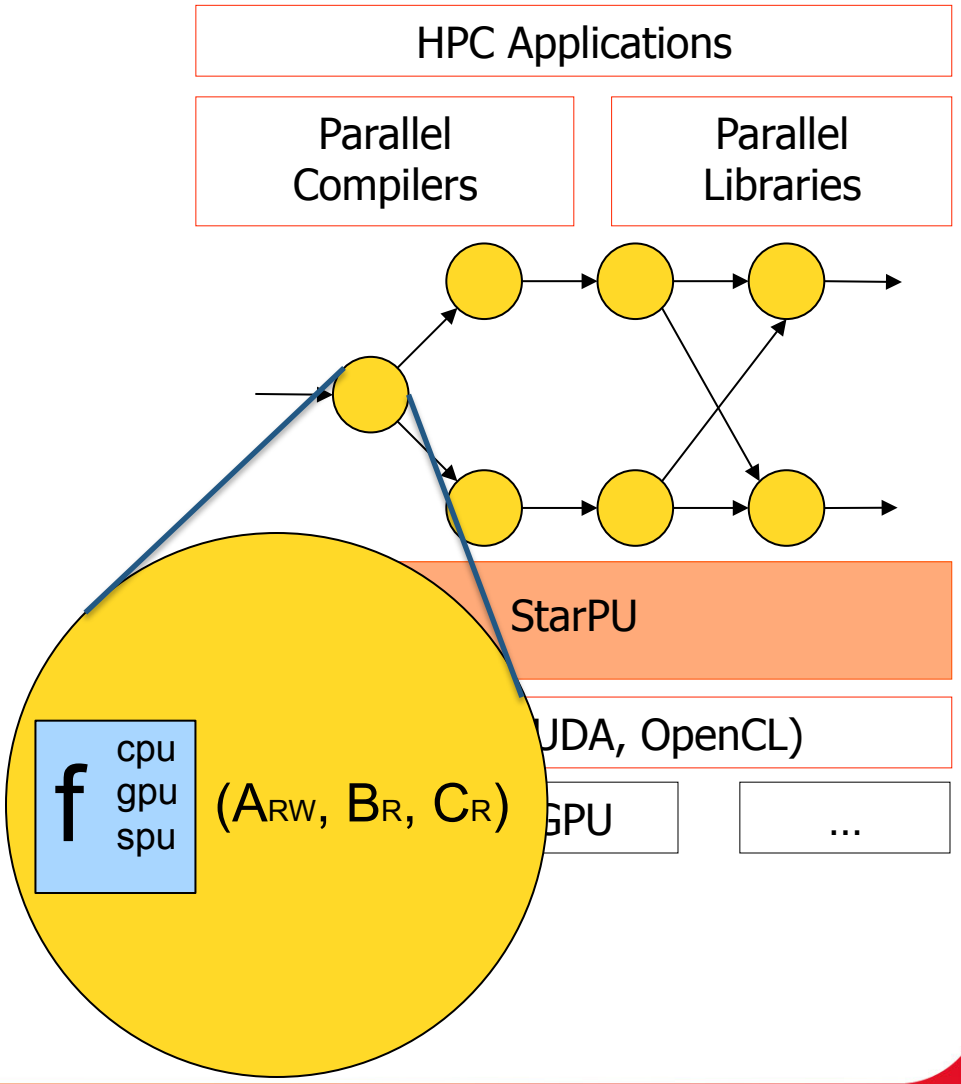


Overview of StarPU

Dealing with heterogeneous hardware accelerators

- Tasks =
  - Data input & output
  - Dependencies with other tasks
  - Multiple implementations
    - E.g. CUDA + CPU implementation
  - Scheduling hints

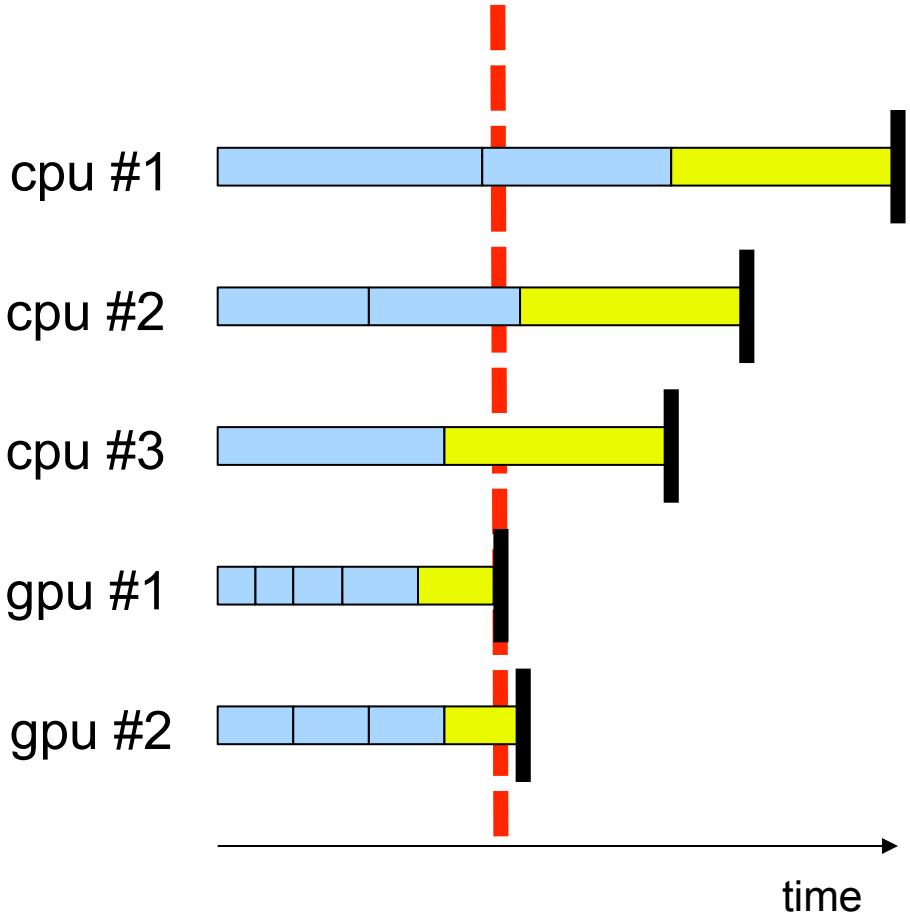
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



