Task scheduling over Heterogeneous Multicore Machines: a Runtime Perspective

Raymond Namyst

"Runtime" group
INRIA Bordeaux Research Center
University of Bordeaux 1
France

Runtime Systems for Petascale Computing Systems: a Pessimistic View

Raymond Namyst

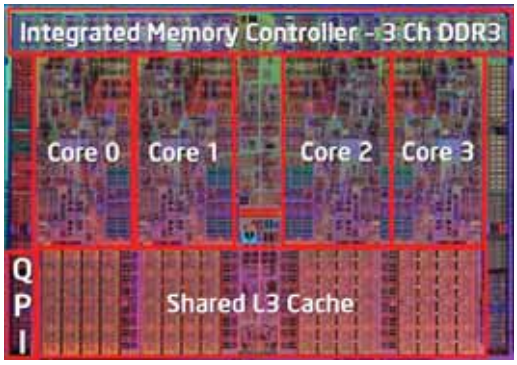
"Runtime" group
INRIA Bordeaux Research Center
University of Bordeaux 1
France

Outline

- The frightening evolution of parallel architectures
  - Multicore + coprocessors + accelerators = heterogeneous architectures
- New programming challenges
  - Hybrid programming models
- Designing runtime systems for heterogeneous machines
  - Scheduling and Memory consistency
- Challenges for the upcoming years
  - Current situation is terrible, but there is hope!

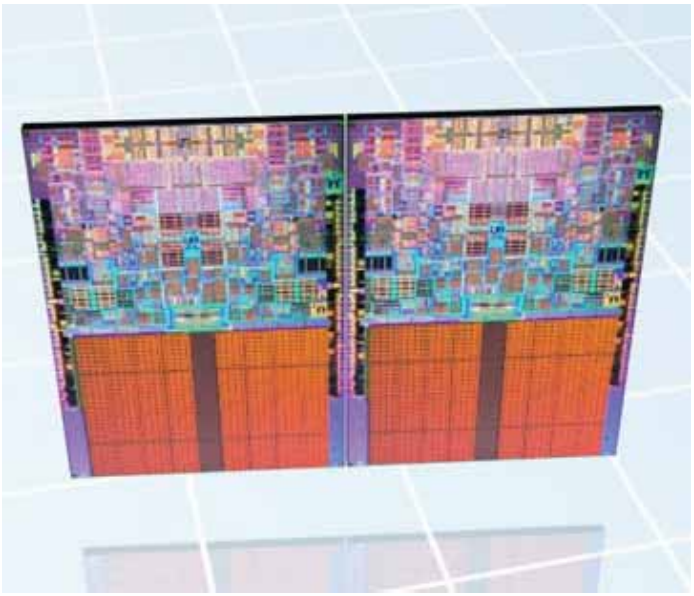
Multicore is a solid architecture trend

- 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…
  - Different from SMPs
    - Hierarchical chips
    - Getting really complex
  - Back to the CC-NUMA era?



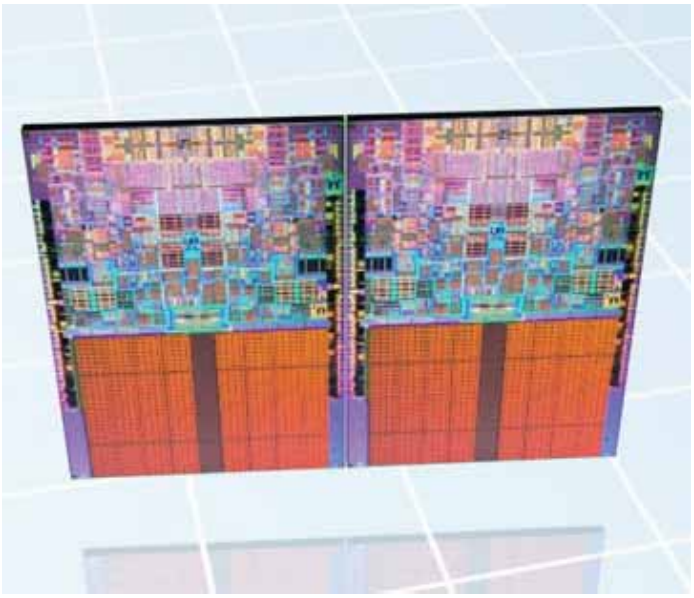
Machines are going heterogeneous

- GPGPU are the *new kids on the block*
  - Very powerful SIMD accelerators
  - Successfully used for offloading data-parallel kernels
- Other chips already feature specialized harware
  - IBM Cell/BE
    - 1 PPU + 8 SPUs
  - Intel Larrabee
    - 48-core with SIMD units



