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

Raymond Namyst
University of Bordeaux
Head of RUNTIME group, INRIA

École CEA-EDF-INRIA 2011
Sophia-Antipolis, June 6-10

Understanding the evolution of parallel machines

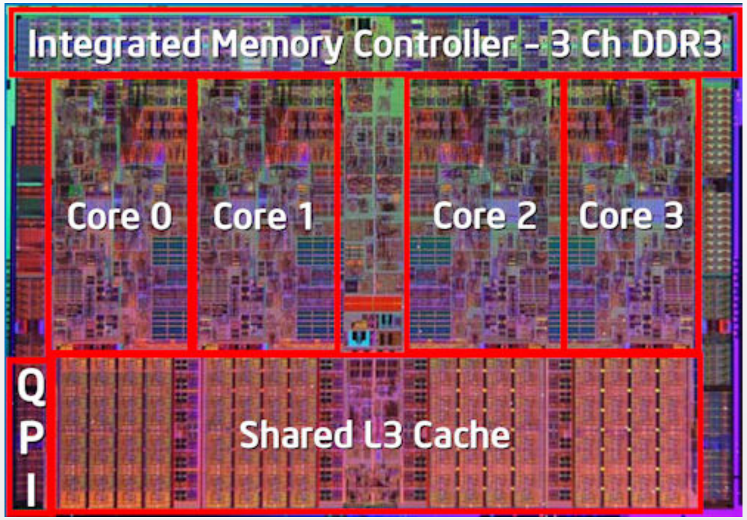
The end of Moore's law?

- 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 circuits should we better add on a die?

Understanding the evolution of parallel machines

Welcome to the multicore era

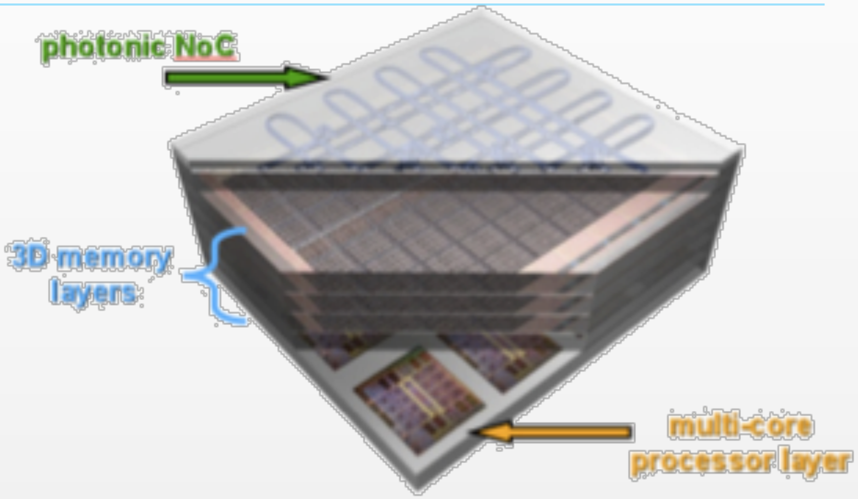
- Answer: Multicore chips
  - Several cores instead of one processor
  - Back to complex memory hierarchies
    - Shared caches
      - Organization is vendor-dependent
    - NUMA penalties
  - Clusters can no longer be considered as "flat sets of processors"

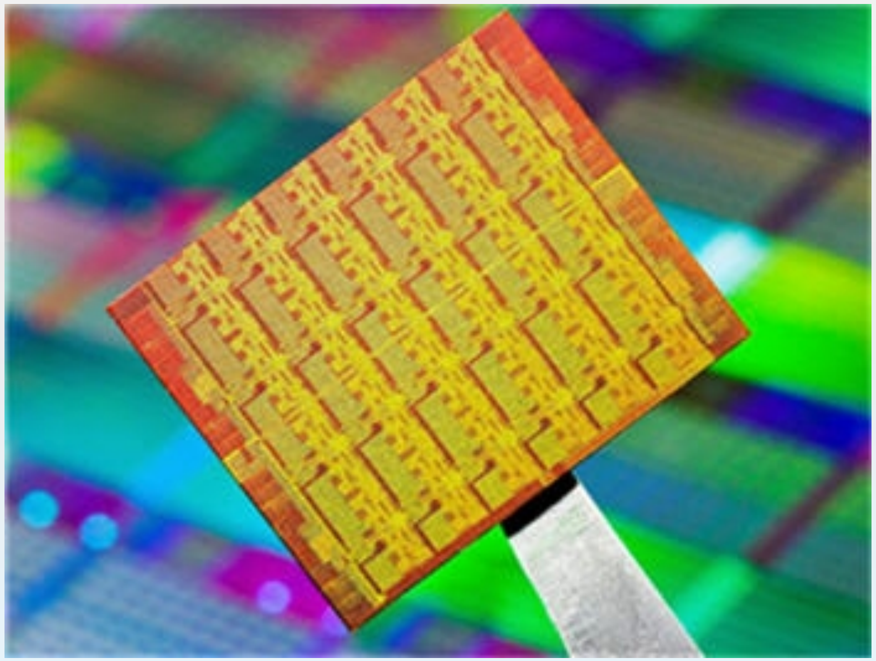


Understanding the evolution of parallel machines

Multicore is a solid trend

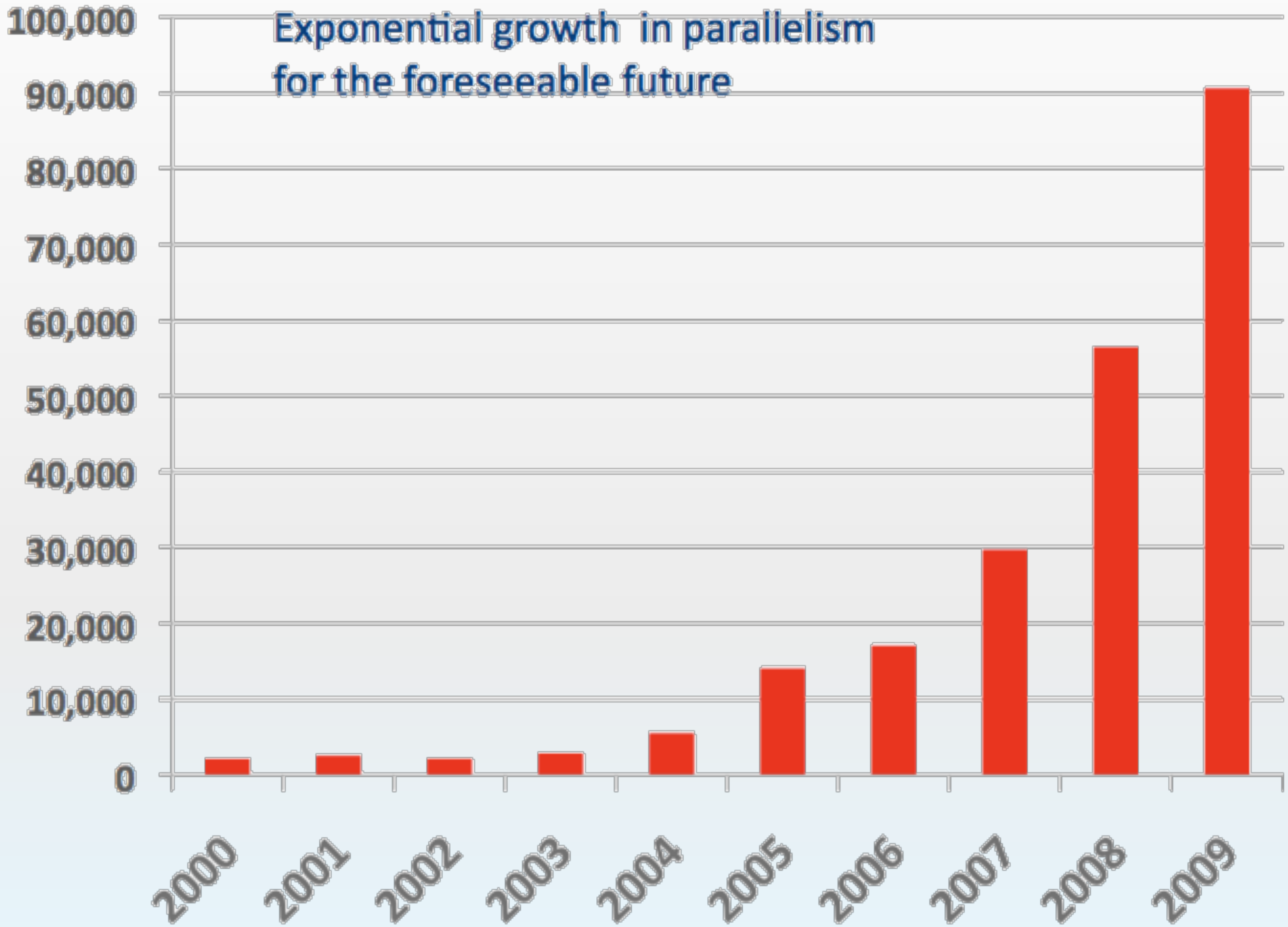
- More performance = more cores
  - Toward embarrassingly parallel machines?
- Designing scalable multicore architectures
  - 3D stacked memory
  - Non-coherent cache architectures
    - Intel SCC
    - IBM Cell/BE





Understanding the evolution of parallel machines

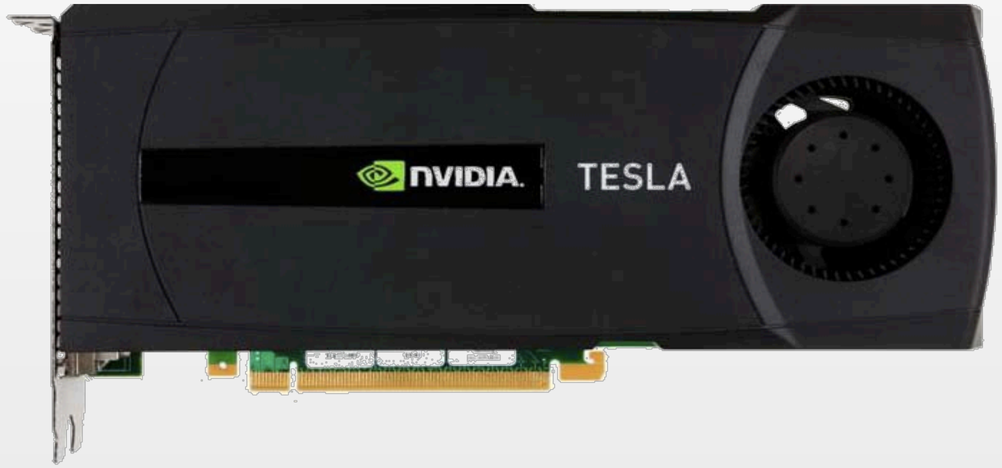
Average number of cores per top20 supercomputer



Heterogeneous computing is here

And portable programming is getting harder…

- GPUs are the *new kids on the block*
  - De facto adoption
  - Concrete success stories
    - “Speedups” > 50

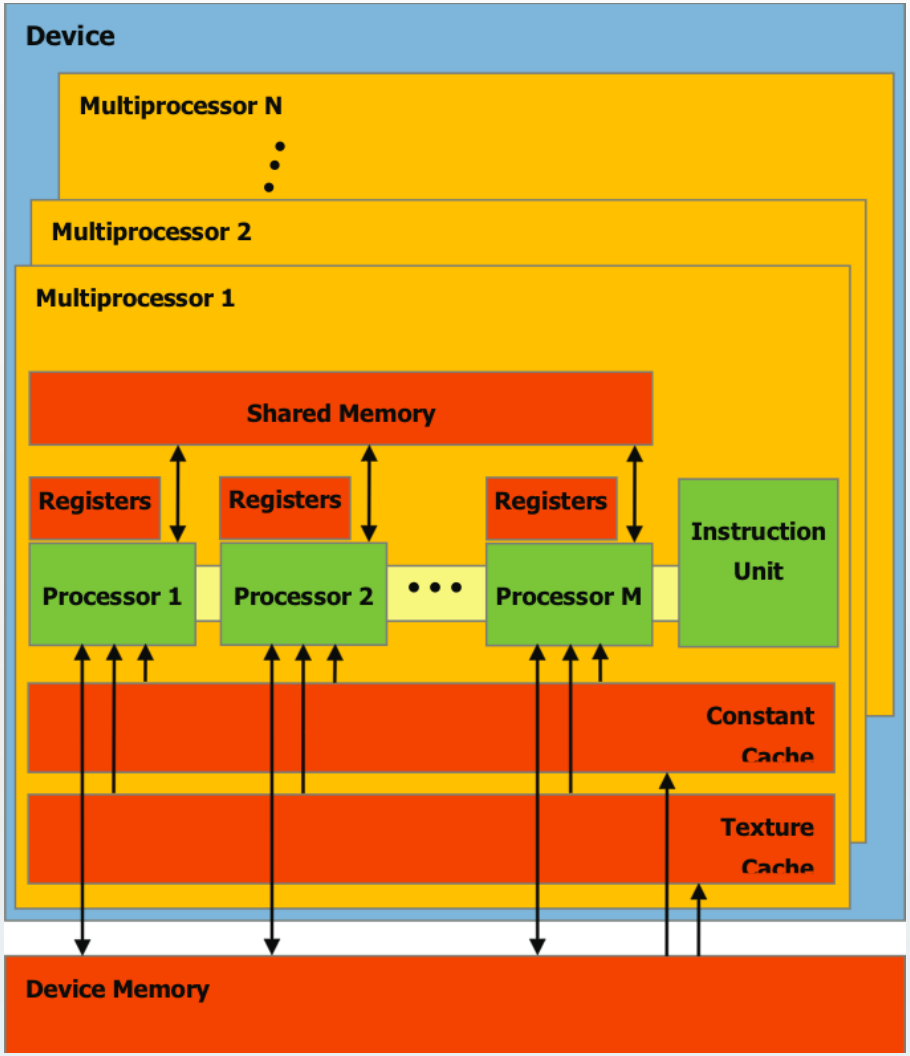


- Clusters featuring accelerators are already heading the Top500 list
  - Tianhe-1A (#1)
  - Nebulae (#3)
  - Tsubame 2.0 (#4)
  - Roadrunner (#7)

Heterogeneous computing is here

And portable programming is getting harder…

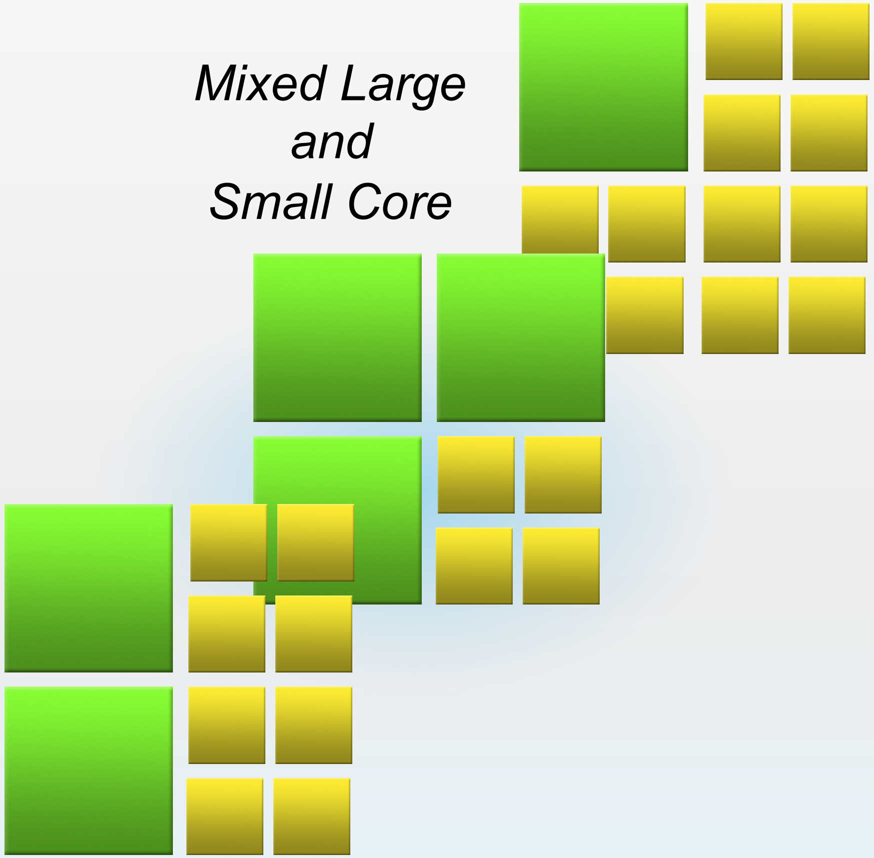
- Programming model
  - Specialized instruction set
  - SIMD execution model
    - Nvidia Fermi GTX 480
      - 512 cores
- Memory
  - Size limitations
  - No hardware consistency
    - Explicit data transfers
- Using GPUs as “side accelerators” is not enough
  - GPU = first class citizens



Heterogeneous computing is here

And it seems to be a solid trend...

- "Future processors will be a mix of general purpose and specialized cores" (anonymous source)
  - One interpretation of "Amdalh's law"
    - Need powerful, general purpose cores to speed up sequential code
- Accelerators will be more integrated
  - Intel Knights Corner (MIC), SandyBridge
  - AMD Fusion
  - Nvidia Tegra-like
- Are we happy with that?
  - No, but it's probably unavoidable!



The Quest for programming models

What Programming Models for such machines?