Dealing with heterogeneous architectures

Performance prediction

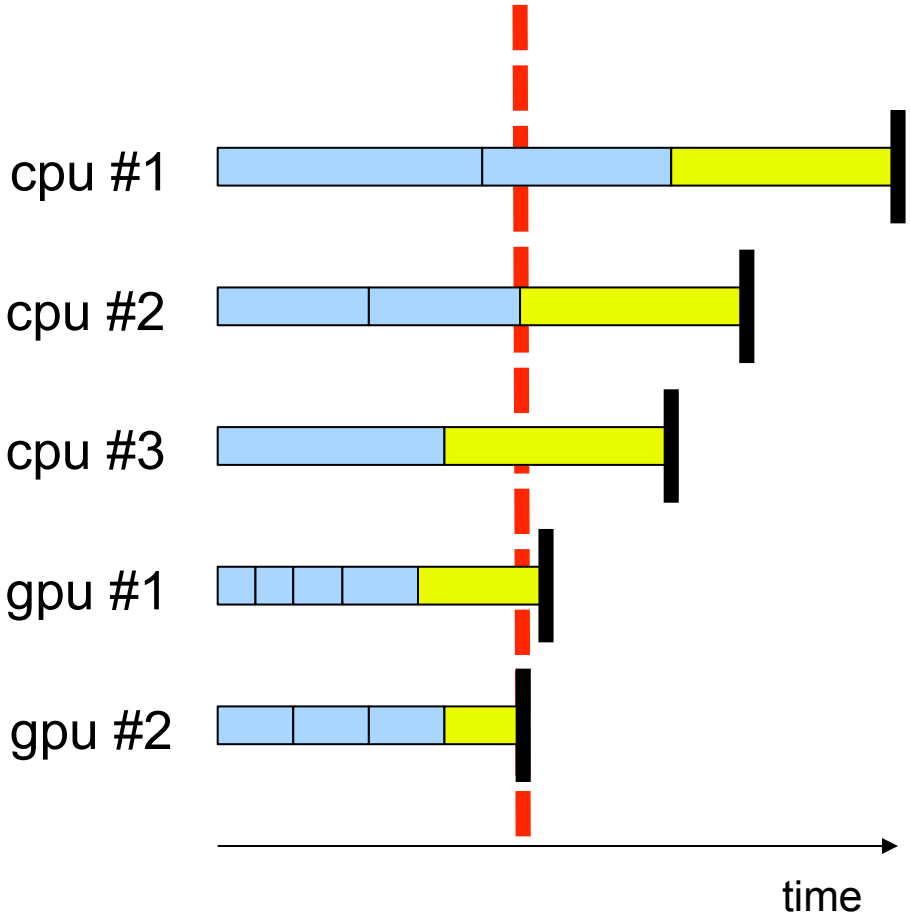
- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
- Can be used to improve scheduling
  - E.g. Heterogeneous Earliest Finish Time



Dealing with heterogeneous architectures

Performance prediction

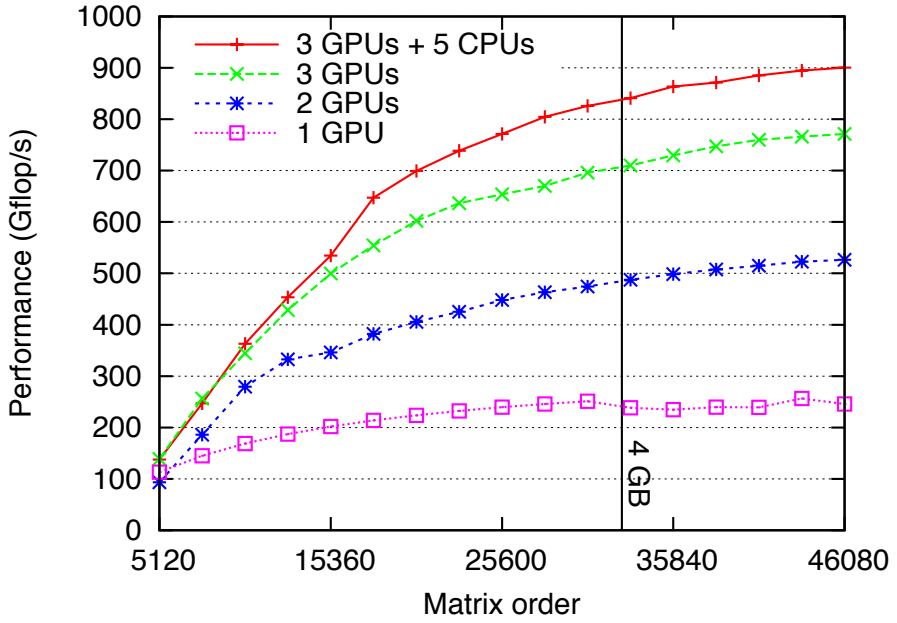
- Data transfer time estimation
  - Sampling based on off-line calibration
- Can be used to
  - Better estimate overall exec time
  - Minimize data movements