I mean “really more heterogeneous”

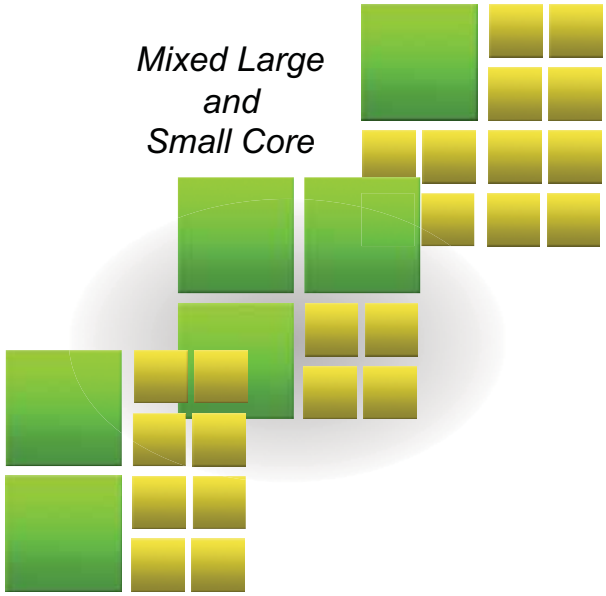
- Programming model
  - Specialized instruction set
  - SIMD execution model
- Memory
  - Size limitations
  - No hardware consistency
    - Explicit data transfers
- Are we happy with that?
  - No, but it’s probably unavoidable!



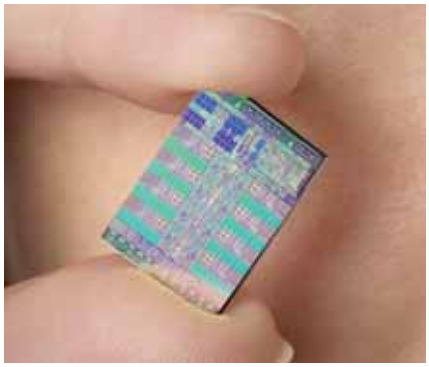
Heterogeneity is also a solid trend

- 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"

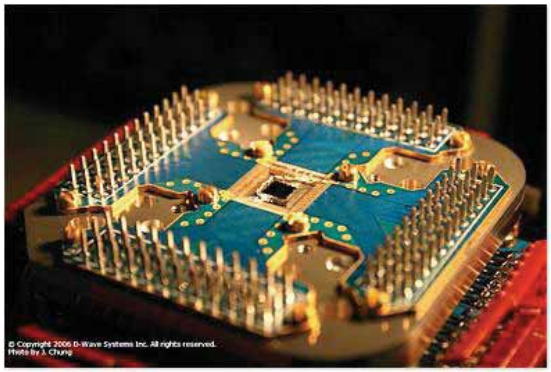
  [anonymous source]



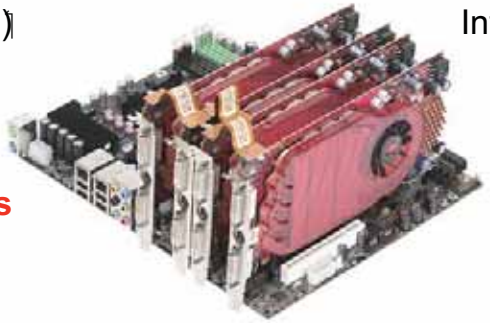
We have to get prepared!



IBM Cell (1+8 cores)



Intel TeraScale (80 cores)



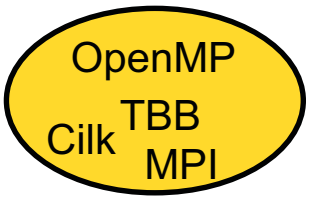
AMD graphic processors

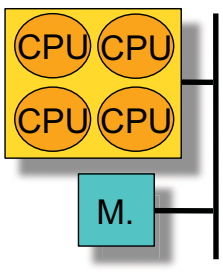
- **Understand today's accelerators**
- **Get ready for tomorrow's architectures**