Widely used, standard programming models

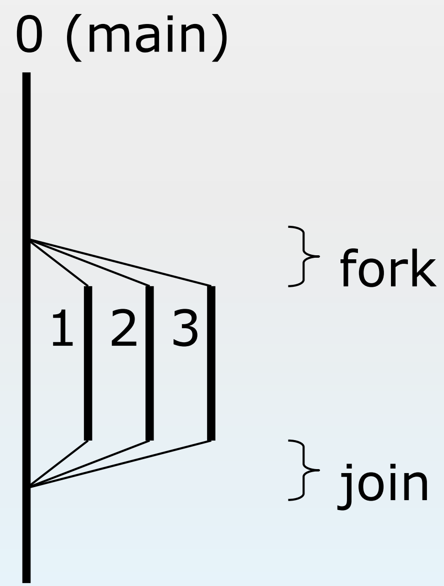
- MPI
  - Communication Interface
  - Scalable implementations exist already
    - Was actually designed with scalability in mind
    - Makes programmers “think” scalable algorithms
  - NUMA awareness?
  - Memory consumption

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

OpenMP (1997)
A portable approach to shared-memory programming

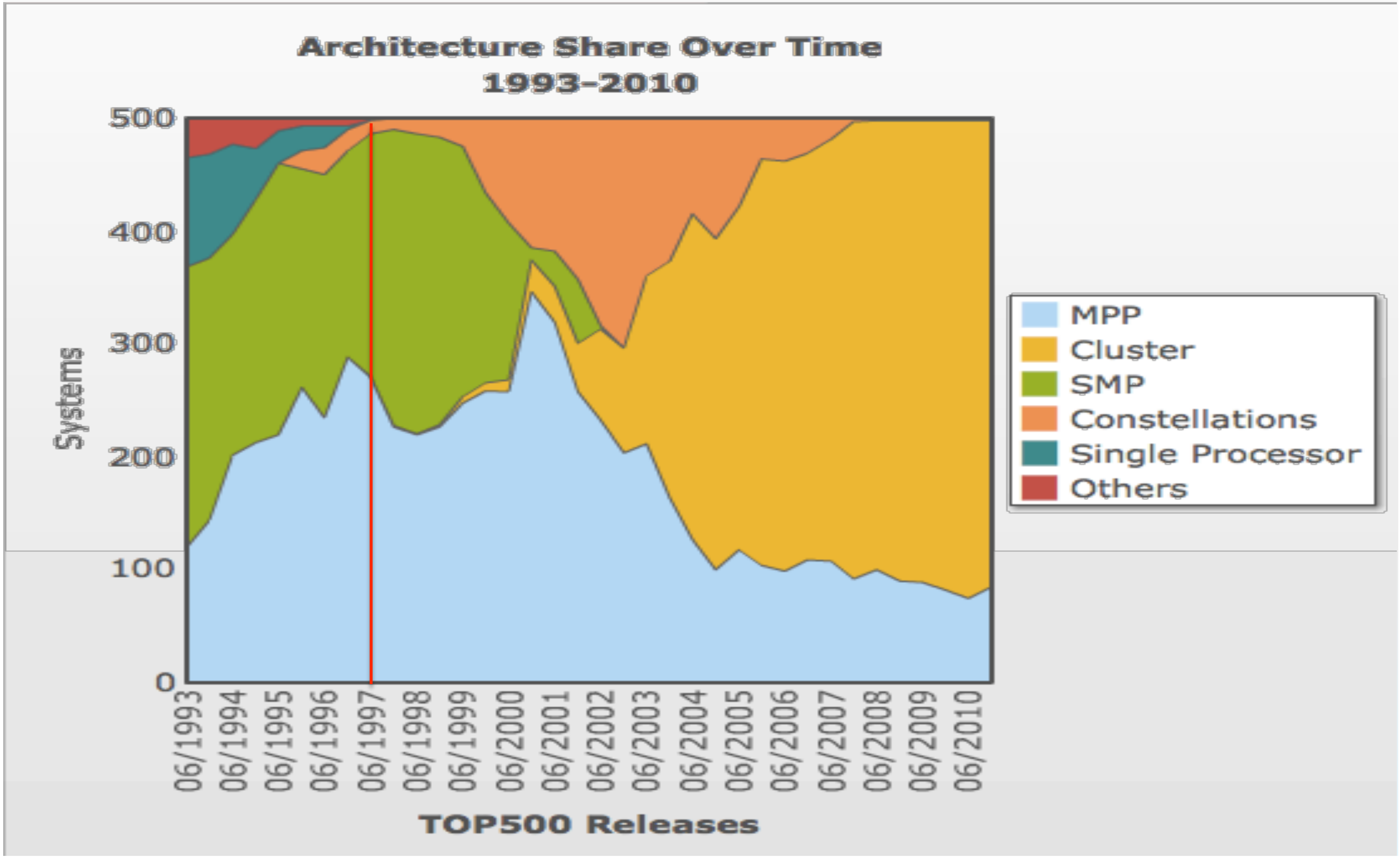
- Extensions to existing languages
  - C, C++, Fortran
  - Set of programming directives
- Fork/join approach
  - Parallel sections
- Well suited to data-parallel programs
  - Parallel loops
- OpenMP 3.0 introduced *tasks*
  - Support for irregular parallelism

```
int matrix[MAX][MAX];
  ...
#pragma omp parallel for
  for (int i; i < 400; i++)
  {
     matrix[i][0] += ...
  }
```



OpenMP (1997)

Multithreading over shared-memory machines

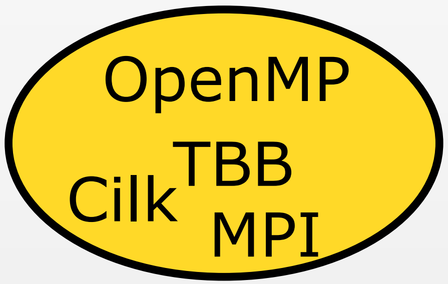


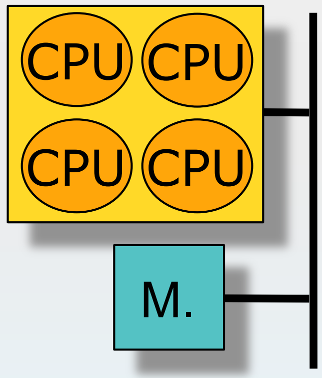
The Quest for Programming Models

Dealing with multicore machines

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

Multicore

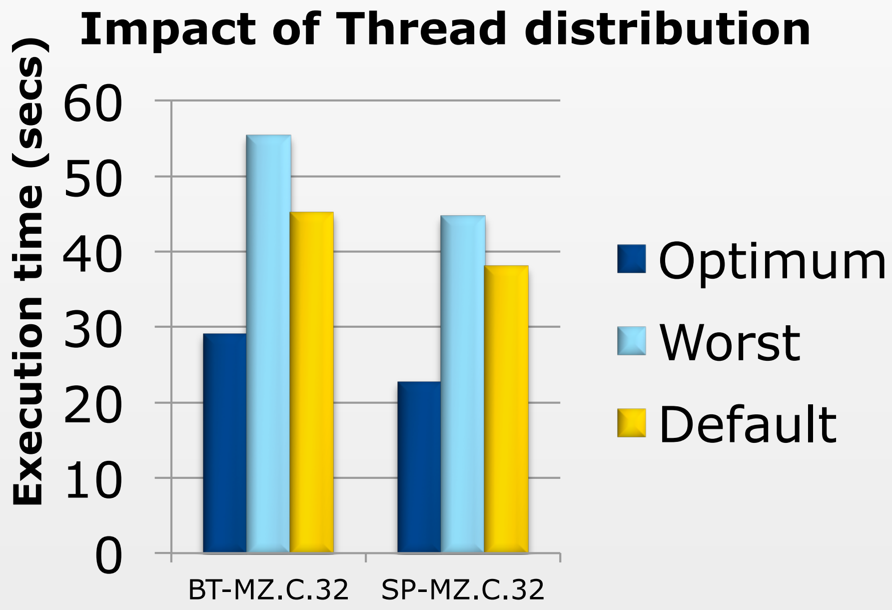


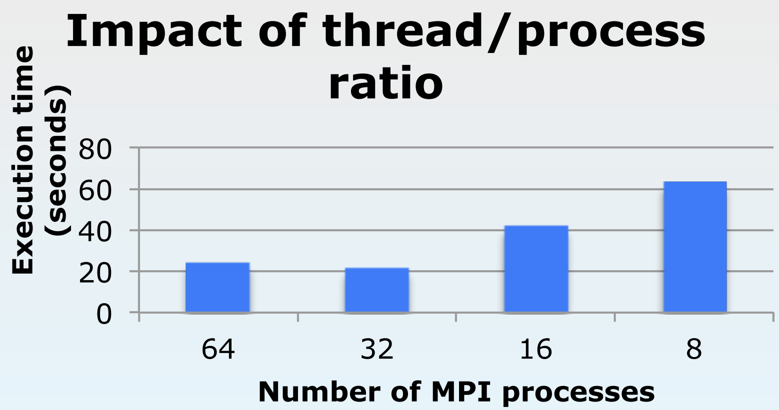


Mixing OpenMP with MPI

It makes sense even on shared-memory machines

- MPI address spaces must fit the underlying topology
- Experimental platforms exit to hybrid applications
  - Topology-aware process allocation
  - Customizable core/process ratio
  - # of OpenMP tasks independent from # of cores



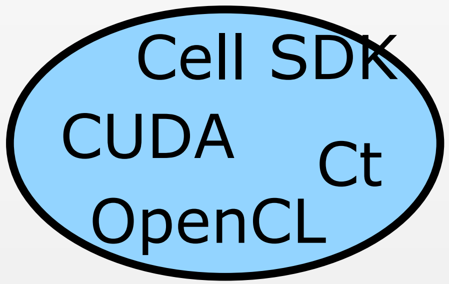


The Quest for Programming Models

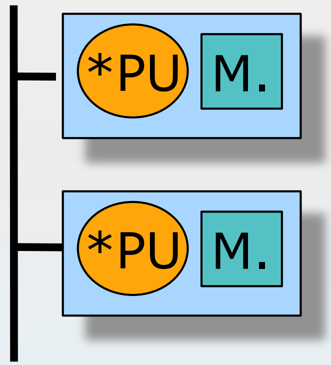
Dealing with accelerators

- 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

Accelerators



- Examples
  - Nvidia's CUDA
    - *Compute Unified Device Architecture*)
  - IBM Cell SDK
  - OpenCL



The Quest for Programming Models

The hidden beauty of CUDA

```
__global__ void mykernel(float * A1, float * A2, float * R)
{
    int p = threadIdx.x;
    R[p] = A1[p] + A2[p];
}

int main()
{
    float A1[]={1,2,3,4,5,6,7,8,9},  A2[]={10,20,30,40,50,60,70,80,90},  R[9];
    int size=sizeof(float) * 9;
    float *a1_device, *a2_device, *r_device;
    cudaMalloc ( (void**) &a1_device, size); cudaMalloc ( (void**) &a2_device, size); cudaMalloc ( (void**) &r_device, size);
    cudaMemcpy( a1_device,A1,size,cudaMemcpyHostToDevice); cudaMemcpy( a2_device,A2,size,cudaMemcpyHostToDevice);

    mykernel<<<1,9>>>(a1_device, a2_device, r_device);

    cudaMemcpy(R,r_device,taille_mem,cudaMemcpyDeviceToHost) ;
}
```

The Quest for Programming Models

Are we forced to use such low-level tools?

- Fortunately, well-known kernels are available
  - BLAS routines
    - e.g. CUBLAS
  - FFT kernels
- Implementations are continuously enhanced
  - High Efficiency
- Limitations
  - Data must usually fit accelerators memory
  - Multi-GPU configurations not well supported
- Ongoing efforts
  - Using multi-GPU + multicore
    - MAGMA (Oak Ridge National Lab)

Directive-based approaches

Offloading tasks to accelerators

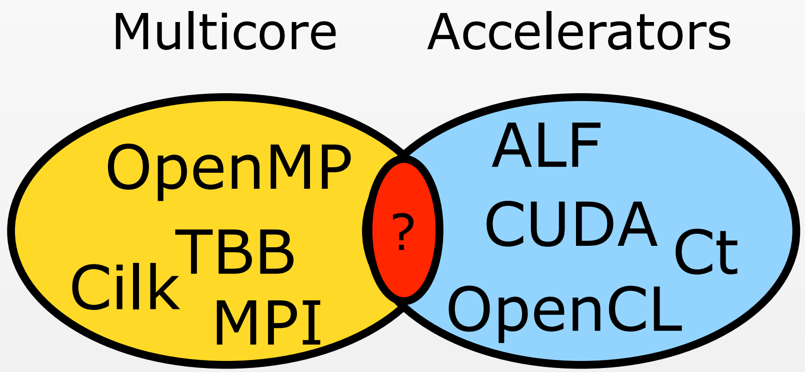
- Idea: use simple directives… and better compilers
  - HMPP (Caps Enterprise)
  - GPU SuperScalar (Barcelona Supercomputing Center)