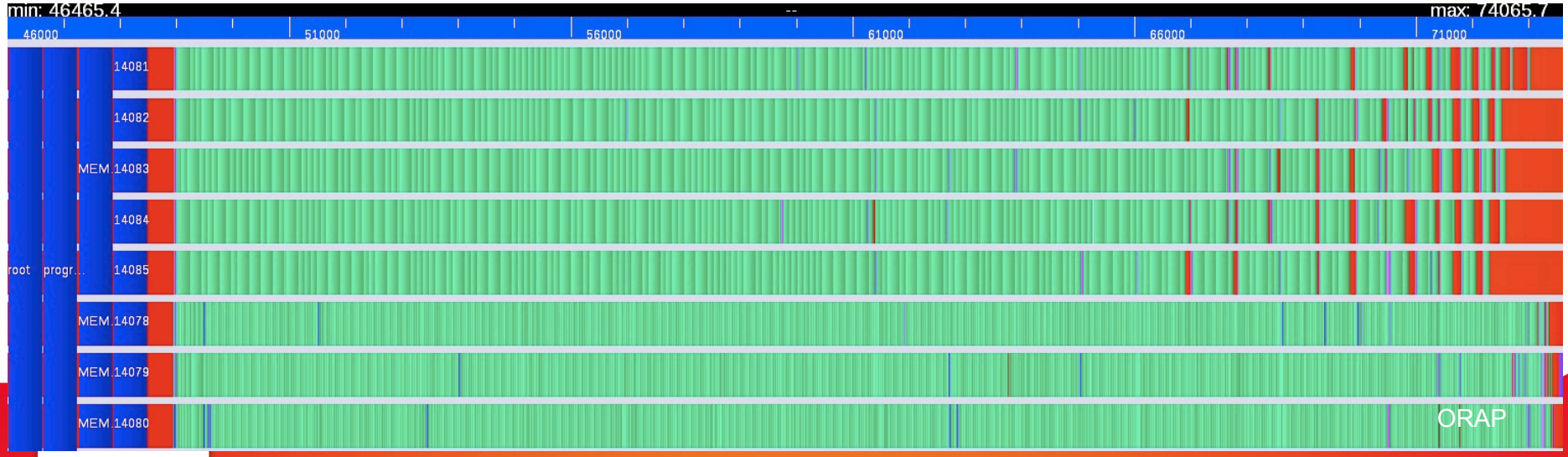


Mixing PLASMA and MAGMA with StarPU

With University of Tennessee & INRIA HiePACS

- Cholesky decomposition
  - 5 CPUs (Nehalem) + 3 GPUs (FX5800)
  - Efficiency > 100%

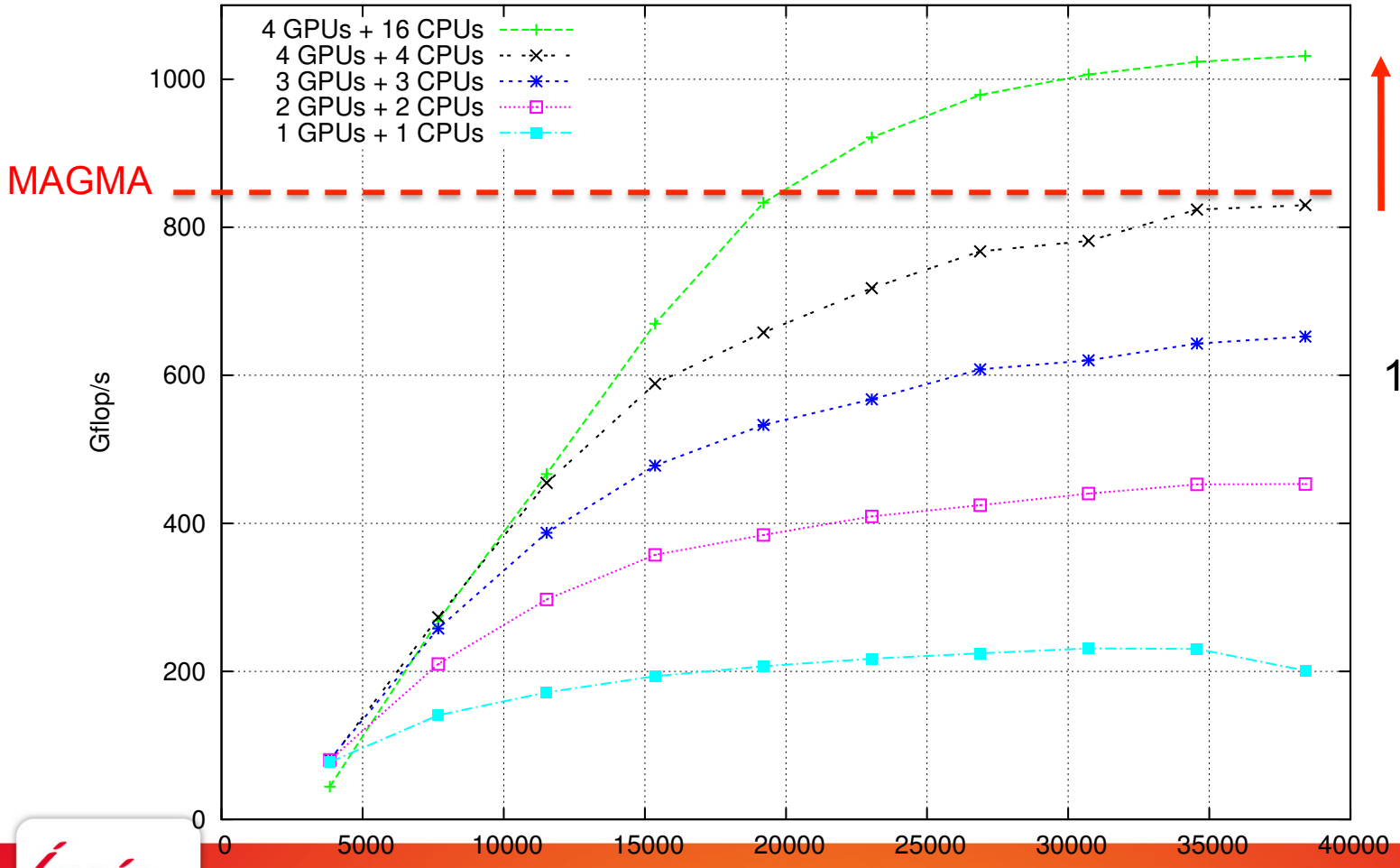




Mixing PLASMA and MAGMA with StarPU

With University of Tennessee & INRIA HiePACS

- QR decomposition
  - 16 CPUs (AMD) + 4 GPUs (C1060)