New Programming Challenges

Programming homogeneous multicore machines

Multicore

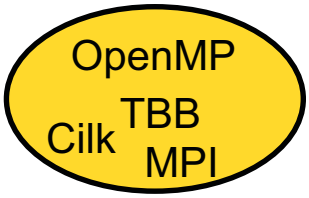


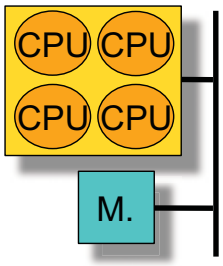


- Why not just try to extend existing solutions?
- Shared-memory approach
  - Scalability
  - NUMA-awareness
  - Affinity-guided scheduling
- Message passing approach
  - Cache-friendly buffers
  - Topology-awareness
  - Collective

Programming homogeneous multicore machines

Multicore

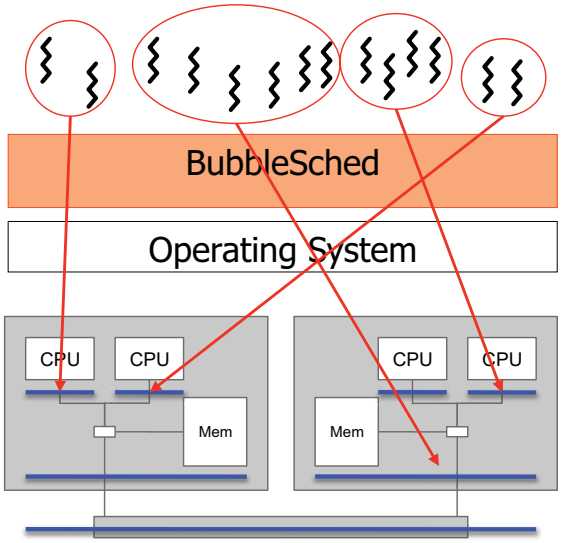




- OpenMP
  - Scheduling in a NUMA context (memory affinity, work stealing)
  - Memory management (page migration)
- MPI
  - NUMA-aware buffer management
  - Efficient collective operations
- Also several interesting approaches
  - Intel TBB, SMP-superscalar, etc.
  - Idea = we need fine-grain parallelism!

Our background: Thread Scheduling over Multicore Machines

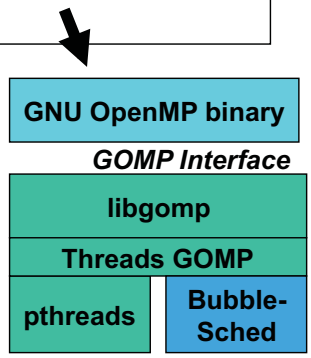
- The Bubble Scheduling concept
  - Capturing application's structure with nested bubbles
  - Scheduling = dynamic mapping trees of threads onto a tree of cores
- The BubbleSched platform
  - Designing portable NUMA-aware scheduling policies
    - Focus on algorithmic issues
  - Debugging/tuning scheduling algorithms
    - FxT tracing toolkit + replay animation
    - [with Univ. New Hampshire, USA]



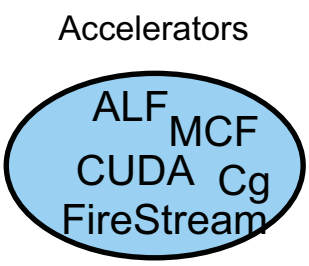
Our background: Thread Scheduling over Multicore Machines

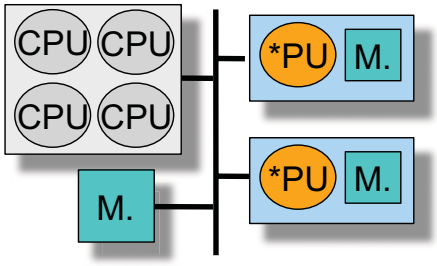
- Designing multicore-friendly programs with OpenMP
  - Parallel sections generate bubbles
  - Nested parallelism is welcome!
    - Lazy creation of threads
- The ForestGOMP platform
  - Extension of GNU OpenMP
    - Binary compliant with existing applications
  - Excellent speedups with irregular applications
    - Implicit 3D surface reconstruction [with iParla]
    - Tree depth > 15, more than 300,000 threads
- BubbleSched also targeted by OMPi
  - [with Univ. of Ioannina, Greece]

```
void Node::compute(){

  // approximate surface
  computeApprox();

  if(_error > _max_error) {
    // precision not sufficient
    // so divide and conquer
    splitCell();

    #pragma omp parallel for
    for(int i=0; i<8; i++)
      _children[i]->compute();
  }
}
```