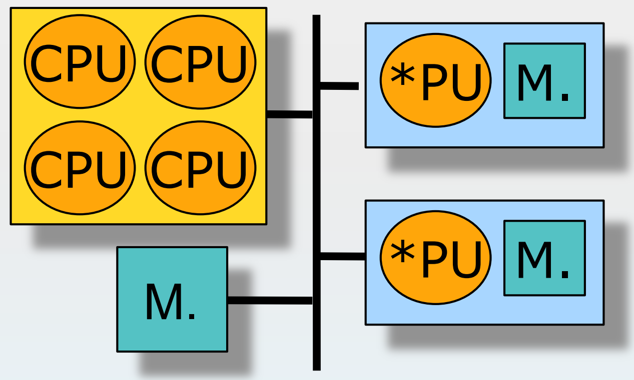
```
#pragma omp task inout(C[BS][BS])
void matmul( float *A, float *B, float *C) {
// regular implementation
}
#pragma omp target device(cuda) implements(matmul)
copy_in(A[BS][BS] , B[BS][BS] , C[BS][BS])
copy_out(C[BS][BS])
void matmul cuda ( float *A, float *B, float *C) {
// optimized kernel for cuda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

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}
  - 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

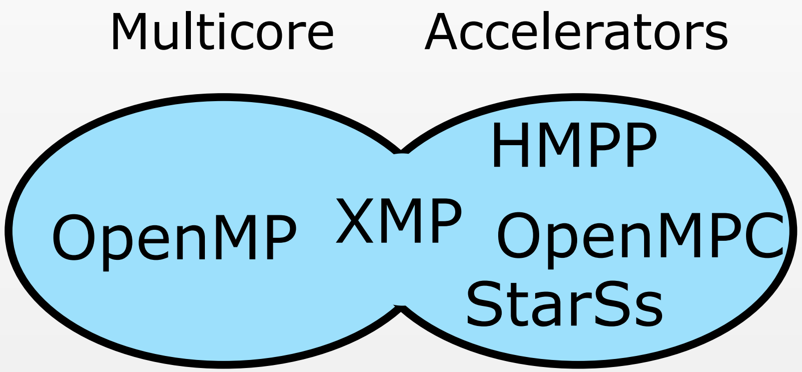


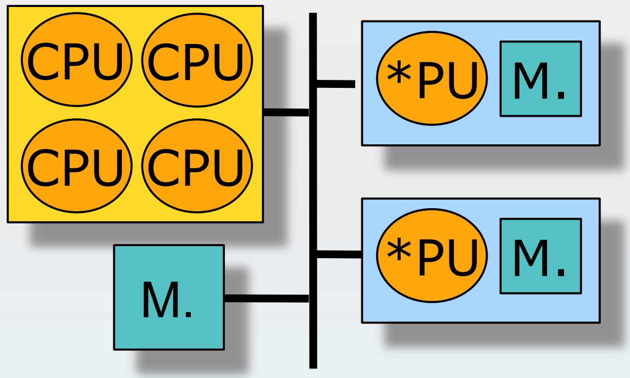


The Quest for Programming Models

How shall we program heterogeneous clusters?

- The uniform way
  - Use 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!





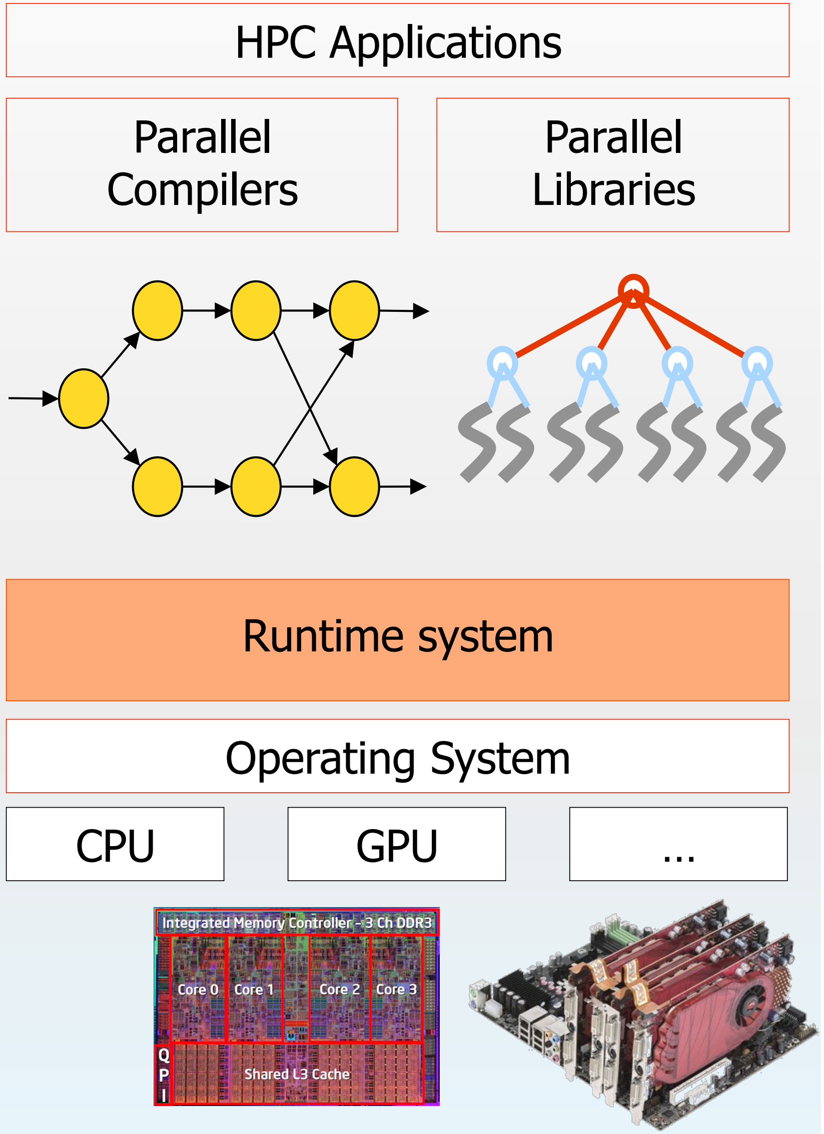
All the things runtime systems can do for you

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!



We need **new** runtime systems!

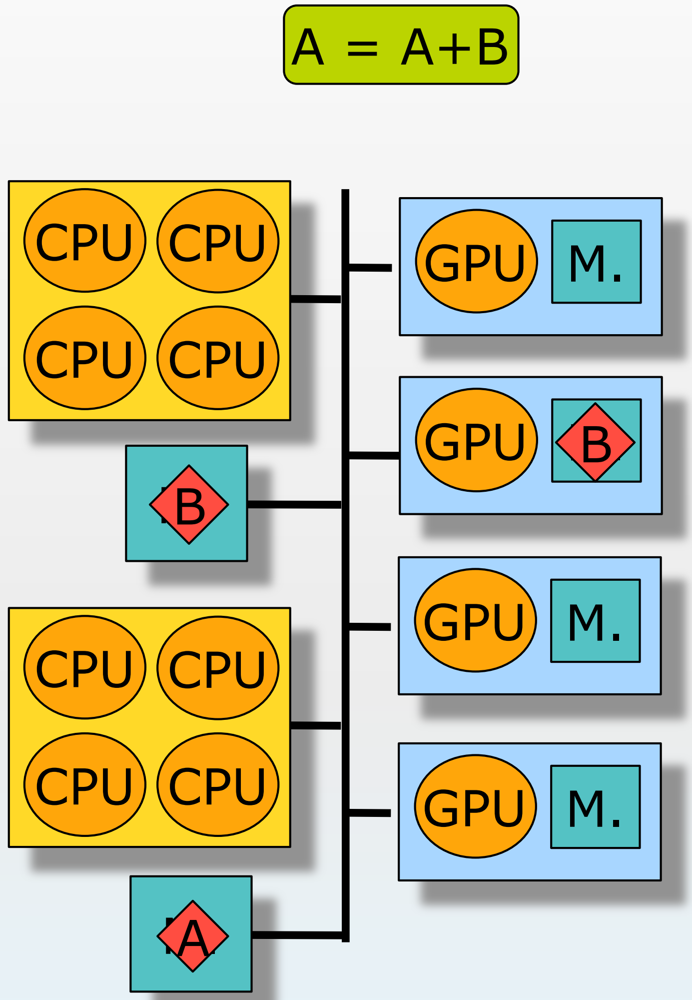
Toward "portability of performance"

- Computations need to exploit accelerators and regular CPUs simultaneously
- Data movements between memory banks
  - Should be minimized
  - Should not be triggered explicitly by application
- Computations need to accommodate to a variable number of processing units
  - Some computations do not scale over a large #cores

Overview of StarPU

A runtime system for heterogeneous architectures

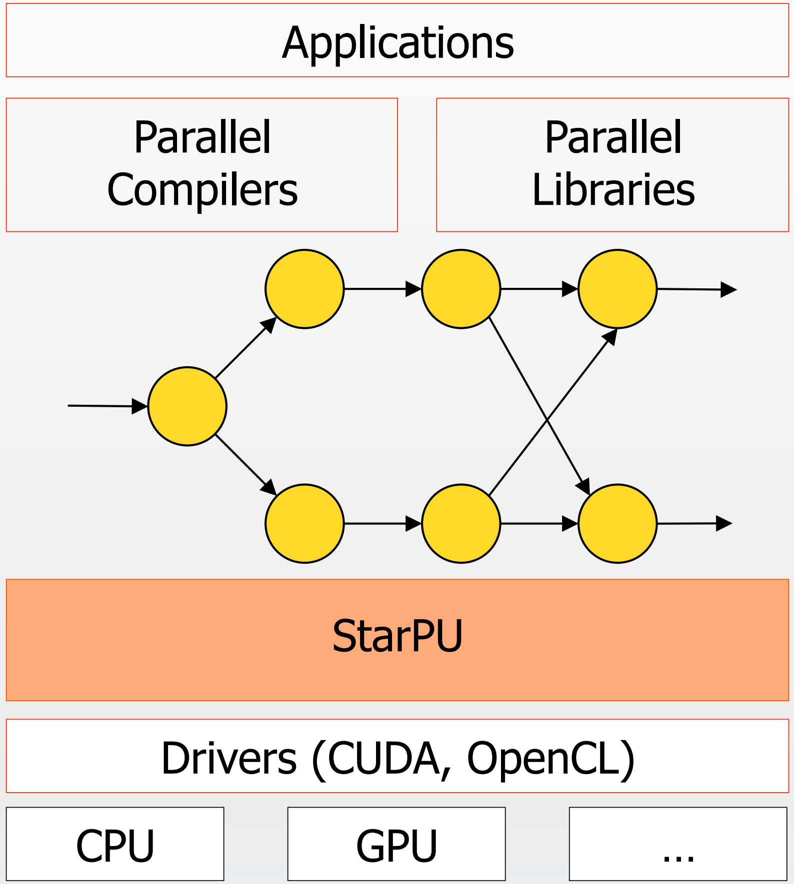
- Rational
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous processing units
  - Avoid unnecessary data transfers between accelerators
    - Software VSM for heterogeneous machines



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
      - 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

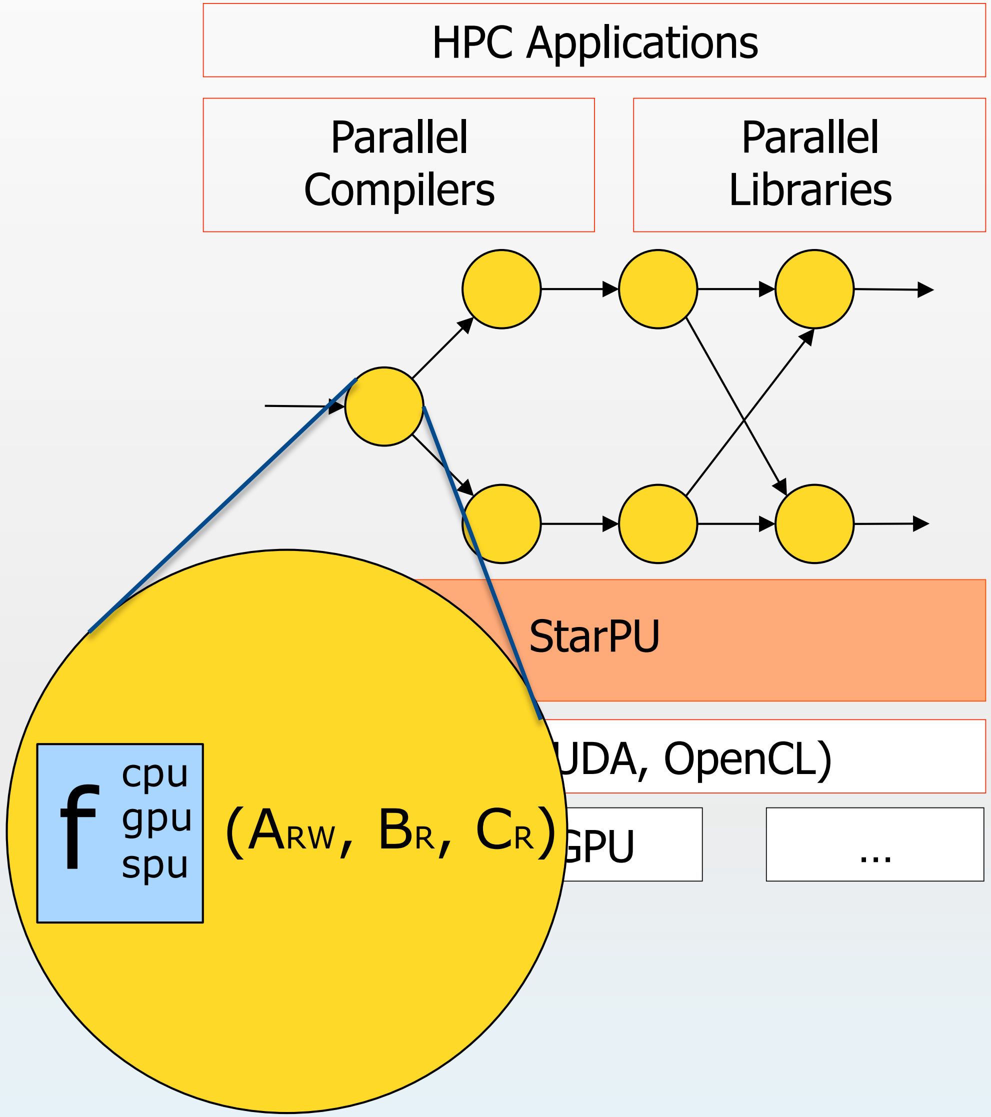


Overview of StarPU

Dealing with heterogeneous hardware accelerators

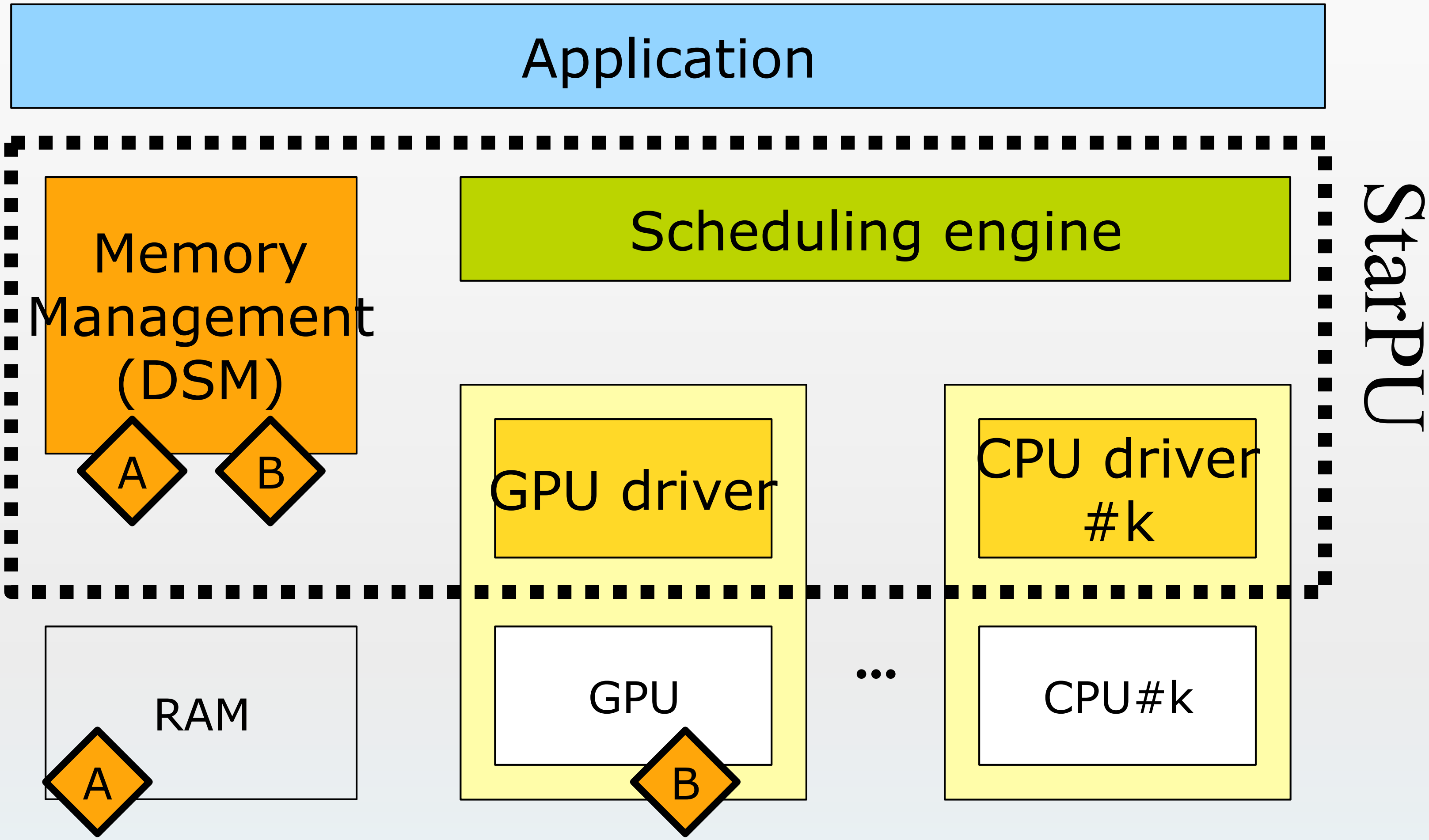
- Tasks =
  - Data input & output
  - Dependencies with other tasks
  - Multiple implementations
    - E.g. CUDA + CPU implementation
  - Scheduling hints

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



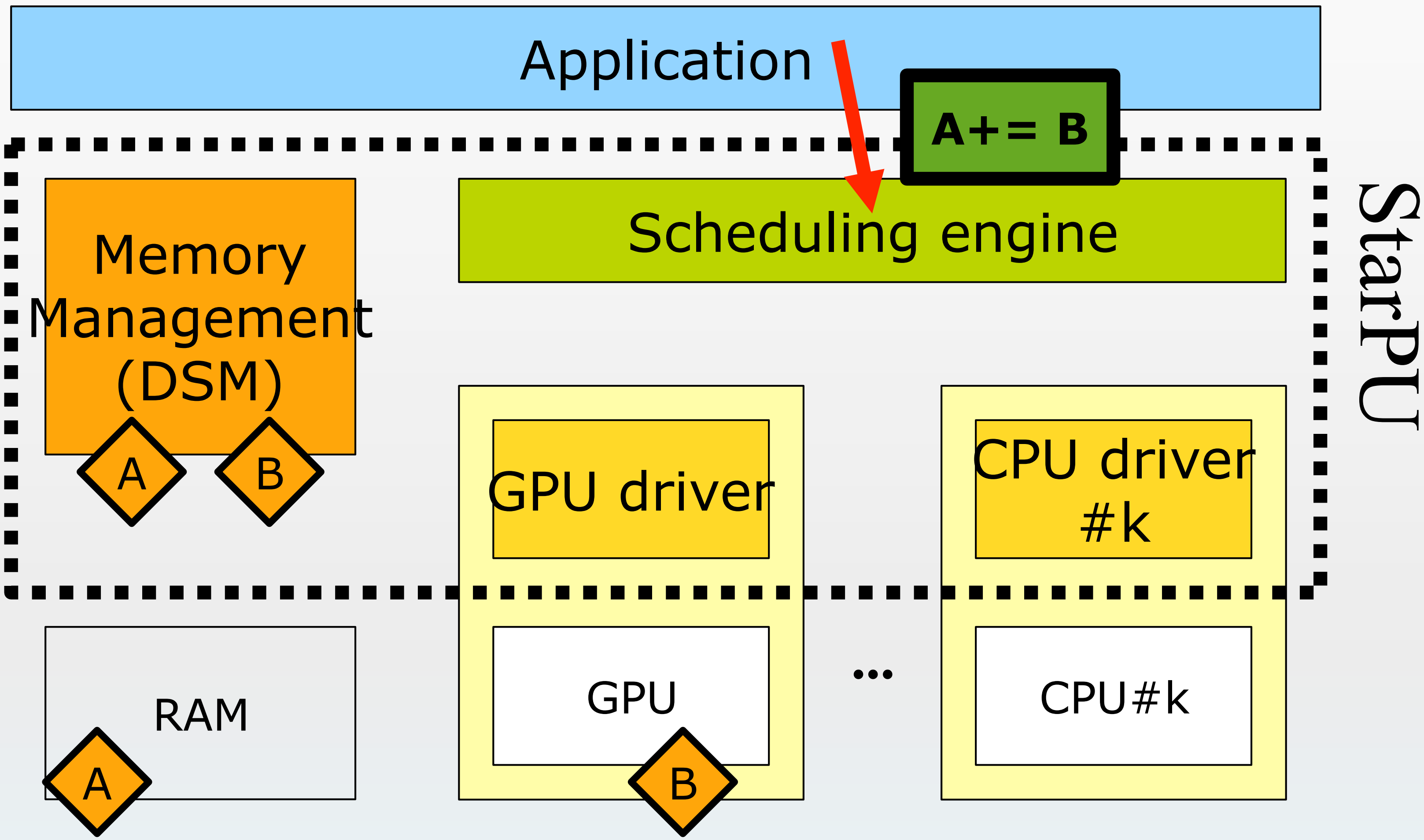