+12 CPUs
~200 GFlops

(although
12 CPUs alone
~150 Gflops)

Mixing PLASMA and MAGMA with StarPU

« Super-Linear » efficiency in QR?

- Kernel efficiency
  - sgeqrt
    - CPU: 9 Gflops GPU: 30 Gflops Ratio: x3
  - stsqrt
    - CPU: 12 Gflops GPU: 37 Gflops Ratio: x3
  - somqr
    - CPU: 8.5 Gflops GPU: 227 GflopsRatio: x27
  - Sssmqr
    - CPU: 10 Gflops GPU: 285 GflopsRatio: x28

- Task distribution observed on StarPU
  - sgeqrt: 20% of tasks on GPUs
  - Sssmqr: 92.5% of tasks on GPUs

- Heterogeneous architectures are cool! ☺

Theoretical bound

Can we perform a lot better?

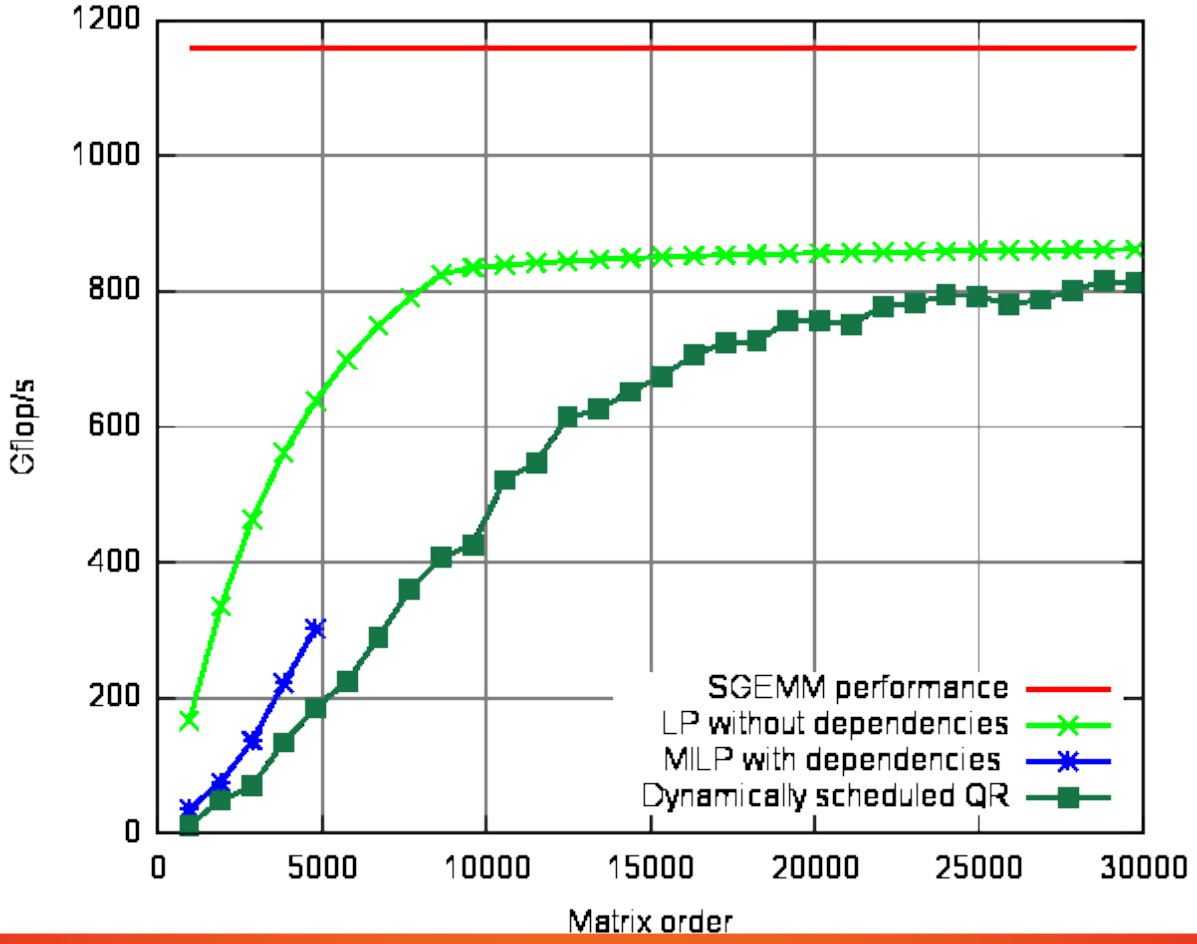
- Express set of tasks (and dependencies) as Linear problem
  - With heterogeneous task durations, and heterogeneous resources

$$\begin{cases} \text{minimize } tmax \\ \forall w \in W, \sum_{t \in T} n_{t,w} t_{t,w} \leq tmax \\ \forall t \in T, \sum_{w \in W} n_{t,w} = n_t. \end{cases}$$