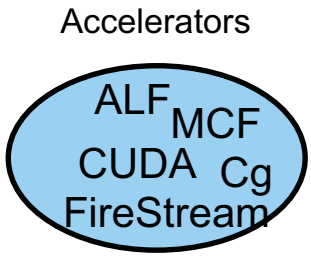
Dealing with heterogenenous accelerators

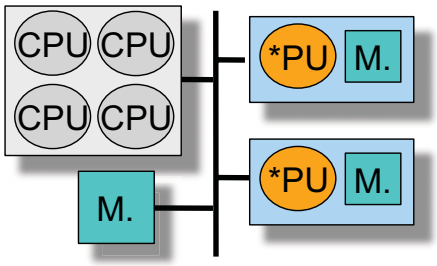




- Specific APIs
  - CUDA, IBM SDK, …
  - No consensus
    - Specialized languages/ compilers
  - OpenCL?
- Communication libraries
  - MCAPI, MPI

Dealing with heterogenenous accelerators



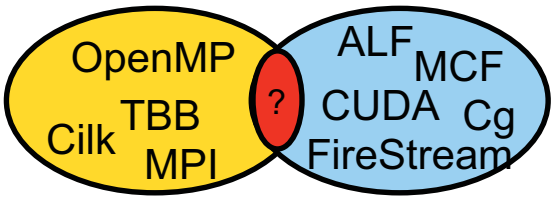


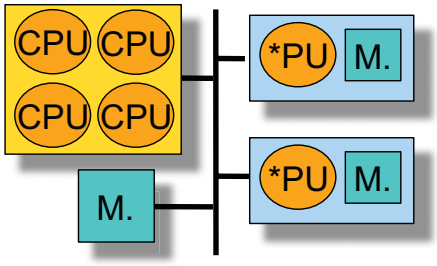
- Language extensions
  - RapidMind, Sieve C++
  - HMPP

    #pragma hmpp target=cuda
  - Cell Superscalar

    #pragma css input(..) output(…)
- Most approaches focus on *offloading*
  - As opposed to *scheduling*

Programming Hybrid Architectures



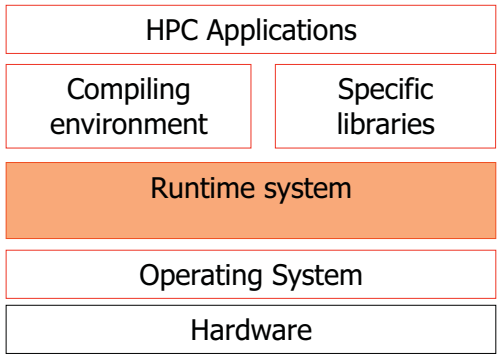


- Challenge = exploiting all computing units simultaneously
- Either use a hybrid programming model
  - E.g. OpenMP + HMPP + Intel TBB + CUBLAS + MKL + …
- Or use a uniform programming model
  - That doesn't exist yet…

In either case,
a common runtime
system is needed!

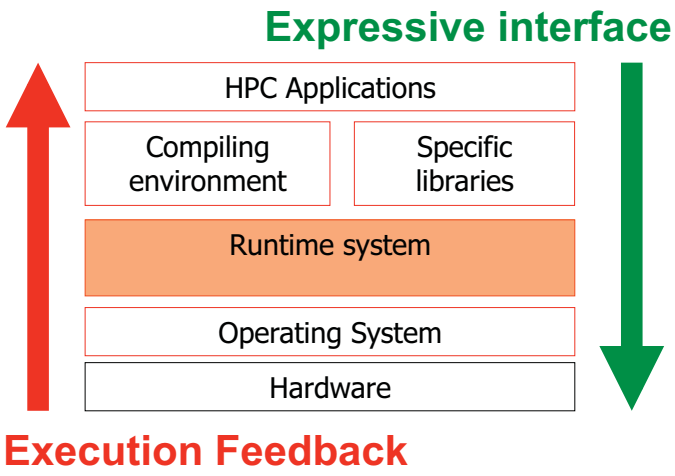
Runtime Systems for Heterogeneous Multicore Architectures

- Runtime systems
  - Perform dynamically what can't be done statically
  - Hide hardware complexity, provide portability (of performance?)
- Just a matter of providing yet another scheduling & memory management API?