Overview of StarPU

Execution model



Overview of StarPU

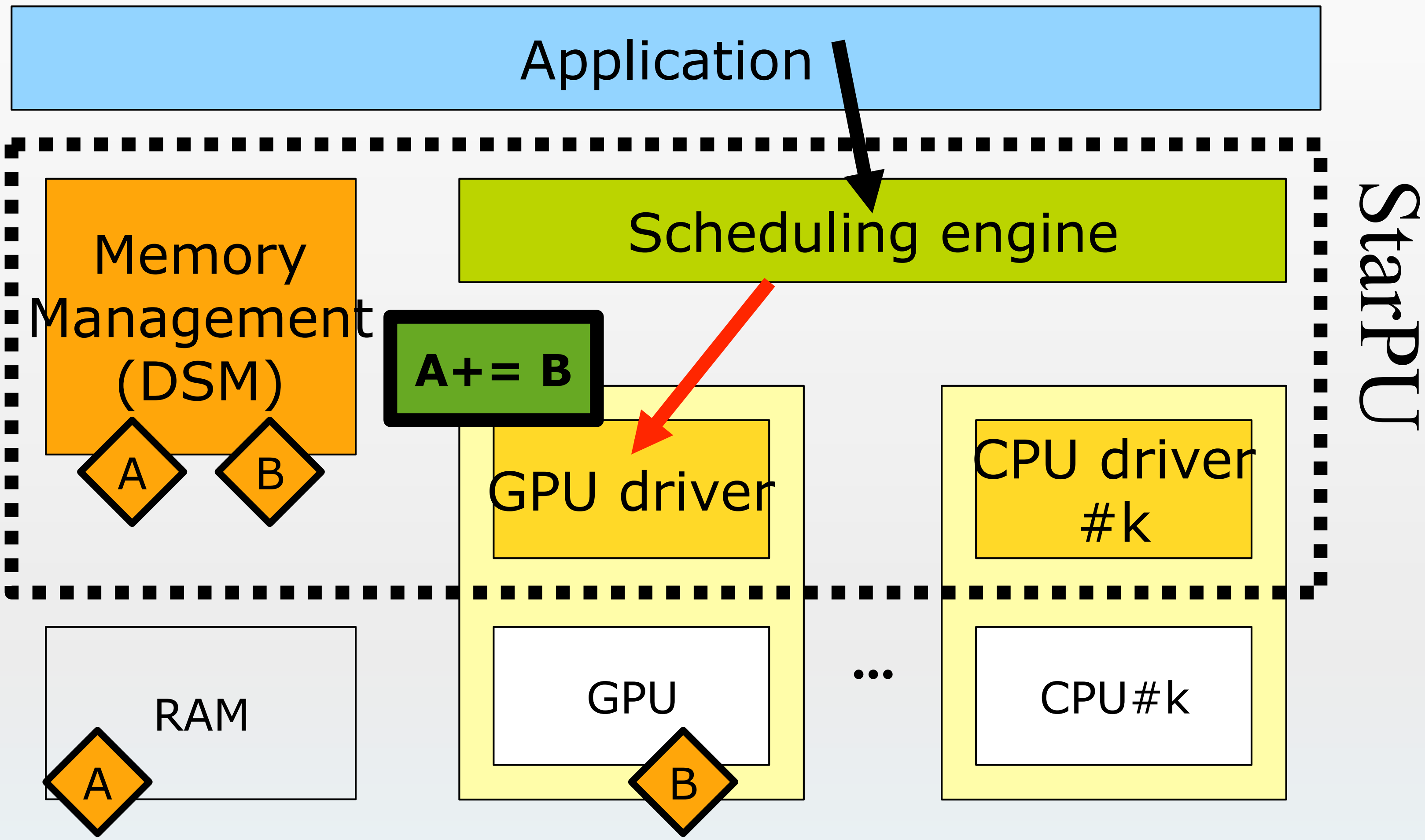
Execution model



Submit task « A += B »

Overview of StarPU

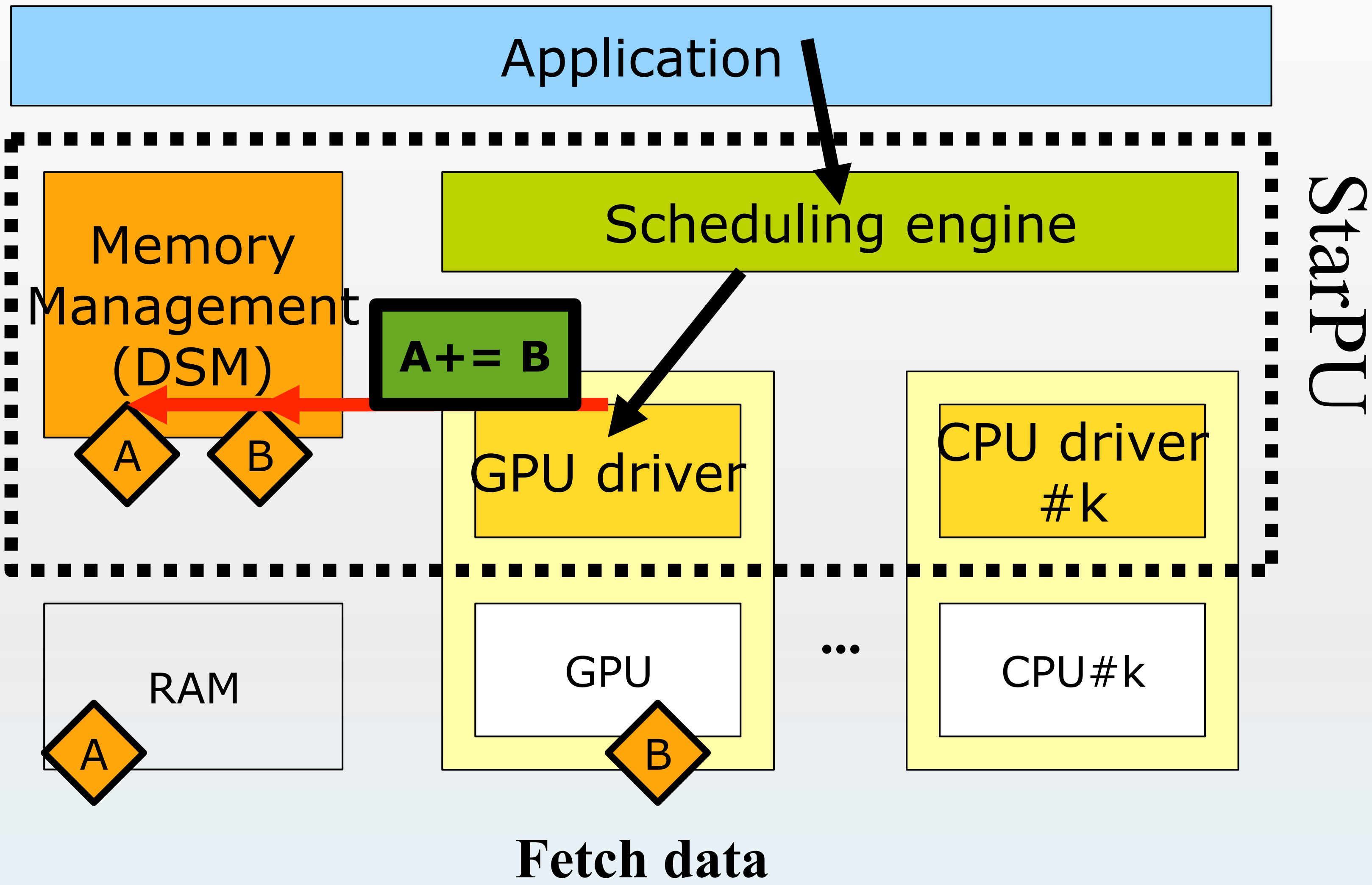
Execution model



Schedule task

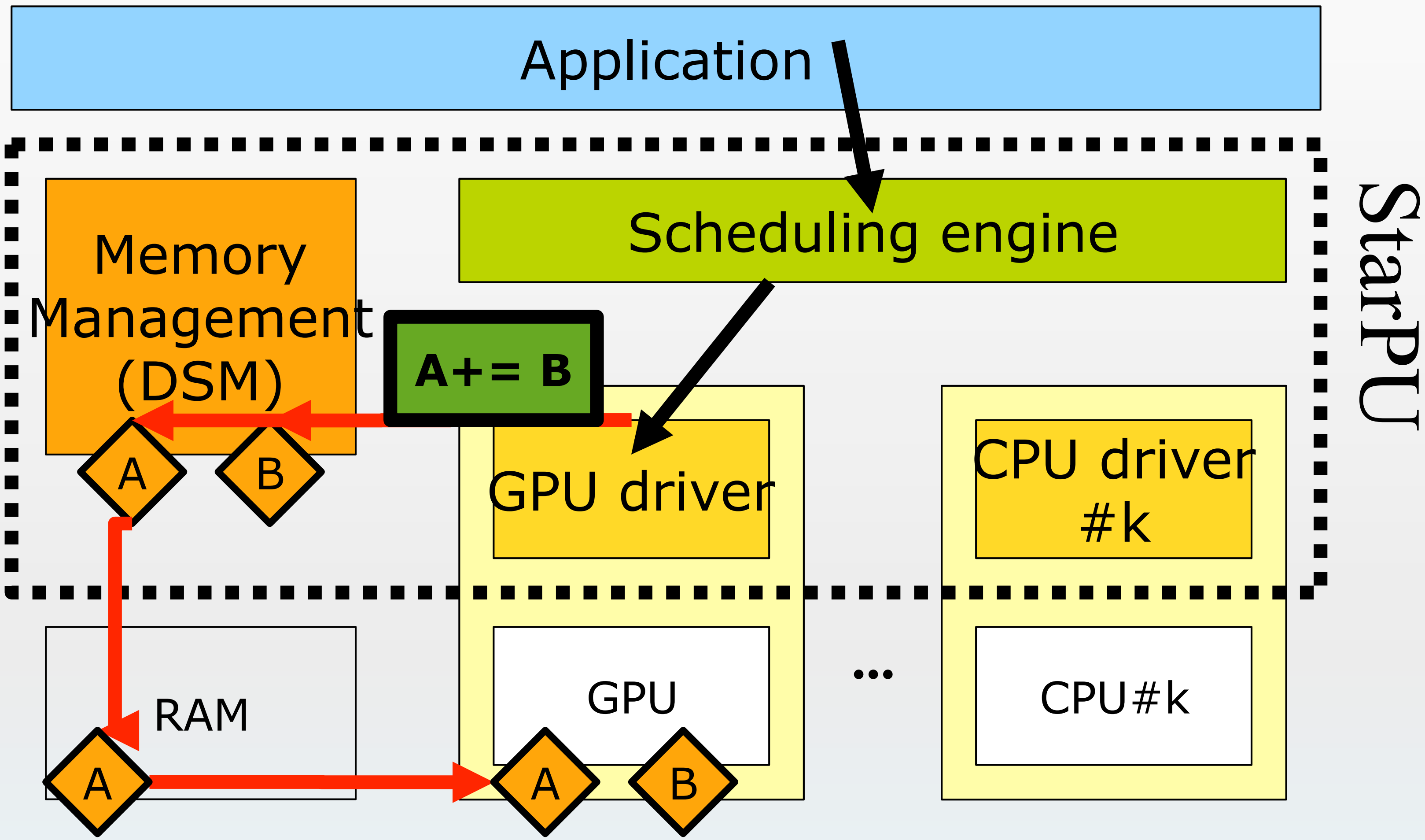
Overview of StarPU

Execution model



Overview of StarPU

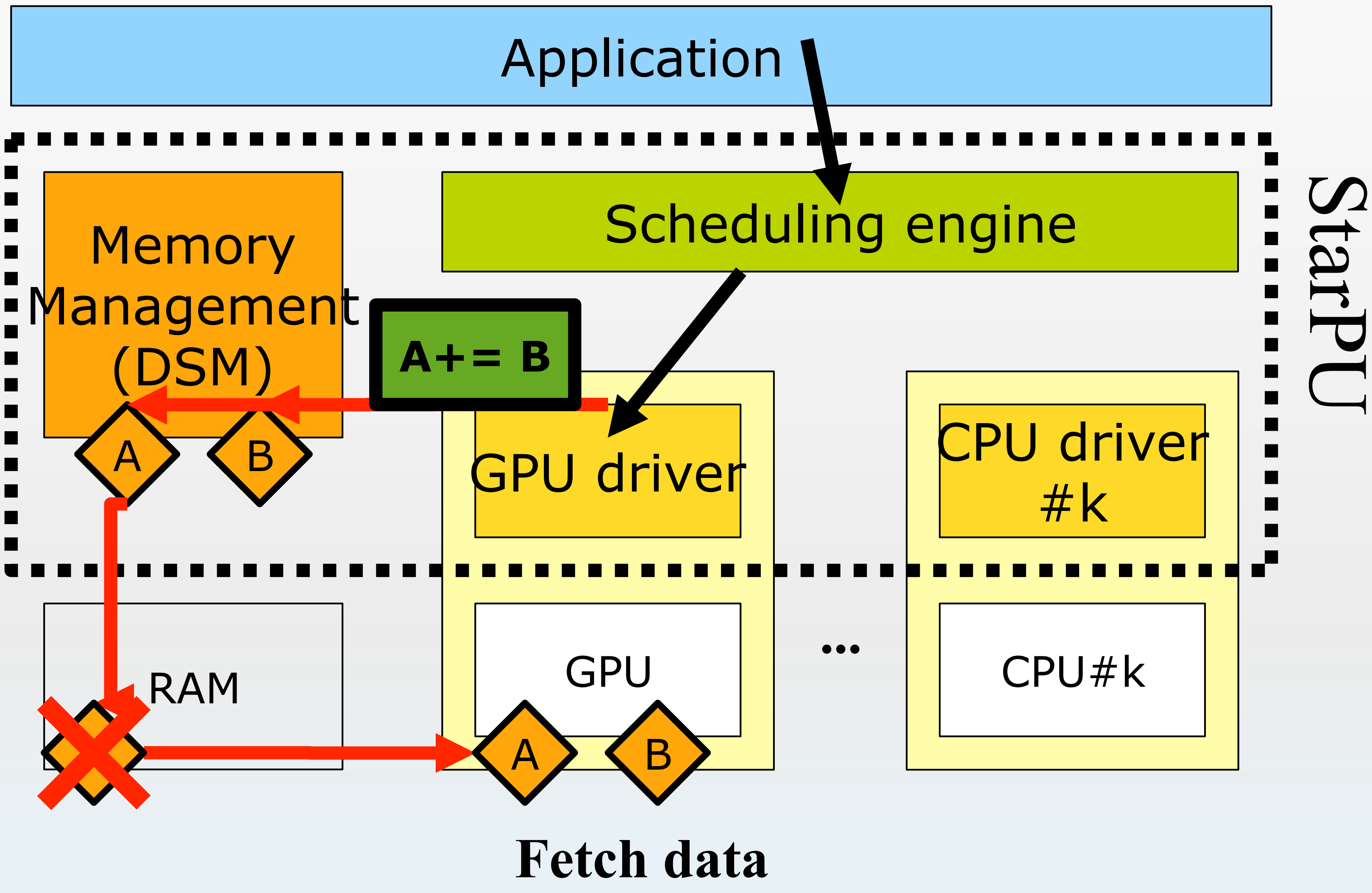
Execution model



Fetch data

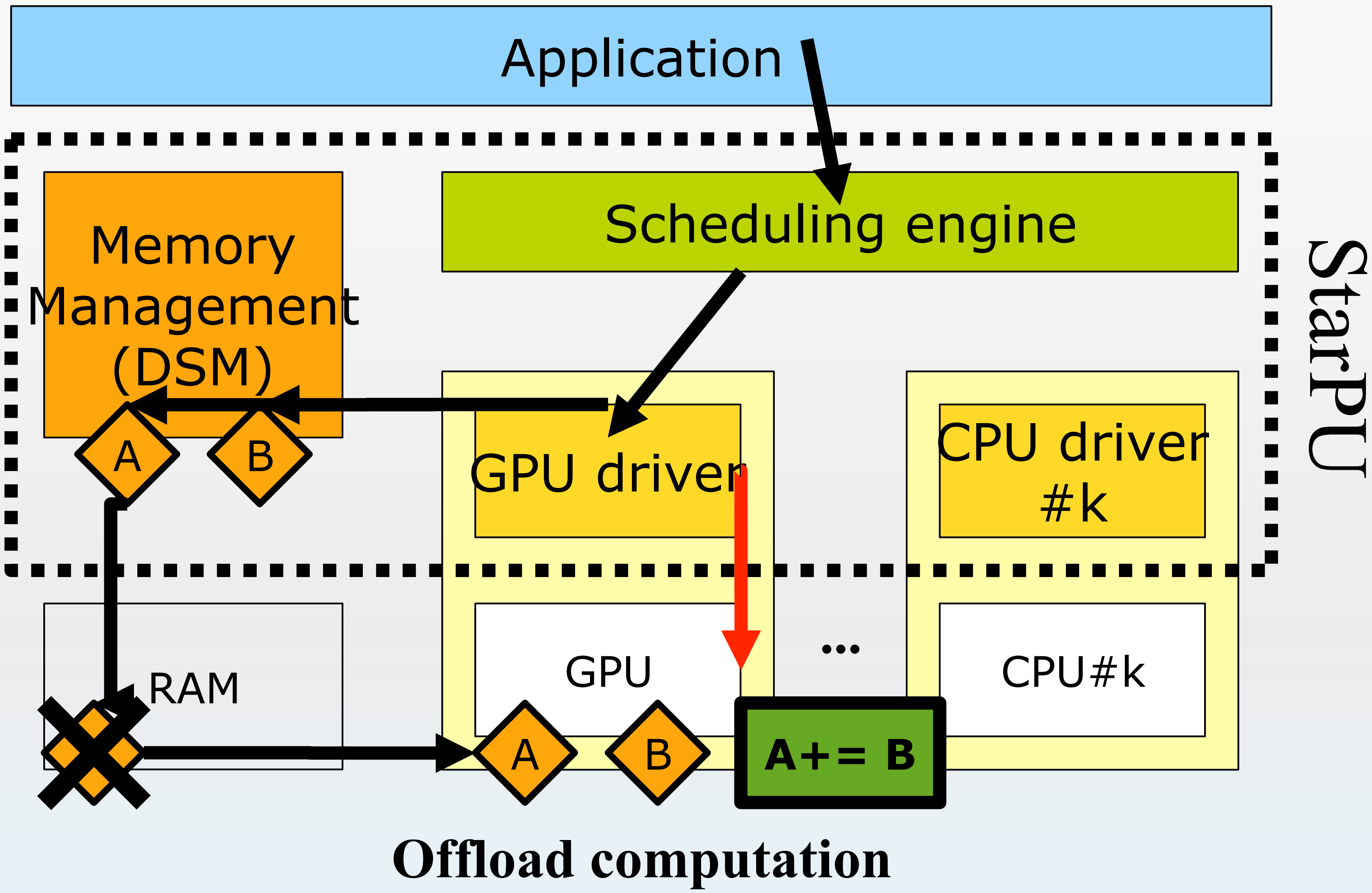
Overview of StarPU

Execution model



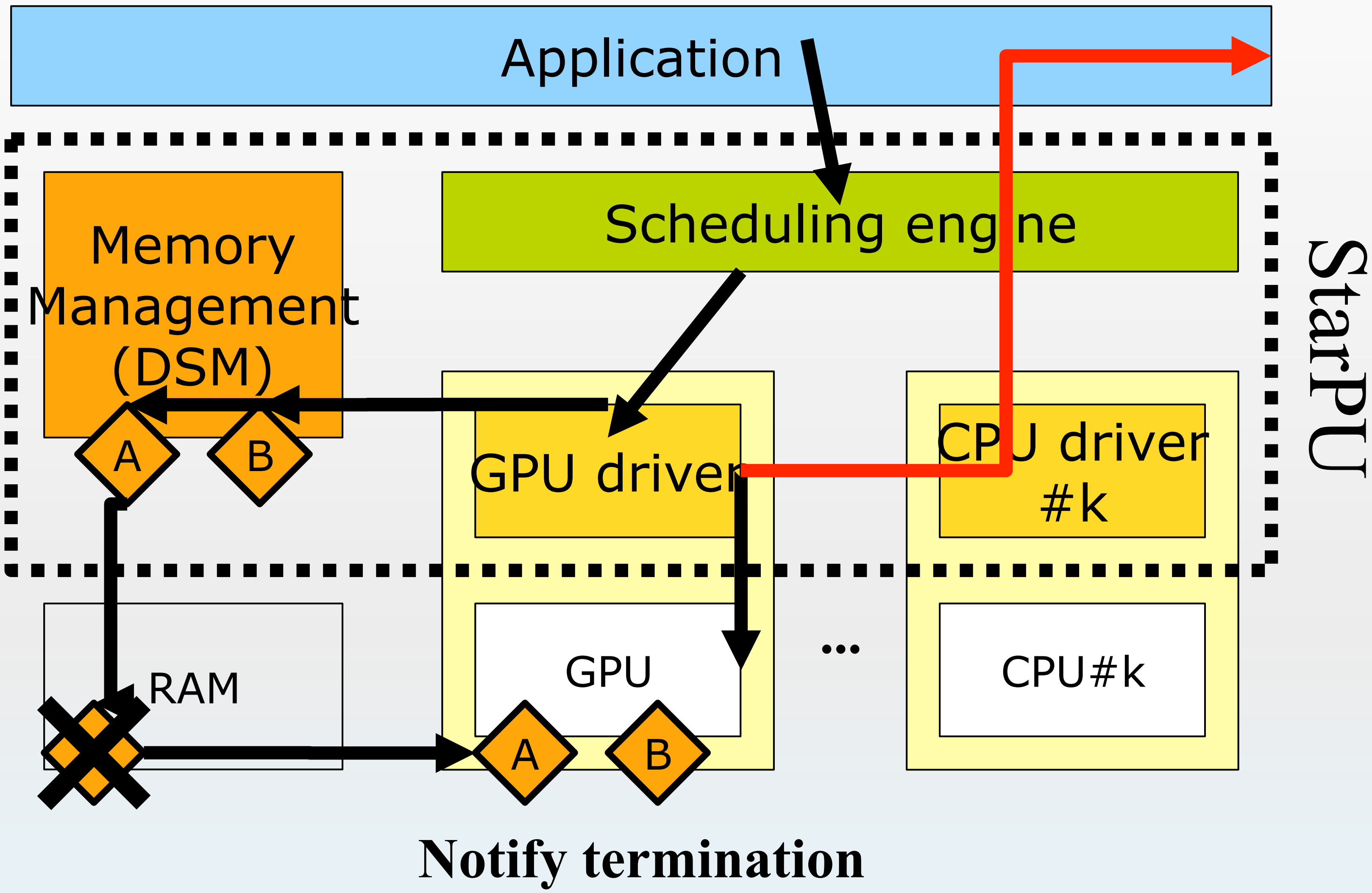
Overview of StarPU

Execution model



Overview of StarPU

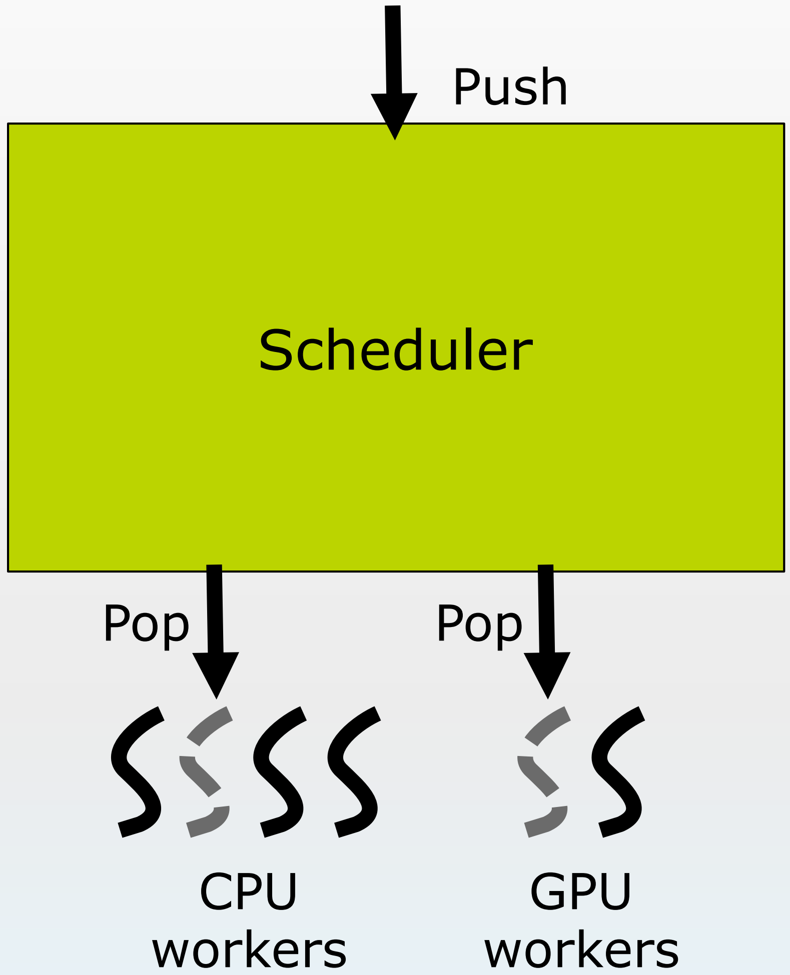
Execution model



Tasks scheduling

How does it work?

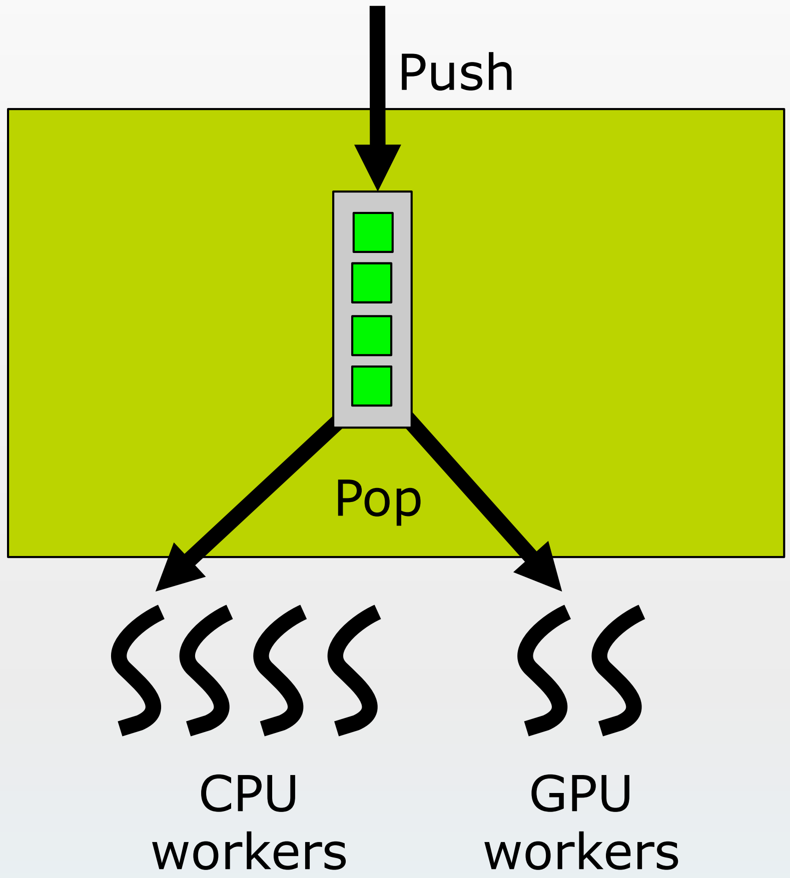
- When a task is submitted, it first goes into a pool of "frozen tasks" until all dependencies are met
- Then, the task is "pushed" to the scheduler
- Idle processing units actively poll for work ("pop")
- What happens inside the scheduler is… up to you!



Tasks scheduling

Developing your own scheduler

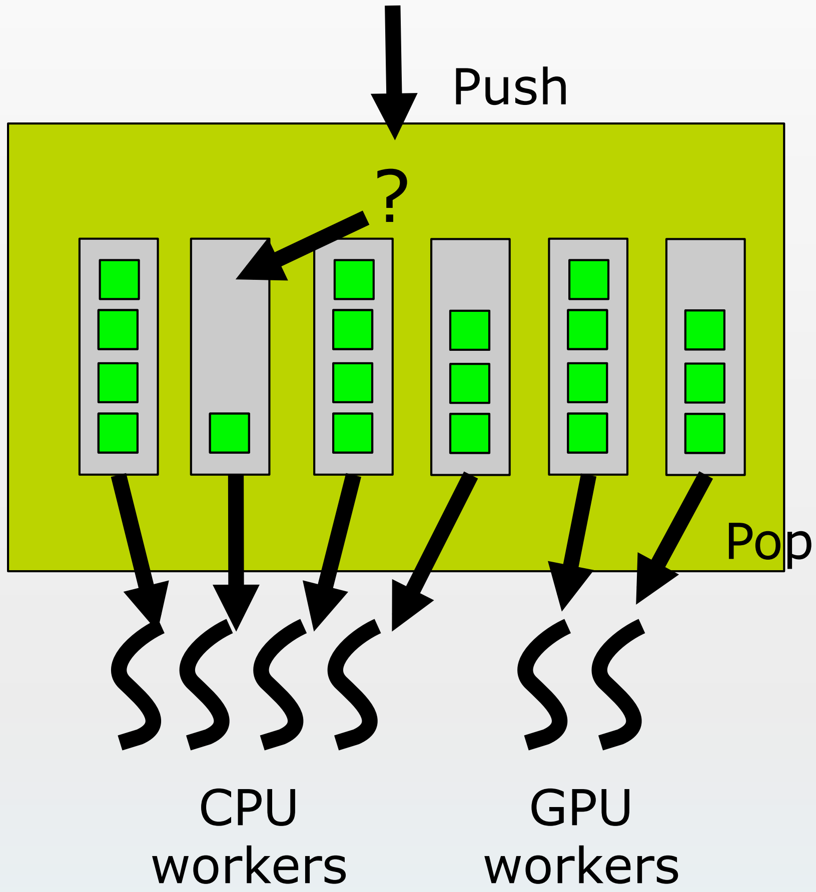
- Queue based scheduler
  - Each worker « pops » task in a specific queue
- Implementing a strategy
  - Easy!
  - Select queue topology
  - Implement « pop » and « push »
    - Priority tasks
    - Work stealing
    - Performance models, ...
- Scheduling algorithms testbed



Tasks scheduling

Developing your own scheduler

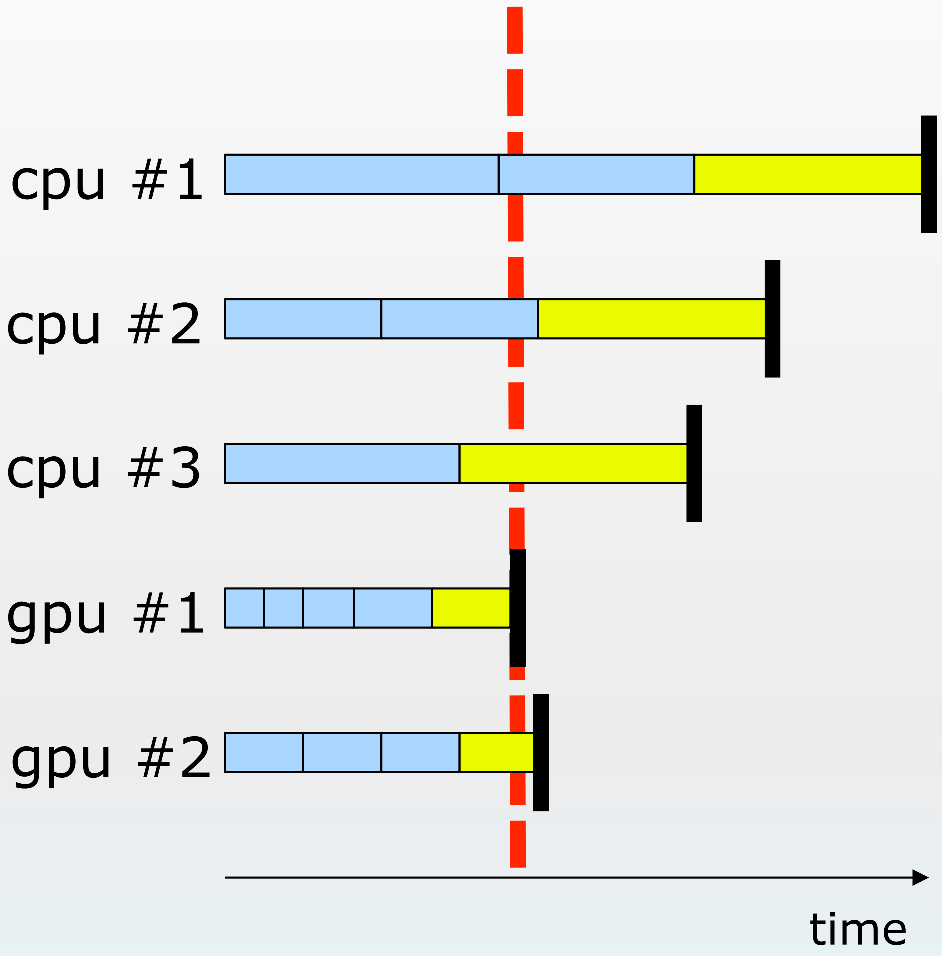
- Queue based scheduler
  - Each worker « pops » task in a specific queue
- Implementing a strategy
  - Easy!
  - Select queue topology
  - Implement « pop » and « push »
    - Priority tasks
    - Work stealing
    - Performance models, …
- Scheduling algorithms testbed



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

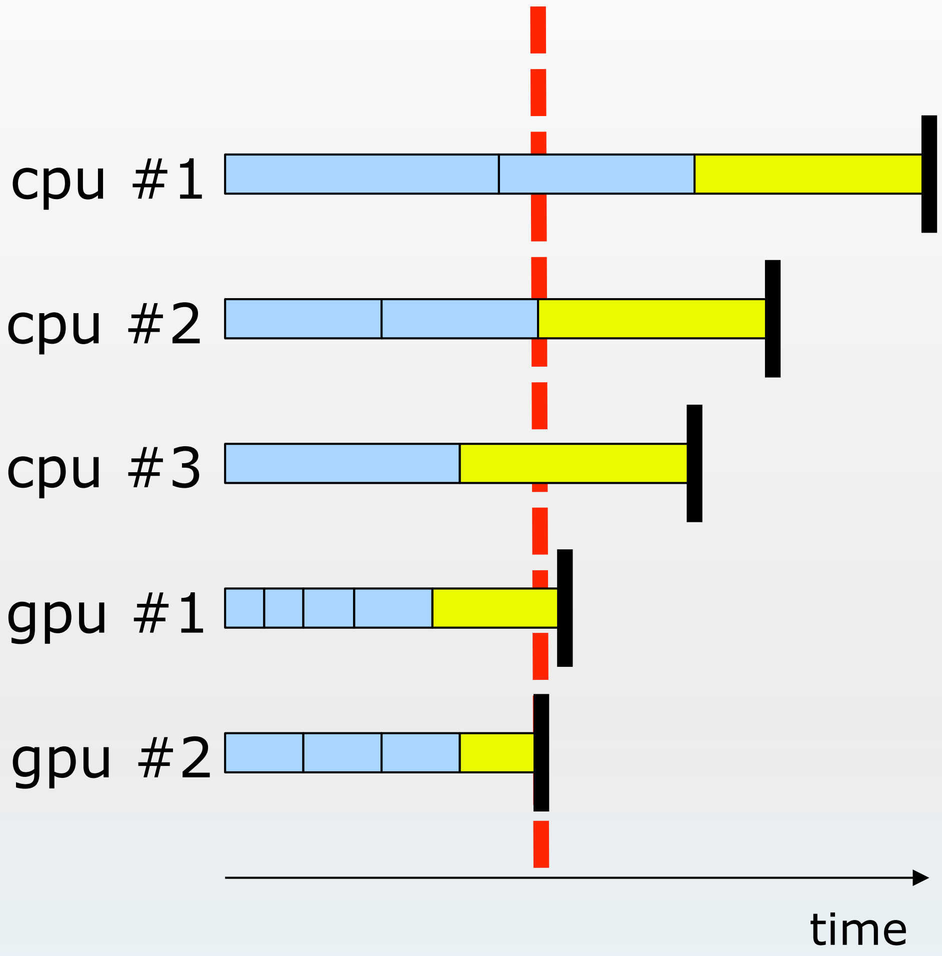


Dealing with heterogeneous architectures

Performance prediction

- Data transfer time estimation
  - Sampling based on off-line calibration

- Can be used to
  - Better estimate overall exec time
  - Minimize data movements



StarPU’s Programming Interface

Scaling vector example

Scaling a vector

Data registration

- Register a piece of data to StarPU