Theoretical bound

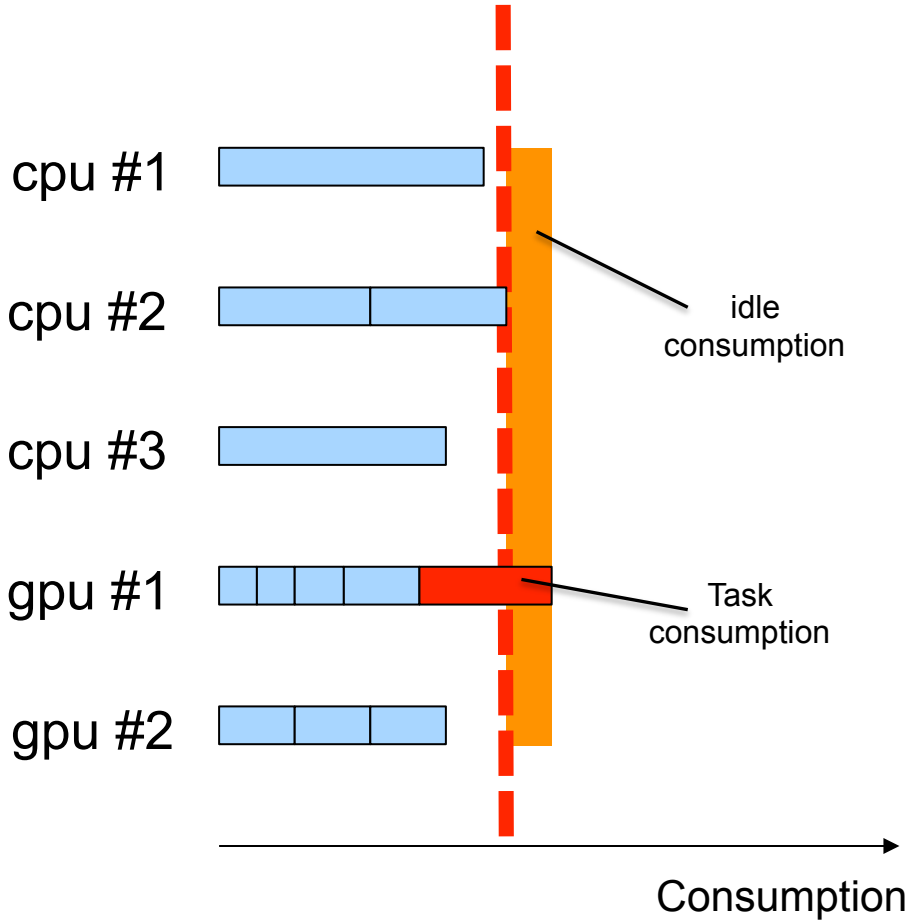
Can we perform a lot better?

- Express set of tasks (and dependencies) as Linear problem
  - With heterogeneous task durations, and heterogeneous resources



Scheduling for Power Consumption

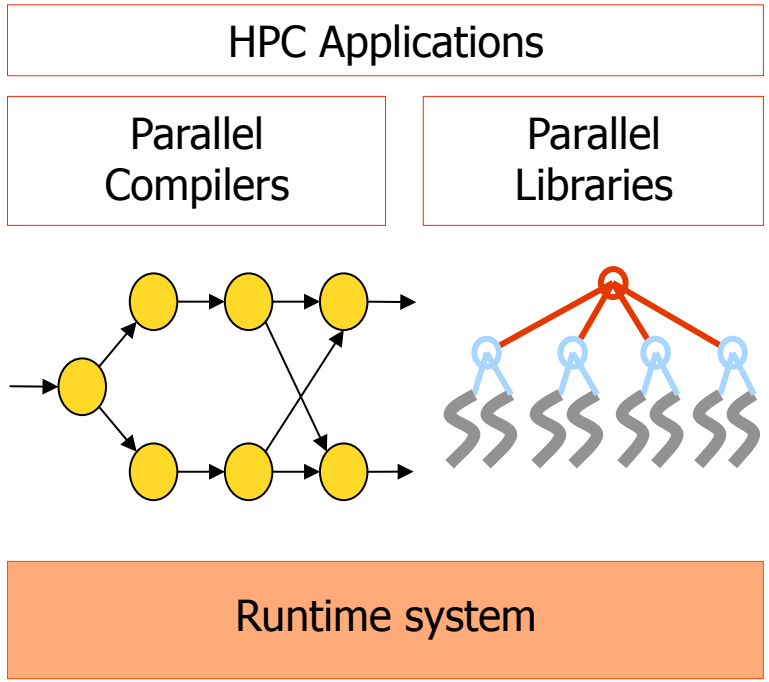
- Power consumption information can be retrieved from OpenCL devices
  - clGetEventProfilingInfo
    - CL_PROFILING_POWER_CONSUMED
  - Implemented by Movidius multicore accelerator
- Autotuning of power models similar to performance
- Scheduling heuristic inspired by MCT
  - Actually, a subtle mix with MCT is possible

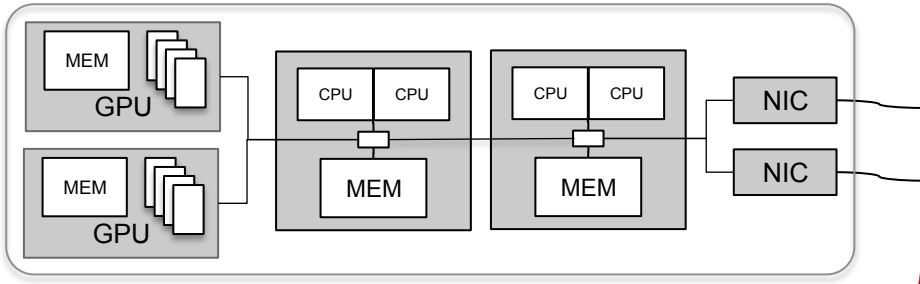


What did we learn?