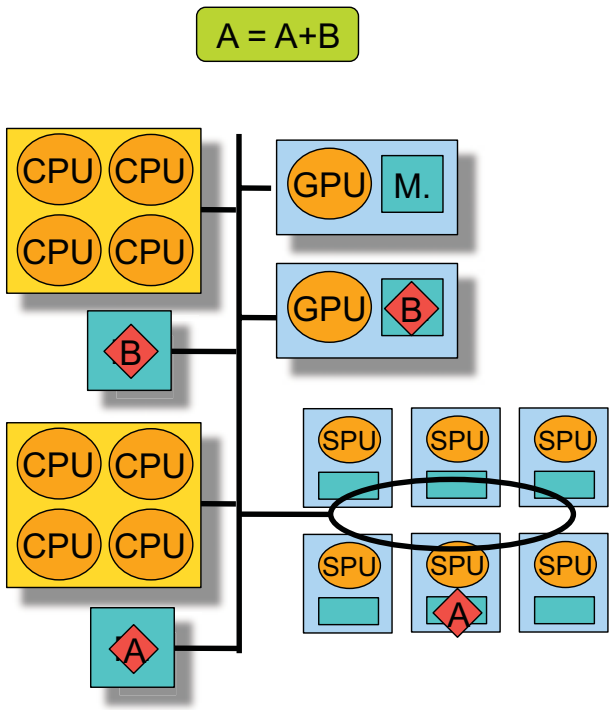
Runtime Systems for Heterogeneous Multicore Architectures

- Programmers (usually) know their application
  - Don't guess what we know!
  - Scheduling hints
- Feedback is important
  - E.g. Performance counters
  - Adaptive applications?
- Other Issues
  - Can we still find a unified execution model?
  - How to determine the appropriate task granularity?



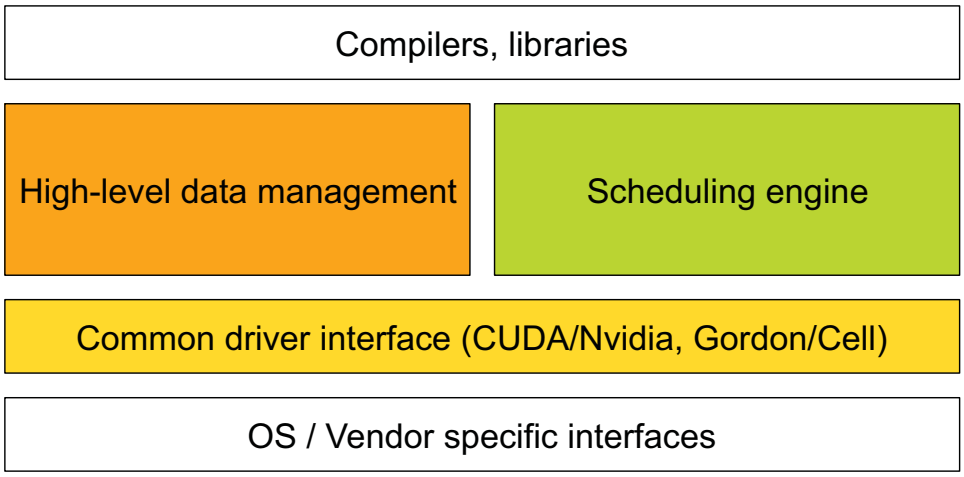
Towards a unified execution model

- We wanted our runtime to fulfill the following requirements:
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous cores
  - Avoid unnecessary data transfers between accelerators
    - Need to keep track of data copies



The StarPU Runtime System

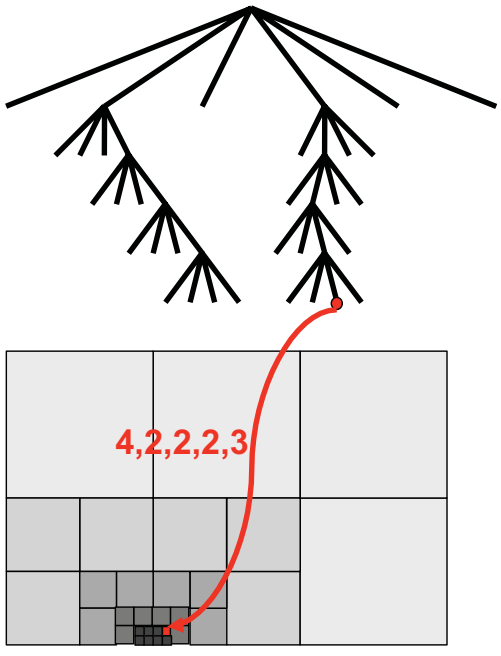
Cédric Augonnet, Samuel Thibault



Mastering CPUs, GPUs, SPUs ...
(hence the name: ***PU**)