```
float array[NX];
for (unsigned i = 0; i < NX; i++)
    array[i] = 1.0f;

starpu_data_handle vector_handle;
starpu_vector_data_register(&vector_handle, 0,
        array, NX, sizeof(vector[0]));
```

- Unregister data

```
starpu_data_unregister(vector_handle);
```

Scaling a vector

Defining a codelet

- CPU kernel

```
void scal_cpu_func(void *buffers[], void *cl_arg)
{
    struct starpu_vector_interface_s *vector = buffers[0];

    unsigned n = STARPU_VECTOR_GET_NX(vector);
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);

    float *factor = cl_arg;

    for (int i = 0; i < n; i++)
        val[i] *= *factor;
}
```

Scaling a vector

Defining a codelet (2)

- CUDA kernel (compiled with nvcc, in a separate .cu file)

```
__global__ void vector_mult_cuda(float *val, unsigned n, float factor)
{
        for(unsigned i = 0 ; i < n ; i++) val[i] *= factor;
}

extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
{
        struct starpu_vector_interface_s *vector = buffers[0];
        unsigned n = STARPU_VECTOR_GET_NX(vector);
        float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
        float *factor = (float *)cl_arg;

        vector_mult_cuda<<<1,1>>>(val, n, *factor);
        cudaThreadSynchronize();
}
```

Scaling a vector

Defining a codelet (3)

- OpenCL kernel

```
__kernel void vector_mult_opencl(__global float *val, unsigned n, float
factor) {
        for(unsigned i = 0 ; i < n ; i++) val[i] *= factor;
}

extern "C" void scal_opencl_func(void *buffers[], void *cl_arg) {
        struct starpu_vector_interface_s *vector = buffers[0];
        unsigned n = STARPU_VECTOR_GET_NX(vector);
        float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
        float *factor = (float *)cl_arg;
        ...
        clSetKernelArg(kernel, 0, sizeof(val), &val);
        ...
        clEnqueueNDRangeKernel(queue, kernel, 1, NULL, …) ;
}
```

Scaling a vector

- Defining a codelet (4)

- Codelet = multi-versioned kernel
  - Function pointers to the different kernels
  - Number of data parameters managed by StarPU