Toward “portability of performance”

- Abstract Architecture
  - Do not hide nor expose everything!
- Extract knowledge from upper layers
  - Structure of parallelism
- Let the runtime system take control!

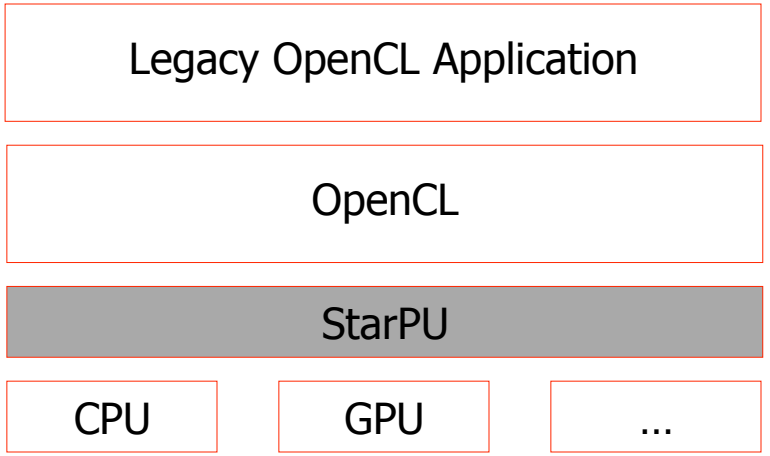




Bridging the gap with standards

SOCL: A StarPU-enabled OpenCL implementation

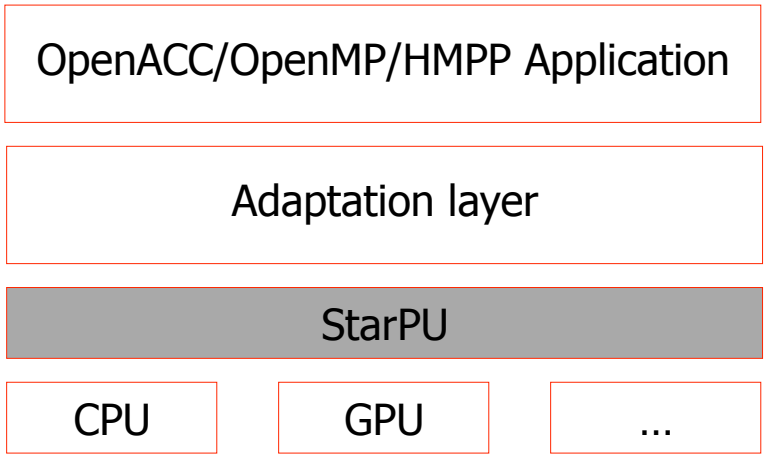
- Run OpenCL apps on top of StarPU
  - StarPU involved only if a slight modification is made:
    - OpenCL queue attached to a context, not to a particular device
  - Incremental introduction of “dynamic scheduling” within OpenCL applications
  - Much better performance if explicit data transfers are avoided



Bridging the gap with standards

Directive-based languages

- Directive-based languages on top of dynamic runtime systems
  - HMPP/StarPU on the way
  - OpenMP
    - Via KaStar Inria initiative
  - OpenACC



- Idea
  - Use a compiler to generate accelerator code
  - Use runtime system to achieve dynamic load balancing

```
// Get rid of such explicit data transfers!
//
#pragma acc region copy(A[0:N-1][0:M-1])
{
   for (int i=0; i<N;++i) { … }
}
```

Hot topics

And hard challenges

- All the magic done by the scheduler (scheduling, prefetching, flushing) makes it harder to understand (bad) performance
  - *Where does this disappointing behavior come from?*

- The quest for “good” granularity
  - Enabling adaptive granularity
    - Divisible tasks
    - Autotuning
  - Need for tight compiler/runtime system integration

- Integration into existing MPI programs
  - Because data is spread among multiple memory banks, we have to slightly change the way we designate data to be sent

Hot topics

Composing parallel libraries

- Integration with other shared-memory programming paradigms
  - We cannot *taskify* the whole world
  - At least provide support for parallel tasks
    - Implemented with threads/OpenMP/…
    - Nested parallelism
  - It raises the composability problem