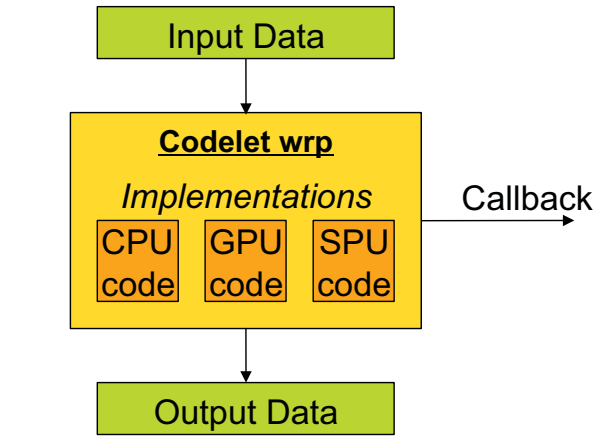
High-Level Data Management

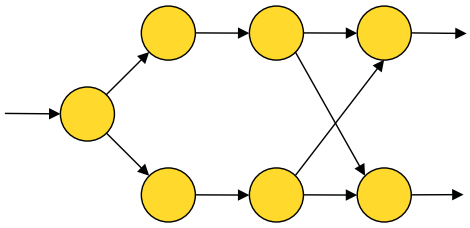
- All we need is a Software DSM system!
  - Consistency, replication, migration
  - Concurrency, accelerator to accelerator transfers
  - Memory reclaiming mechanism
    - Problem size > accelerator size
- Data partitioned with filters
  - Various interfaces
    - BLAS, vector, CSR, CSC
  - Recursively applied
    - Structured data = tree



Scheduling Engine

- Tasks are manipulated through "codelet wrappers"
  - May provide multiple implementations
    - Scheduling hints
      - Optional cost model per implementation, priority, …
  - List data dependencies
    - Using the filter interface
  - Maybe automatically generated
- Schedulers are plug-ins
  - Assign tasks to run queues
  - Dependencies and data prefetching are hidden

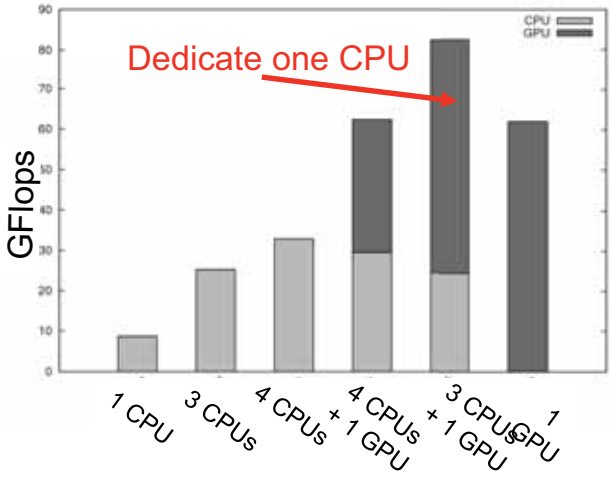




Evaluation

Blocked matrix multiplication

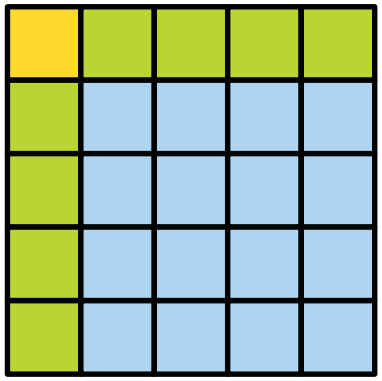
- ✓ Exploit heterogeneous platform
  - 4 CPUs + 1 GPU
- ✓ CPUs must not be neglicted!
- ✗ Issues with 4 CPUs + 1 GPU
  - Busy CPU delays GPU management
  - Cache-sensitive CPU code
- Trade-off : dedicate one core