```
starpu_codelet scal_cl = {
    .where = STARPU_CPU
            | STARPU_CUDA
            | STARPU_OPENCL,
    .cpu_func = scal_cpu_func,
    .cuda_func = scal_cuda_func,
    .opencl_func = scal_opencl_func,
    .nbuffers = 1
};
```

Scaling a vector

Defining a task

- Define a task that scales the vector by a constant

```
struct starpu_task *task = starpu_task_create();
task->cl = &scal_cl;

task->buffers[0].handle = vector_handle;
task->buffers[0].mode = STARPU_RW;

float factor = 3.14;
task->cl_arg = &factor;
task->cl_arg_size = sizeof(factor);

starpu_task_submit(task);
starpu_task_wait(task);
```

Scaling a vector

Defining a task, starpu_insert_task helper

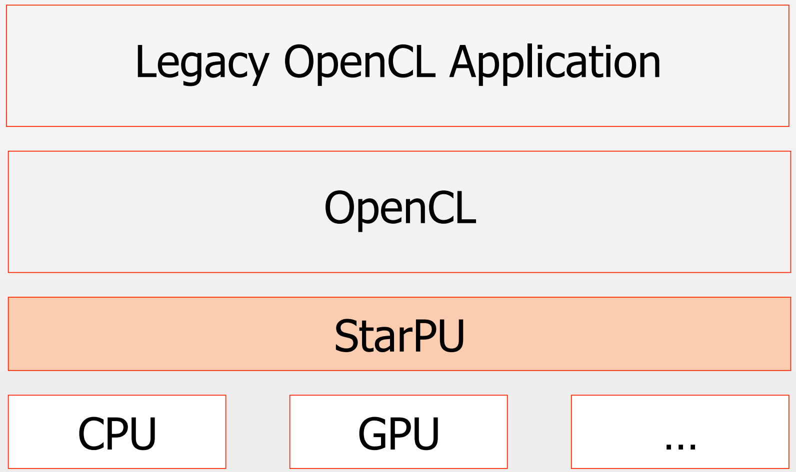
- Define a task that scales the vector by a constant

```
float factor = 3.14;

starpu_insert_task(
        &scal_cl,
        STARPU_RW, vector_handle,
        STARPU_VALUE, &factor, sizeof(factor),
        0);
```

Using StarPU through a standard API

A StarPU driver for OpenCL

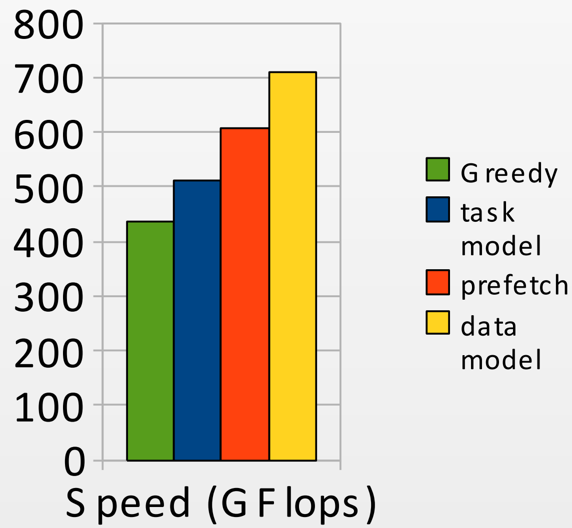
- Run legacy OpenCL codes on top of StarPU
  - OpenCL sees a number of starPU devices
- Performance limitations
  - Data transfers performed just-in-time
  - Data replication not managed by StarPU
- Ongoing work
  - We propose light extensions to OpenCL
    - Greatly improves flexibility when used
 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

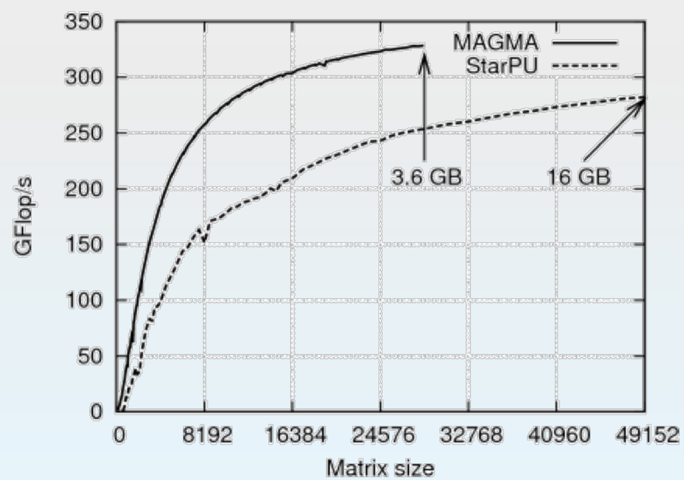


Parallel Dense Linear Algebra over StarPU

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

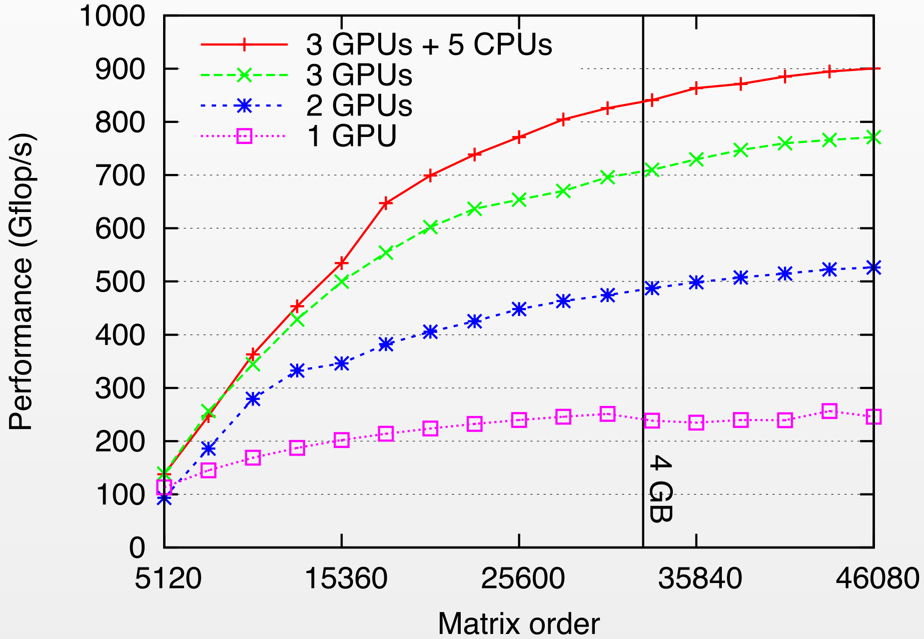


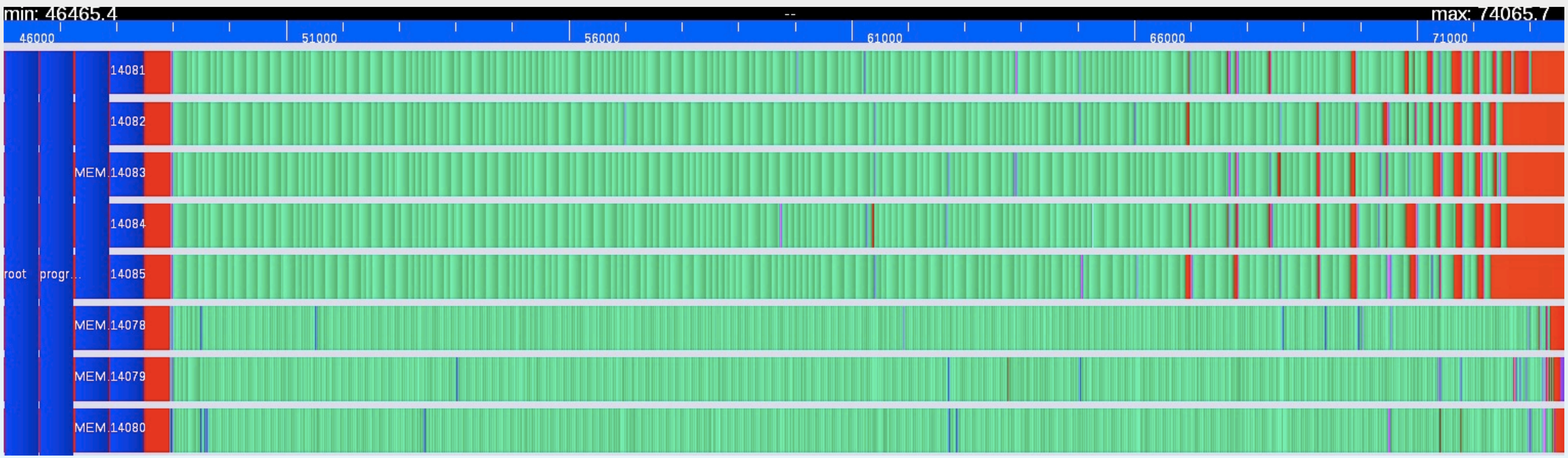


Mixing PLASMA and MAGMA with StarPU

With University of Tennessee & INRIA HiePACS

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

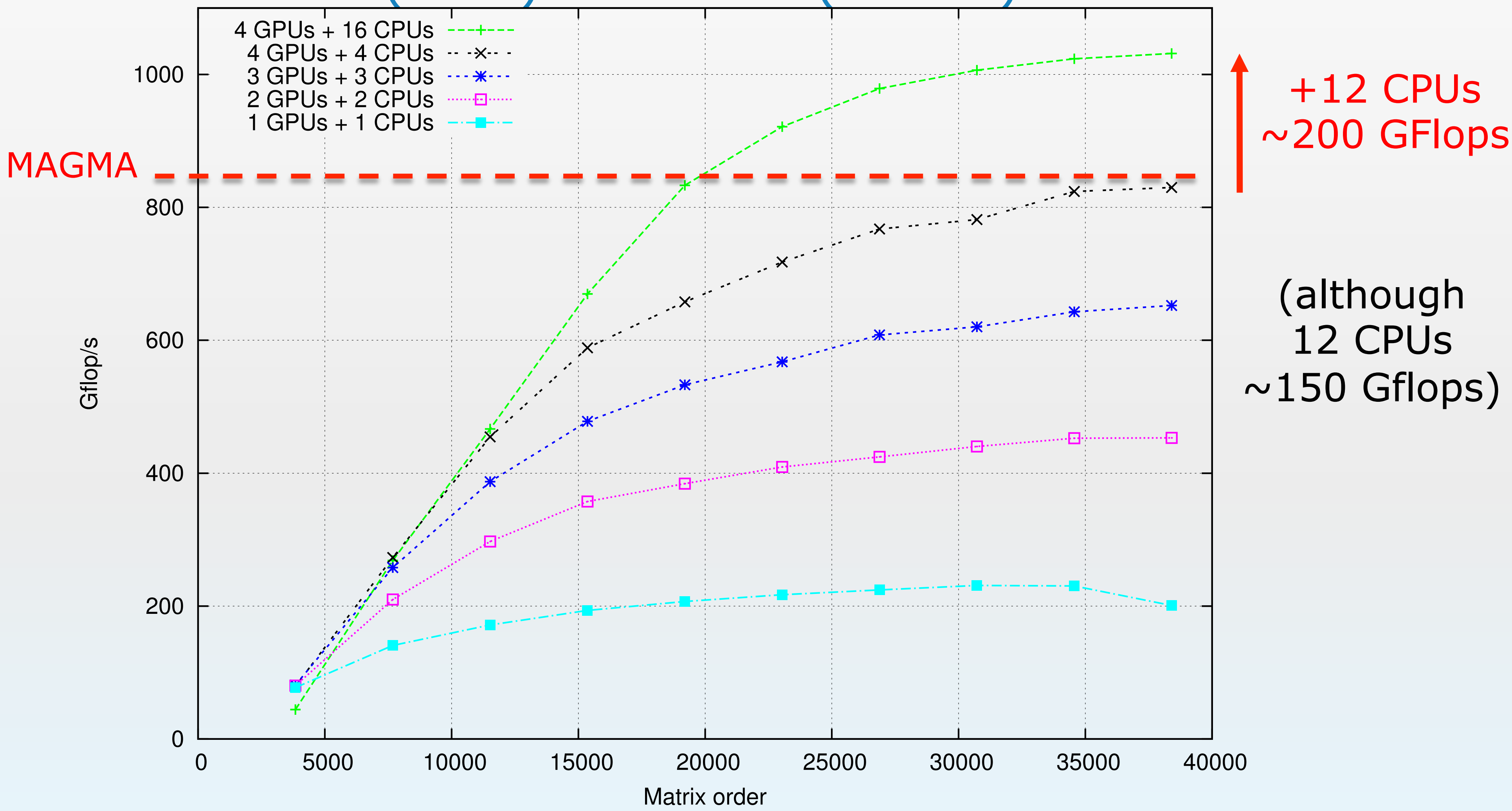




Mixing PLASMA and MAGMA with StarPU

With University of Tennessee & INRIA HiePACS

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



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 Gflops Ratio: x27
  - Sssmqr
    - CPU: 10 Gflops GPU: 285 Gflops Ratio: x28

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

- Heterogeneous architectures are cool! ☺

Using MPI and StarPU

- Keep an MPI-looking code
  - Work on StarPU data instead of plain data buffers.
- Data transfers can be partially/totally automated
  - starpu_mpi_send/recv, isend/irecv, ...
    - Equivalents of MPI_Send/Recv, Isend/Irecv,... but working on StarPU data
    - Handles all needed CPU/GPU transfers
    - Handles task/communications dependencies
    - Overlaps MPI communications, CPU/GPU communications, and CPU/GPU computations

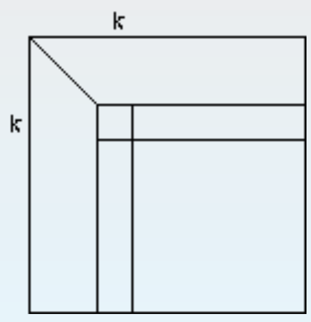
MPI version of starpu_insert_task

- Data distribution over MPI nodes decided by application
- Data consistency enforced at the cluster level
  - Automatic starpu_mpi_send/recv calls for each task
  - ≈ DSM with task-based granularity
- All nodes execute the same algorithm
  - Actual task distribution according to data being written to
    - Owner compute rule
- Sequential-looking code !

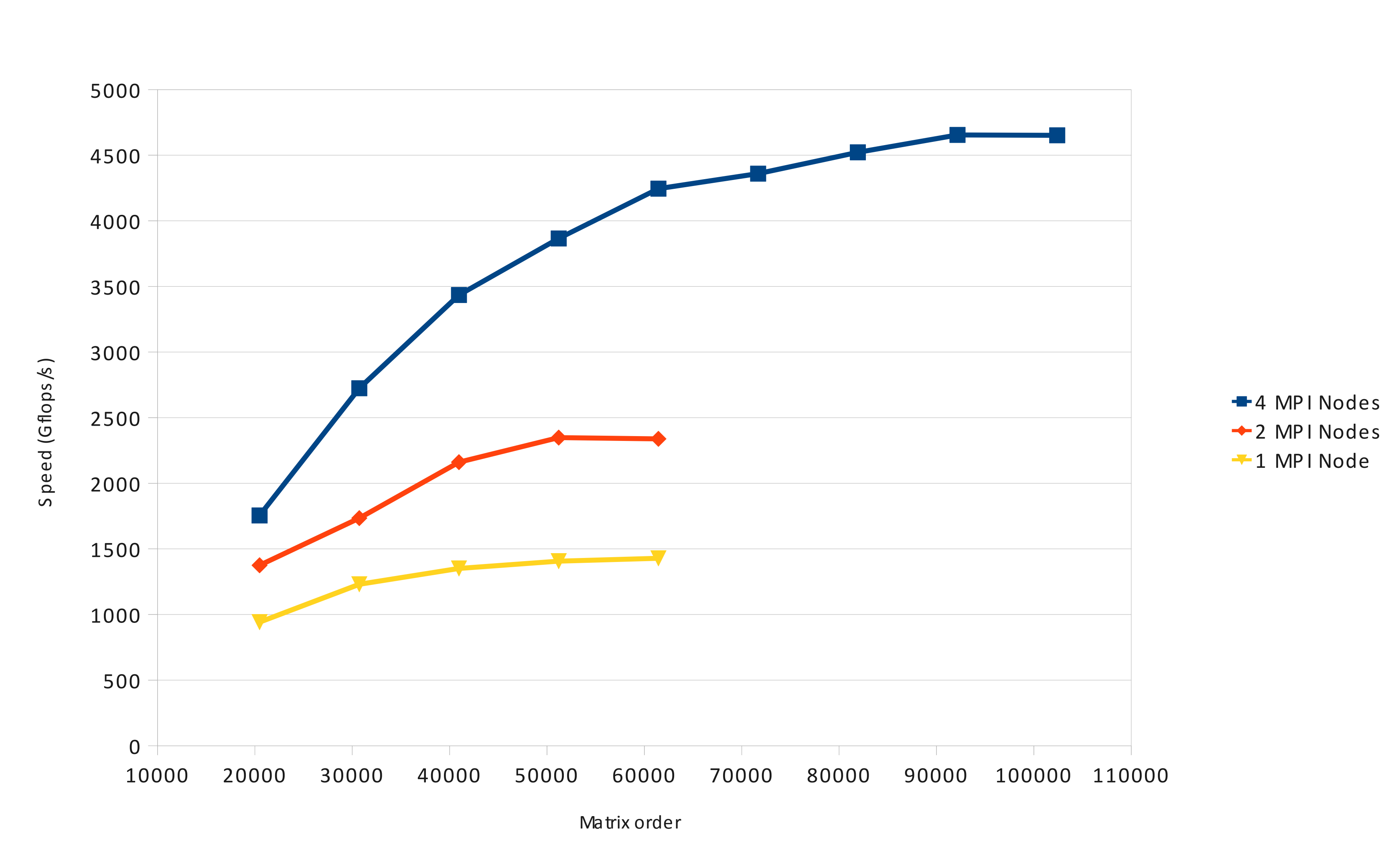
MPI version of starpu_insert_task

cholesky decomposition