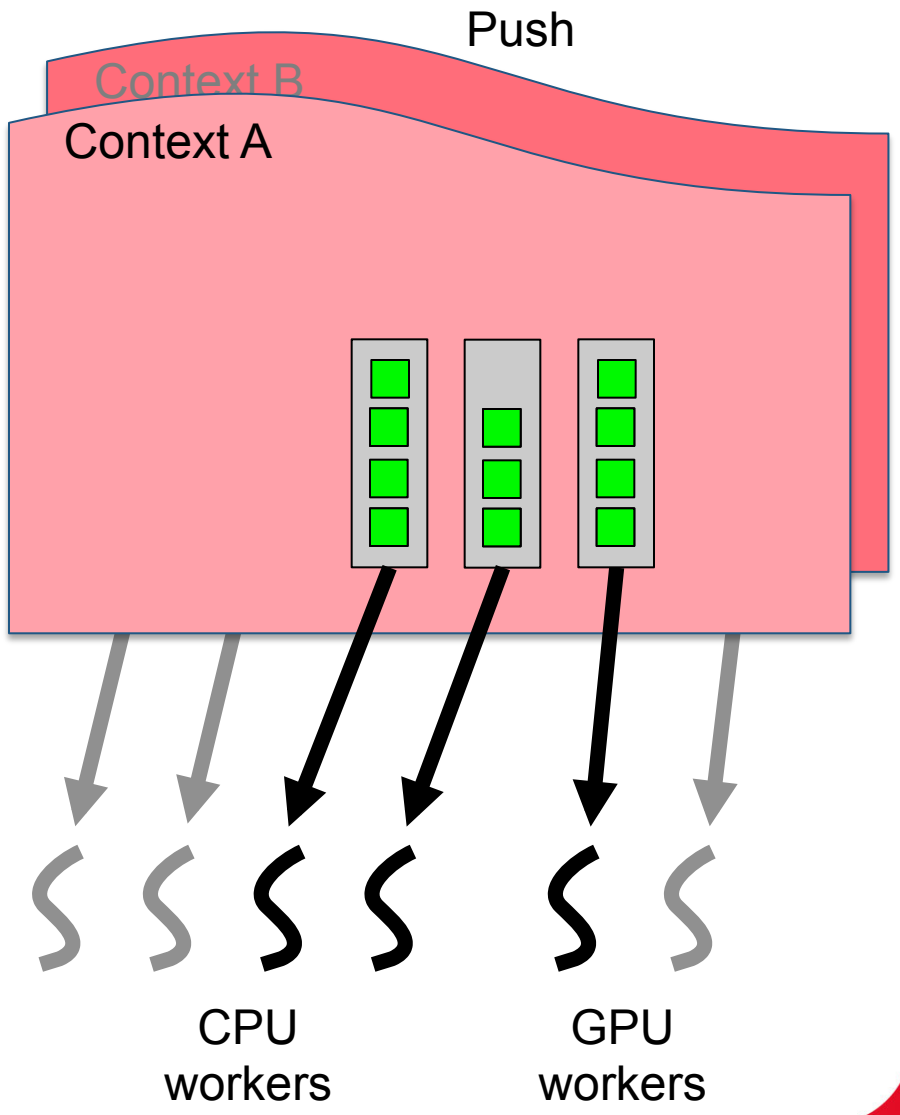
- It's all about composability!
  - Probably the biggest upcoming challenge for runtime systems
    - Hybridization will mostly be indirect (linking libraries)

- And with composability come a lot of related issues
  - Need for autotuning / scheduling hints

Scheduling contexts

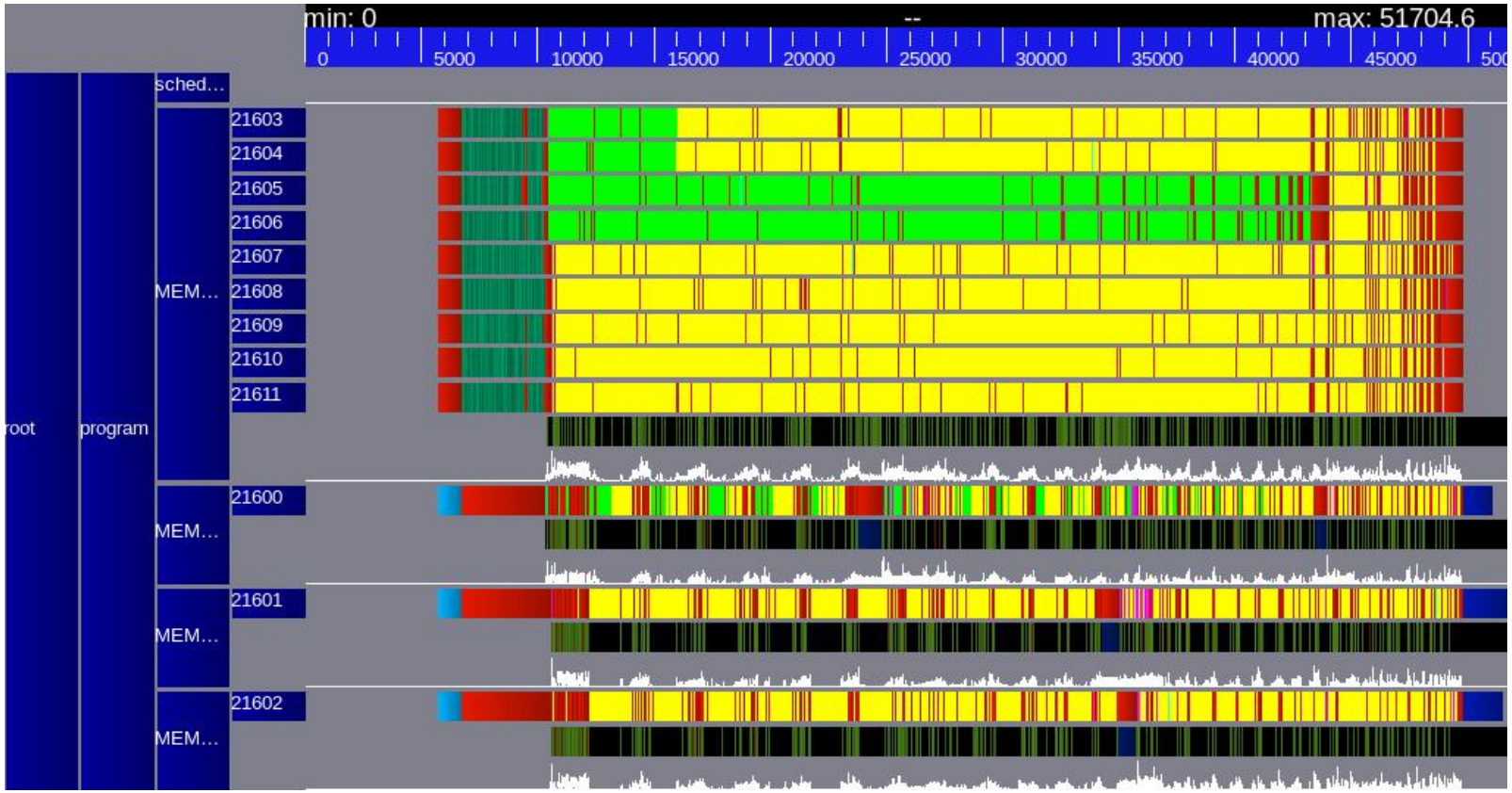
Toward code composability

- Each context features its own scheduler
- Multiple parallel libraries can run simultaneously
  - Virtualization of resources
  - Scalability workaround
- Contexts may share processing units
  - Avoid underutilized resources
- Contexts may expand and shrink
  - Hypervised approach
  - Maximize overall throughput
  - Use dynamic feedback both from application and runtime



Scheduling contexts

- Dynamic adjustment of context size
  - Based on computation velocity feedback



Future parallel machines

Exascale (10^{18} flop/s) systems, by 2020?