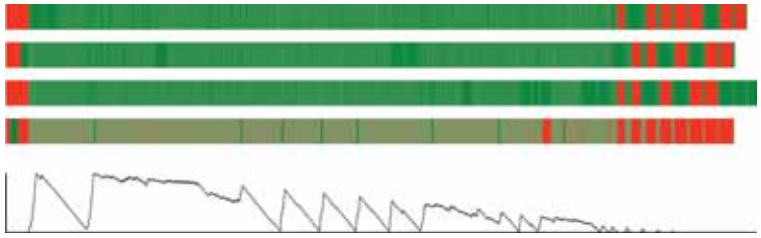


quadcore Intel Xeon
+ nVidia Quadro FX4600

Evaluation

Dense LU decomposition



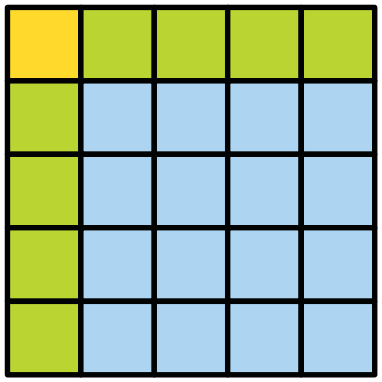


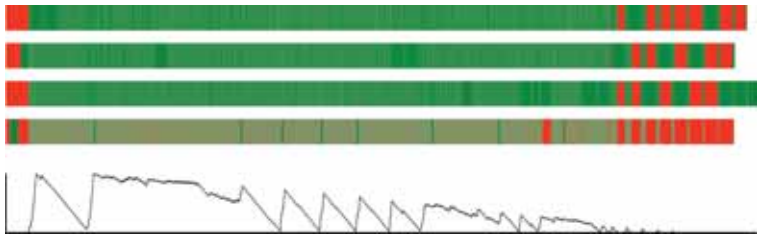
Some tasks are critical for the algorithm

Lack of parallelism
Cannot feed all *PUs with enough work

Evaluation

Dense LU decomposition



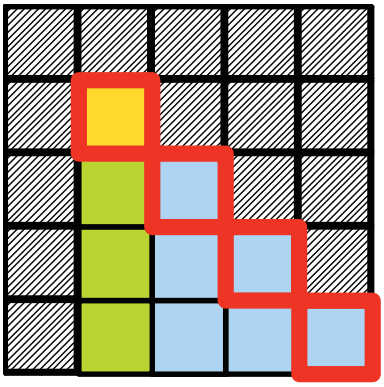


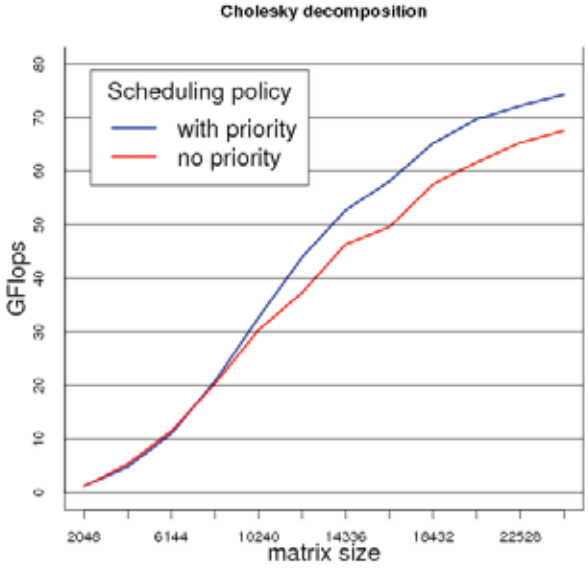
Some tasks are critical for the algorithm

...Even worse with Cholesky !