```
for (k = 0; k < nblocks; k++) {
    starpu_mpi_insert_task(MPI_COMM_WORLD, &cl11,
                STARPU_RW, data_handles[k][k], 0);
    for (j = k+1; j<nblocks; j++) {
      starpu_mpi_insert_task(MPI_COMM_WORLD, &cl21,
                STARPU_R, data_handles[k][k],
                STARPU_RW, data_handles[k][j], 0);
      for (i = k+1; i<nblocks; i++)
        if (i <= j)
          starpu_mpi_insert_task(MPI_COMM_WORLD, &cl22,
                    STARPU_R, data_handles[k][i],
                    STARPU_R, data_handles[k][j],
                    STARPU_RW, data_handles[i][j], 0);
    }
  }
starpu_task_wait_for_all();
```



Cholesky Using MPI+StarPU + Magma kernels
Early results

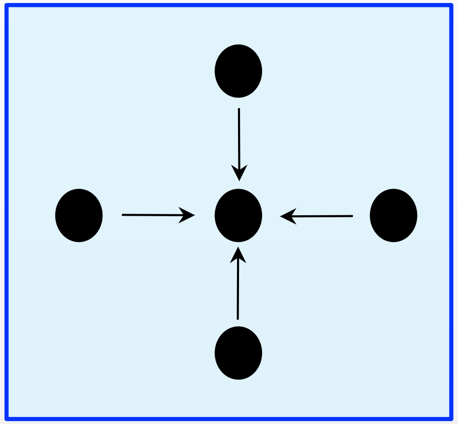


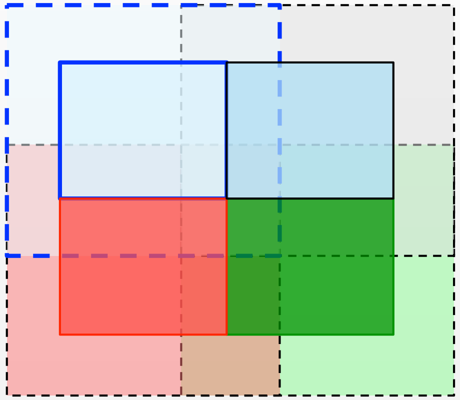
Integrating multithreading and StarPU

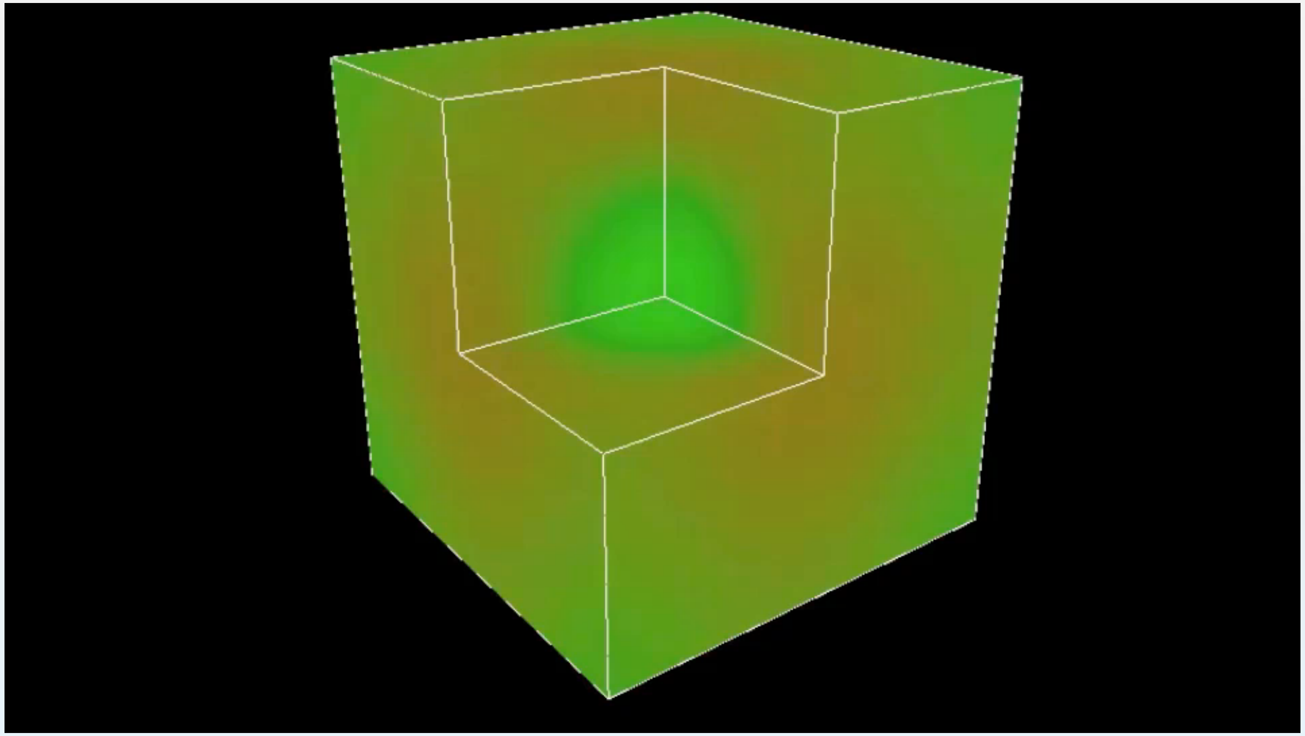
Static vs Dynamic scheduling

Stencil computation

- Wave propagation
  - Prefetching
  - Asynchronism





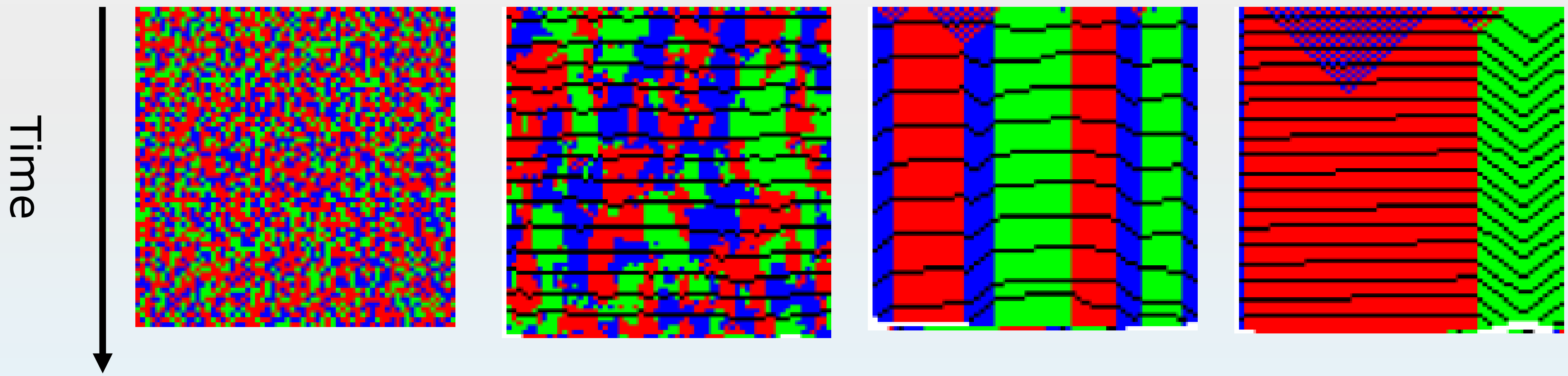


Static vs Dynamic scheduling

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

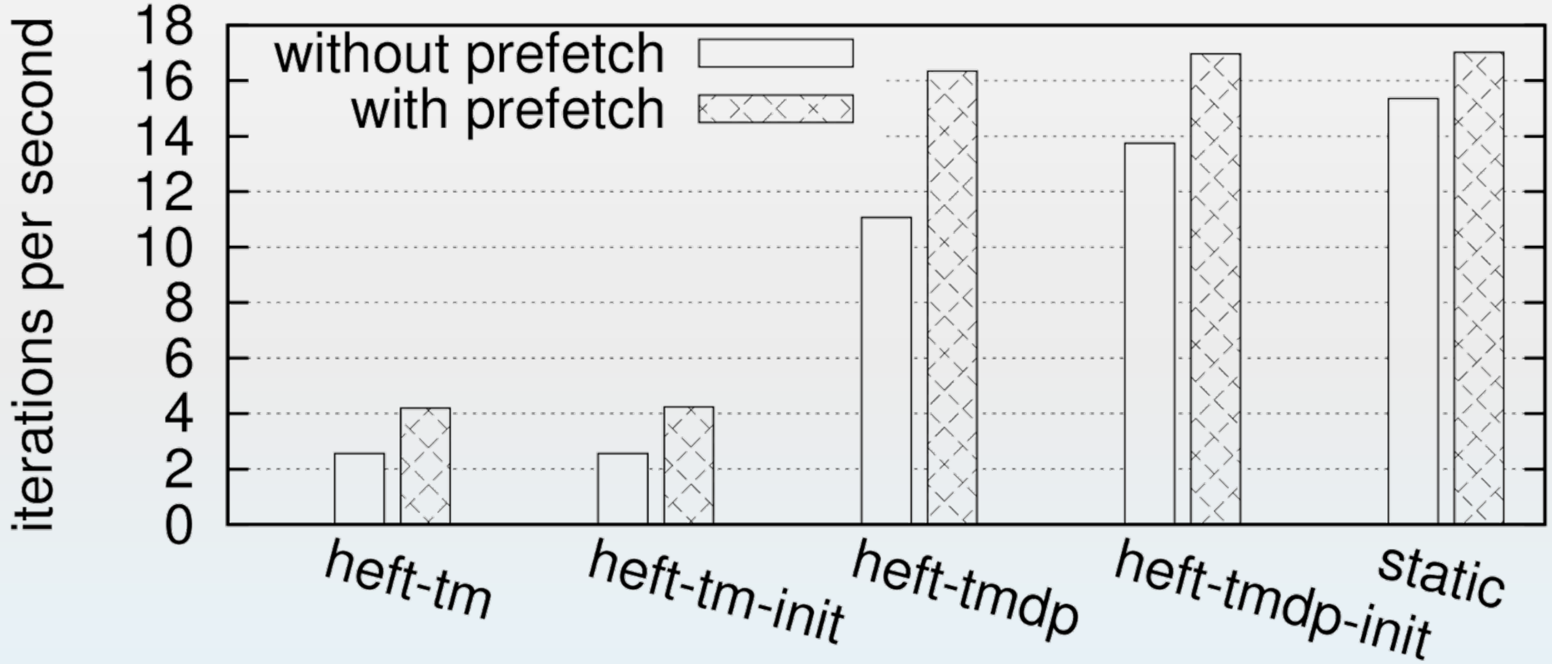


$\beta = 0$ $\beta = 0.5$ $\beta = 3$ $\beta = 6$

Static vs Dynamic scheduling

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)

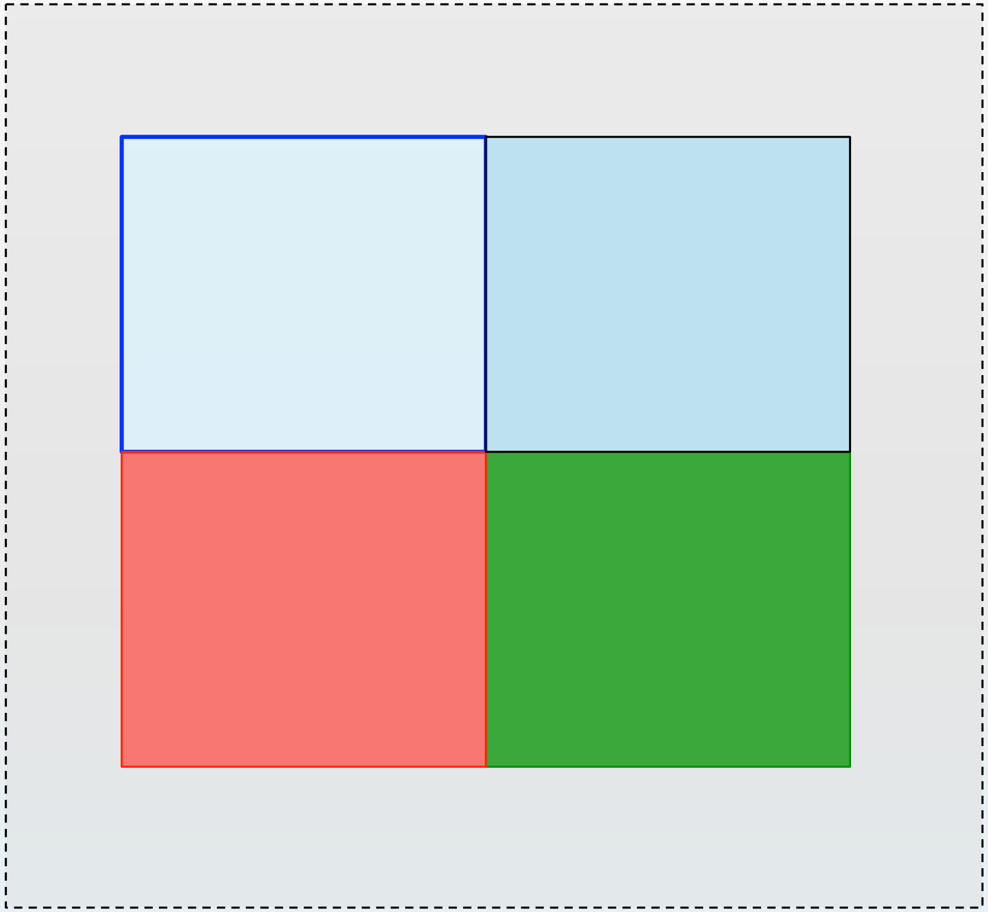


Towards parallel tasks on CPUs

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



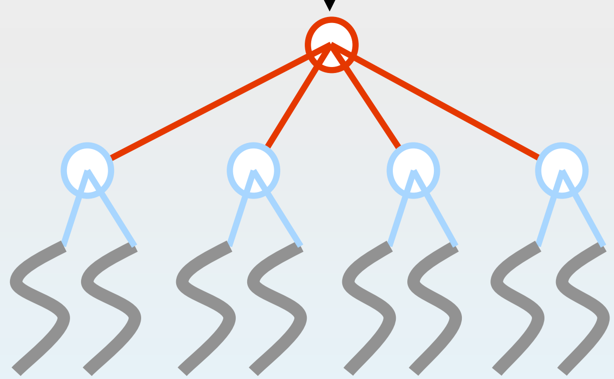
Integrating StarPU and Multithreading

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

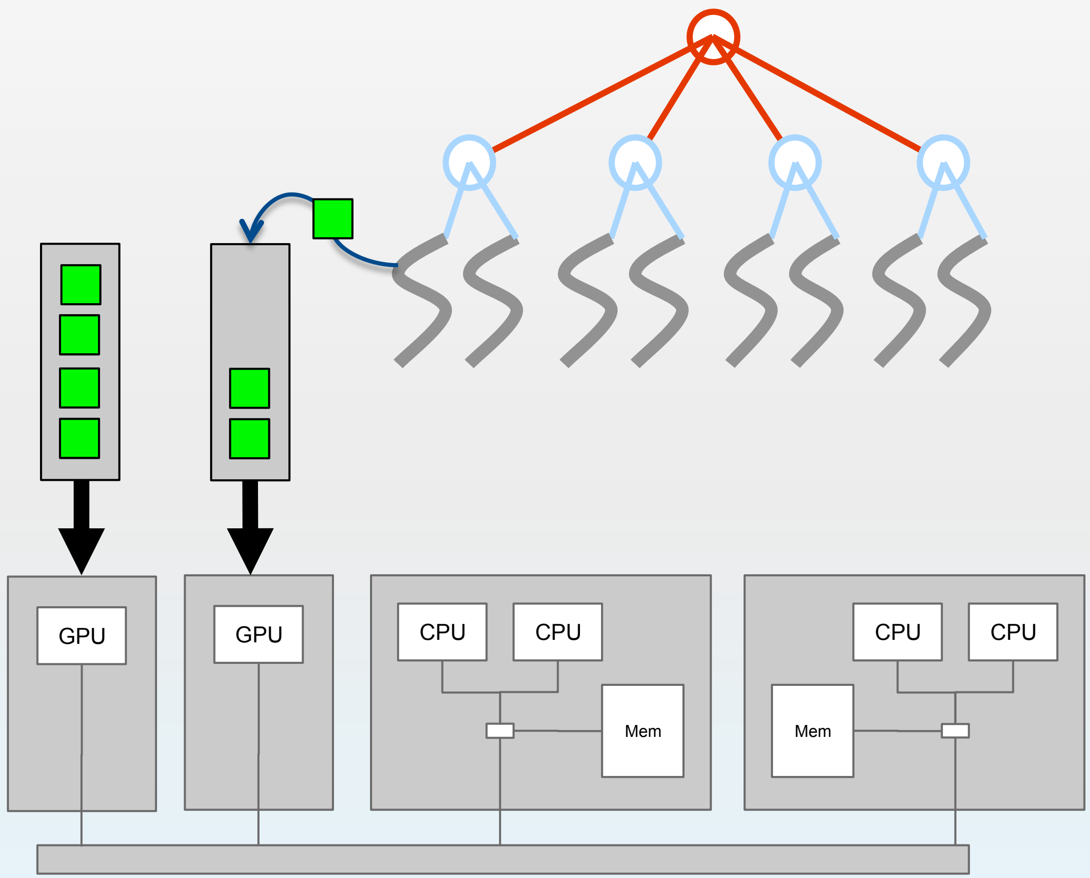
```
void work()
{
  ...