- The biggest change comes from node architecture

- Hybrid systems will be commonplace
  - Multicore chips + accelerators (GPUs?)
  - More integrated design

- Extreme parallelism
  - Total system concurrency ~ O(10^9)!
    - Including O(10) to O(100) to hide latency

= x 10 000 increase

How will we program these machines?

Let's prepare for serious changes

- Billions of threads will be necessary to occupy exascale machines
  - Exploit every source of (fine-grain) parallelism
    - Not every algorithm/problem resolution can scale that far ☹
  - Multi-scale, Multi-physics applications are welcome!
    - Great opportunity to exploit multiple levels of parallelism
  - Is SIMD the only reasonable approach?
    - Are CUDA & OpenCL our future?
- No global, consistent view of node's state
  - Local algorithms
  - Hierarchical coordination/load balance
- Maybe, this time, we should seriously consider enabling (parallel) code reuse…

Parallel code reuse

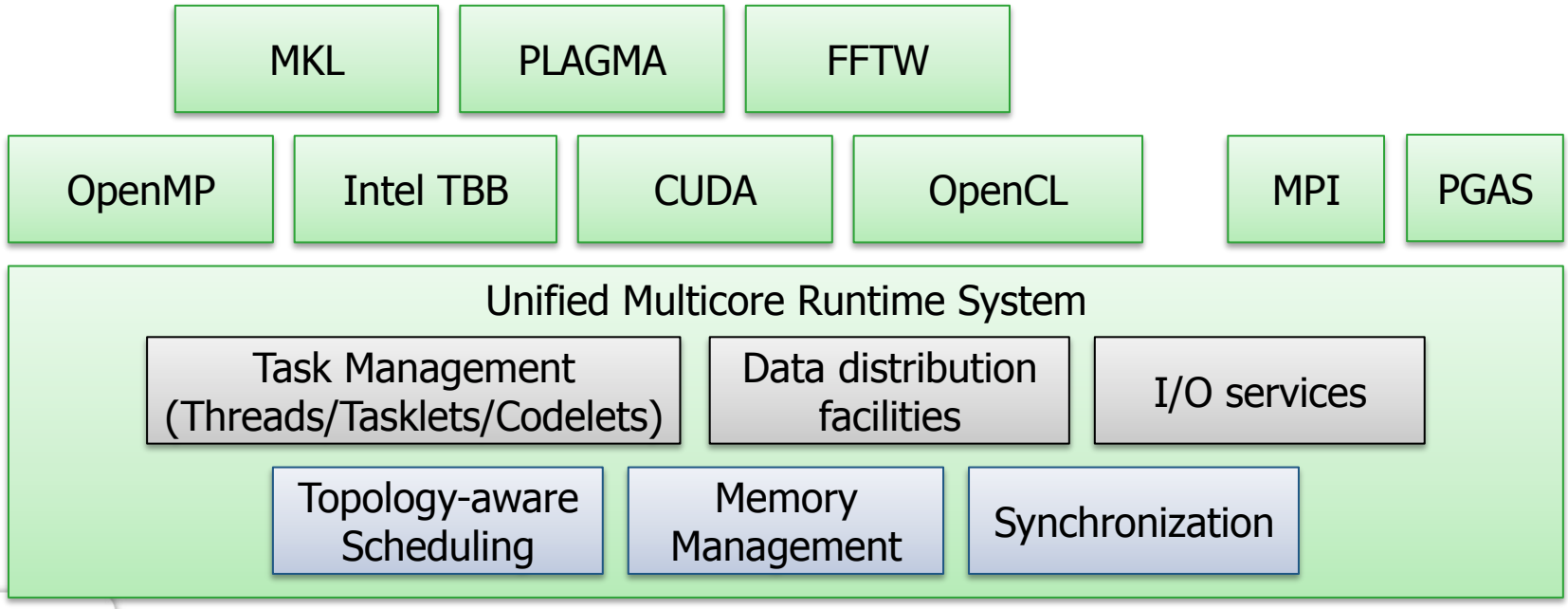
Mixing different paradigms leads to several issues

- Can we really use several hybrid parallel kernels simultaneously?
  - Ever tried to mix OpenMP and Intel 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

Toward a common runtime system?

i.e. Unified Software Stack

- There is currently no consensus on a common runtime system
  - One objective of the Exascale Software Center
    - Coordinated exascale software stack
  - Technically feasible…



Major Challenges are ahead…

We are living exciting times!

more information:
http://runtime.bordeaux.inria.fr

CEA/EDF/Inria 2013 Computer Science Summer School « Programming Heterogeneous Parallel Architectures » June 24-July 5, 2013 Cadarache

Directive-based programming
Michael WOLFE (Portland Group)

Programming Massively Parallel Processors Using CUDA and C++AMP
Wen-Mei HWU (University of Illinois at Urbana-Champaign)

Implicit and task-based approaches to heterogeneous parallel programming
Josef WEIDENDORFER (Technical University Munich)

http://www-hpc.cea.fr/SummerSchools2013-CS.htm