Evaluation

Cholesky decomposition

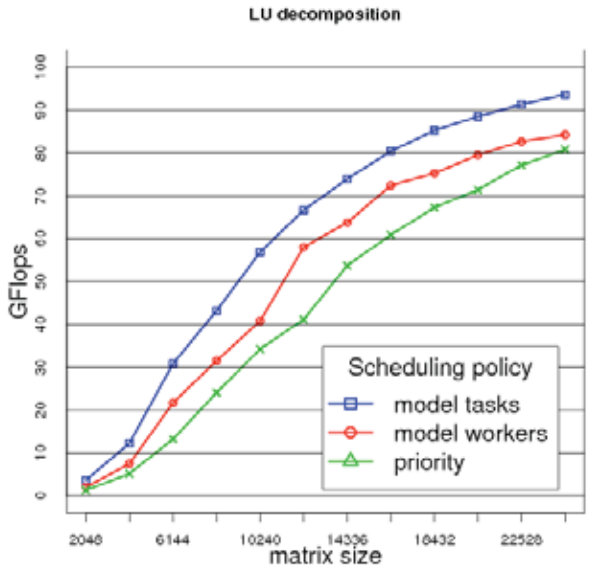




Priorities -> gain ~ 10 %

Evaluation

About the importance of performance models



Modeling **workers**' performance

- "1 GPU = 10x faster than 1 CPU"
- Reduce load imbalance
- Fuzzy approximation

Modeling **tasks** execution time

- Precise performance models
  - "mathematical" models
  - user-provided models
- automatic "learning" for unknown codelets

What did we learn?

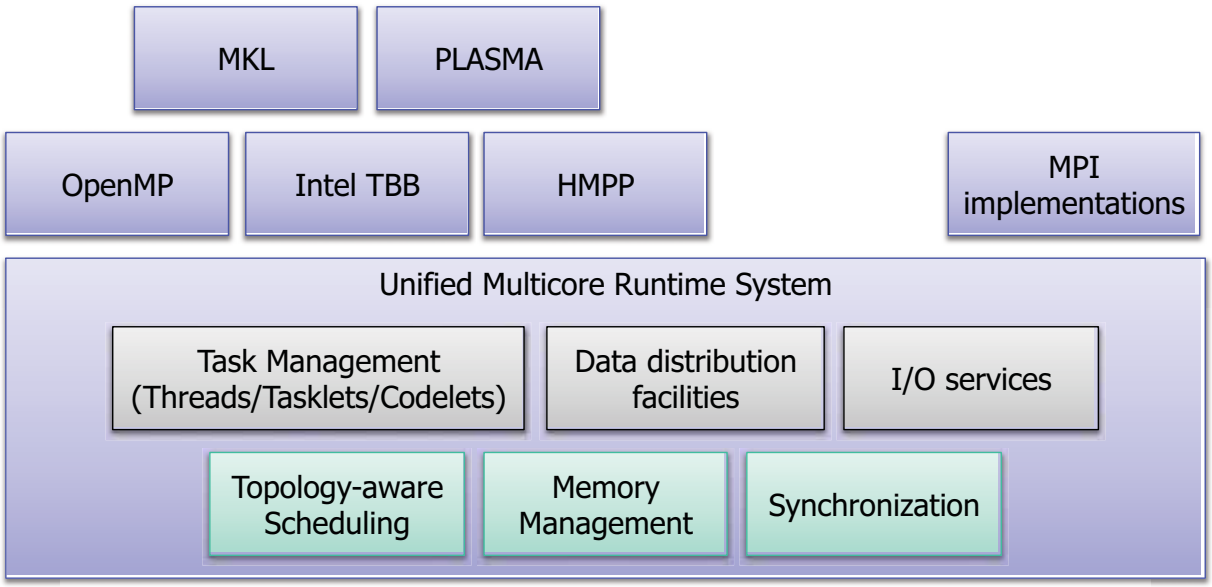
- All computing units must be used simultaneously to achieve high performance
  - "Pure offloading"is not sufficient
- Performance models have a high impact over scheduling quality
  - Rather easy for numerical kernel, but for other algorithms?
- Finding the best task granularity is *very* difficult
  - Has to be decided dynamically!

Challenges for the upcoming years

- Integration with "traditionnal" multithreading solutions
  - We can't seriously consider codeletizing the world…
  - E.g. support execution of OpenMP + HMPP (+ StarPU kernels) programs
- Towards a tighter integration of hardware within runtime systems
  - Adaptive, portable scheduling/optimization strategies
    - Linking hardware performance counters to application-level abstractions
    - Using hardware feedback to refine/correct scheduling directives
- Enhance cooperation between runtime systems and compilers
  - Runtime support for "divisible tasks"

Challenges for the upcoming years

- There's currently no consensus for a *common runtime system*
  - But future application will be composed of several types of bricks



Thank you!

- More information about Runtime

  http://runtime.bordeaux.inria.fr

- More information about StarPU and ForestGOMP

  http://runtime.bordeaux.inria.fr/starpu

  http://runtime.bordeaux.inria.fr/forestgomp

- Software available on INRIA Gforge:

  http://gforge.inria.fr/projects/pm2/