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



Integrating StarPU and Multithreading

Integrating tasks and threads

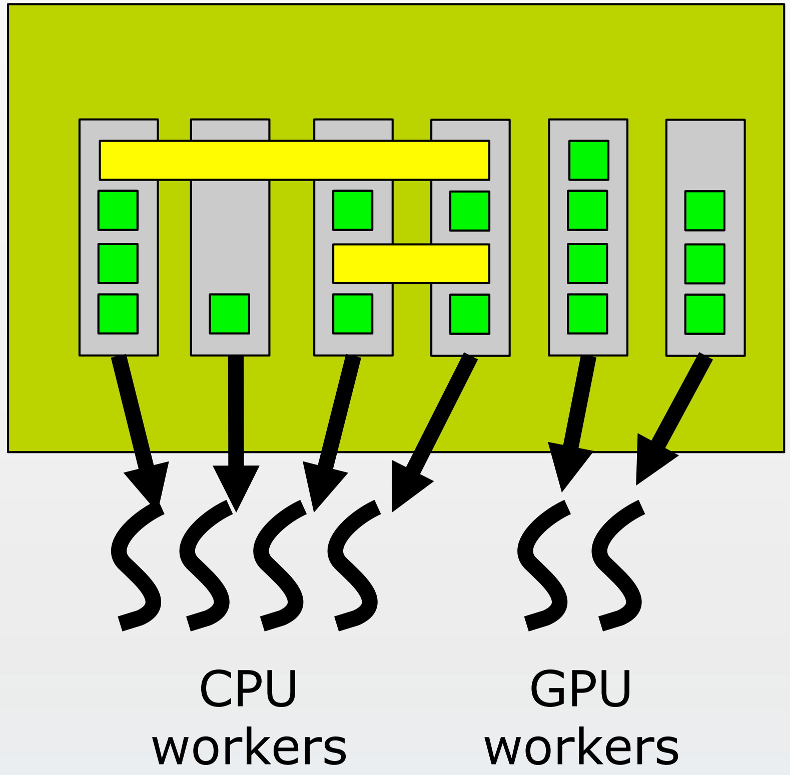
- First approach
  - Use an OpenMP main stream
    - Suggested (?) by recent parallel language extension proposals
      - E.g. Star SuperScalar (UPC Barcelona)
      - HMPP (CAPS Enterprise)
    - Implementing scheduling is difficult
      - Much more than a simple offloading approach…



Integrating StarPU and Multithreading

Integrating tasks and threads

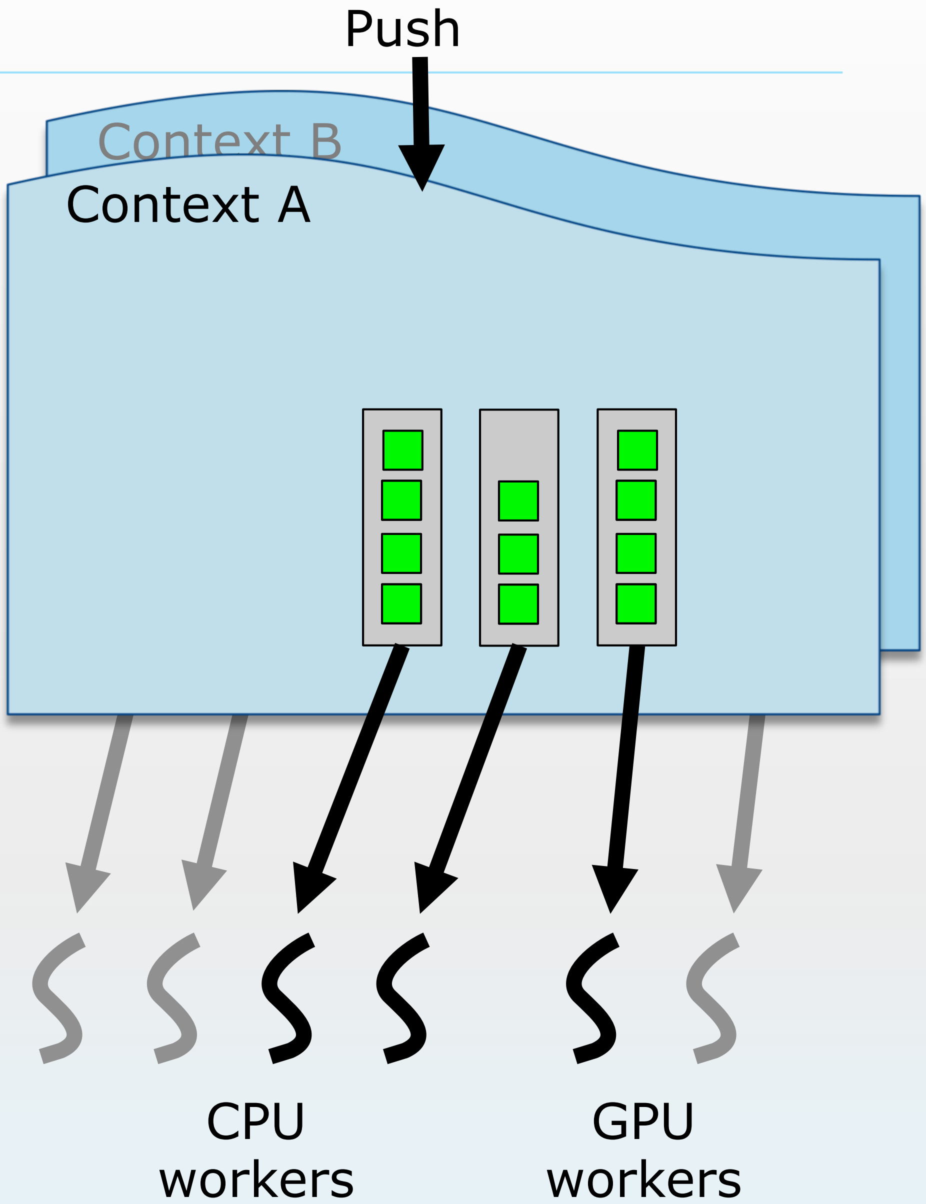
- Alternate approach
  - Let StarPU spawn OpenMP tasks
    - Performance modeling would still be valid
    - Would also work with other tools
      - E.g. Intel TBB
    - How to find the appropriate granularity?
      - May depend on the concurrent tasks!
    - StarPU tasks = first class citizen
   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



StarPU's Scheduling Contexts

Toward code

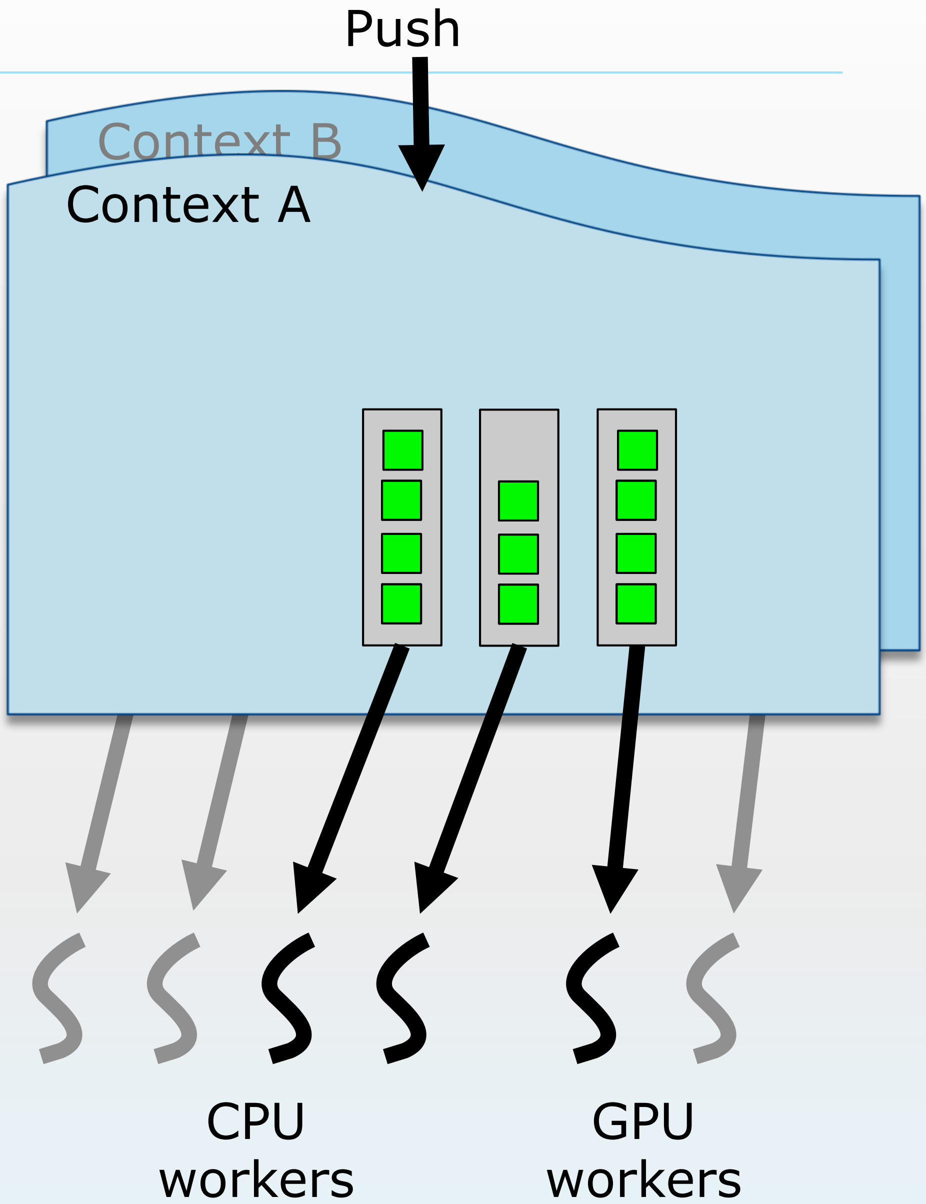
- Similar to OpenCL contexts
  - Except that each context features its own scheduler
- Multiple parallel libraries can run simultaneously
  - Virtualization of resources
    - At minimal overhead
  - Scheduling overhead reduced
  - Scalability workaround



StarPU's Scheduling Contexts

Toward code

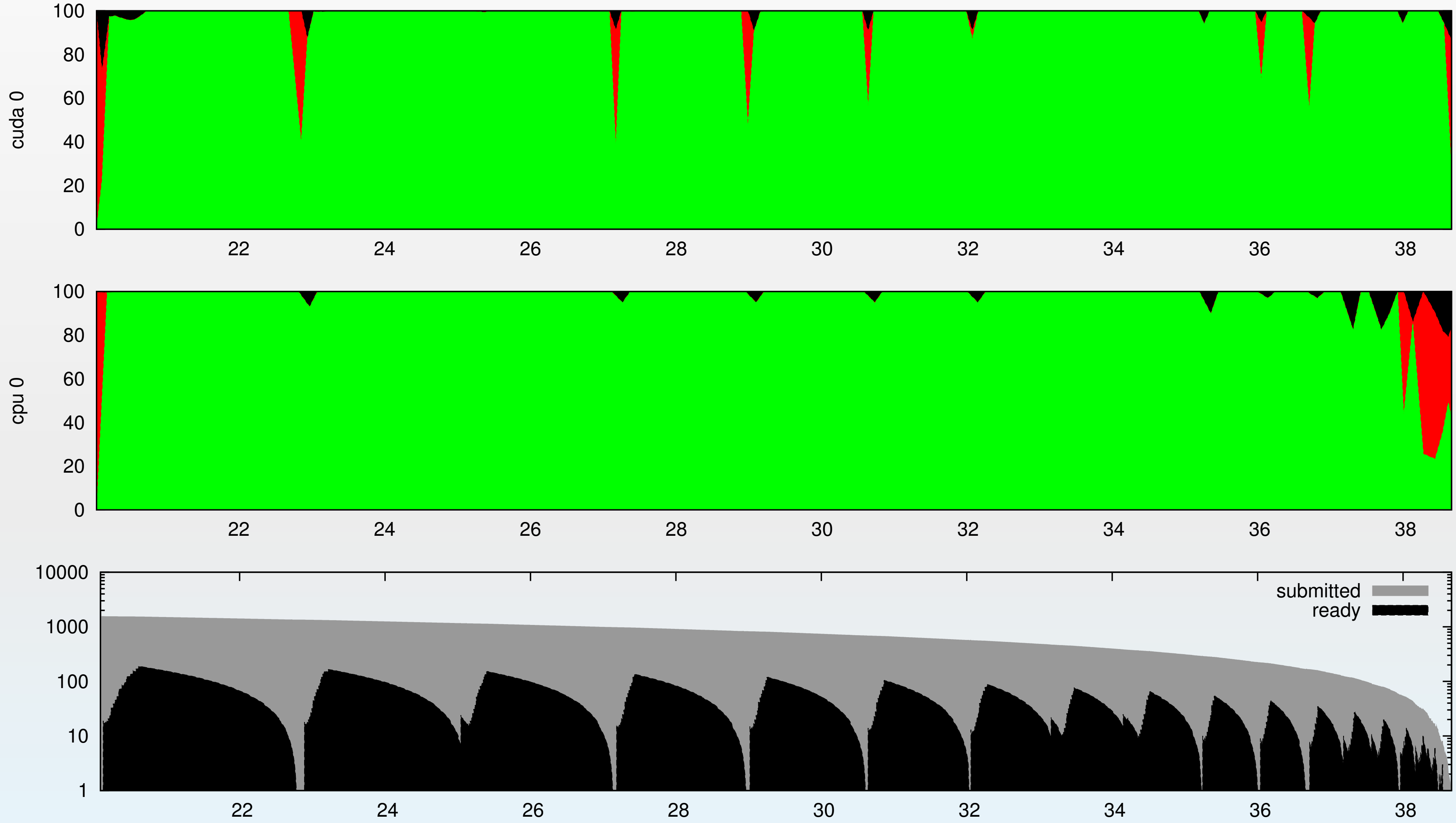
- Contexts may share processing units
  - Avoid underutilized resources
    - Schedulers are aware of each other
- Contexts may expand and shrink
  - Maximize overall throughput
  - Use dynamic feedback both from application and runtime



Integrating StarPU and Multithreading

Adapting granularity

- Real-time performance feedback



What’s next?

Future parallel machines

Exascale (10^{18} flop/s) systems, by 2018?

- 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 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

International Exascale Software Project (IESP)

"A call to action"

- Build an international plan for coordinating research for the next generation open source software for scientific high-performance computing
  - Hardware is evolving more rapidly than software
    - New hardware trends not handled by existing software
  - Emerging software technologies not yet integrated into a common software stack
  - No global evaluation of key missing components

European Exascale Software Initiative (EESI)

Position of Europe in the international HPC landscape

- WP4: Enabling technologies for Exascale computing
  - Assess novel HW and SW technologies for Exascale challenges
  - Build a European vision and a roadmap

- WG 4.2: Software eco-systems
  - Subtopic: Runtime systems (Raymond Namyst, Jesús Labarta)

European Exascale Software Initiative (EESI)

Runtime systems: Scientific and Technical Hurdles

- Mastering heterogeneity
  - Unified/transparent accelerator models
  - Providing support for adaptive granularity
  - Fine grain parallelism
  - Scheduling for latency/bandwidth
  - (NC)-NUMA
- Supporting multiple programming models
  - Hybrid runtimes
  - MPI + threading model + accelerator model
  - Matching hybrid parallelism on heterogeneous architectures
  - Tuning the number of (processes/threads) per level

European Exascale Software Initiative (EESI)

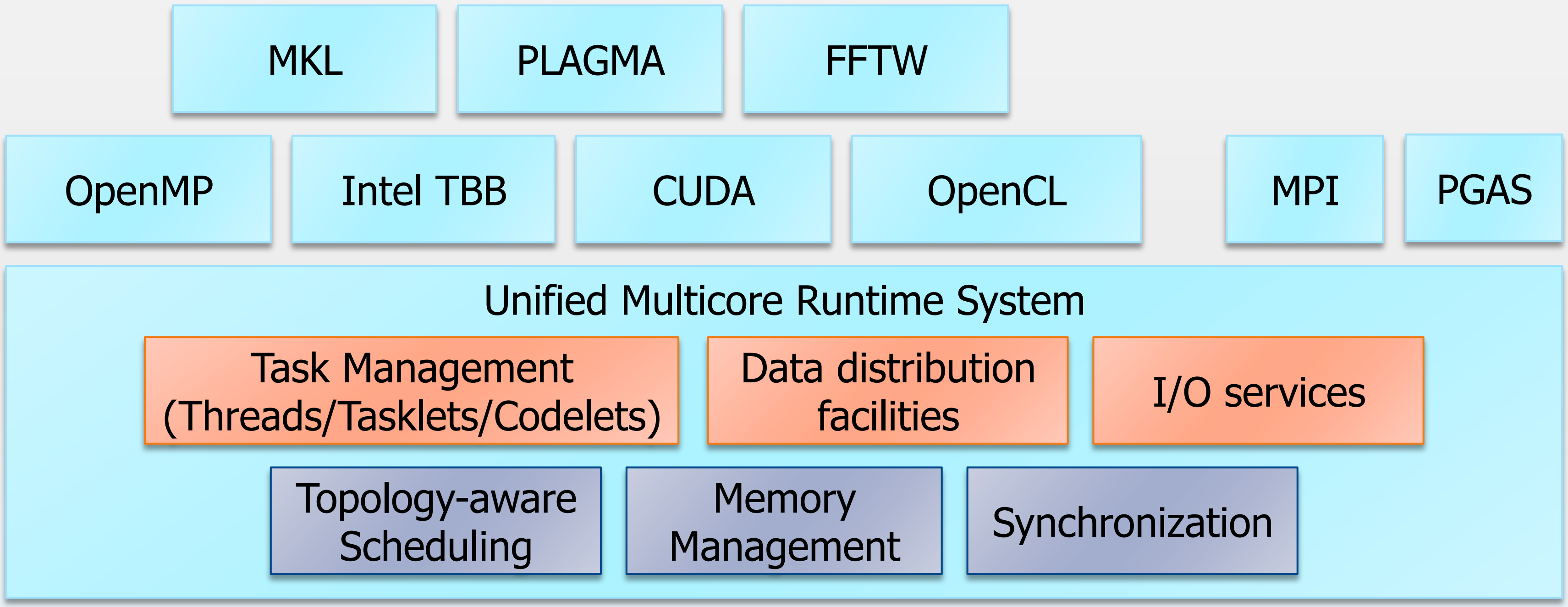
Runtime systems: Scientific and Technical Hurdles

- Dealing with millions of cores/nodes
  - Scheduling
    - Hierarchical scheduling
    - Data-flow task bases approaches
      - Non-coherent architectures
      - Software data prefetching
    - Imbalance detection, prediction and (local) correction
      - Avoid global balancing strategies
      - Work stealing
  - Communication
    - Scalable implementations of MPI/PGAS
    - Minimize memory consumption (per connection)
    - Redesign of collective operations
      - Asynchrony, ovelap
    - Discourage use of global synchronization primitives?

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 in exciting times! (let's stay positive ☺)

Thank you!

Questions?

- NB: more information at
  http://runtime.bordeaux.inria.fr