StarPU : Exploiting heterogeneous architectures through task-based programming

Cédric Augonnet
Nathalie Furmento
Raymond Namyst
Samuel Thibault

INRIA Bordeaux, LaBRI, University of Bordeaux

Fourth Swedish Workshop on Multicore Computing 23rd 2011

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

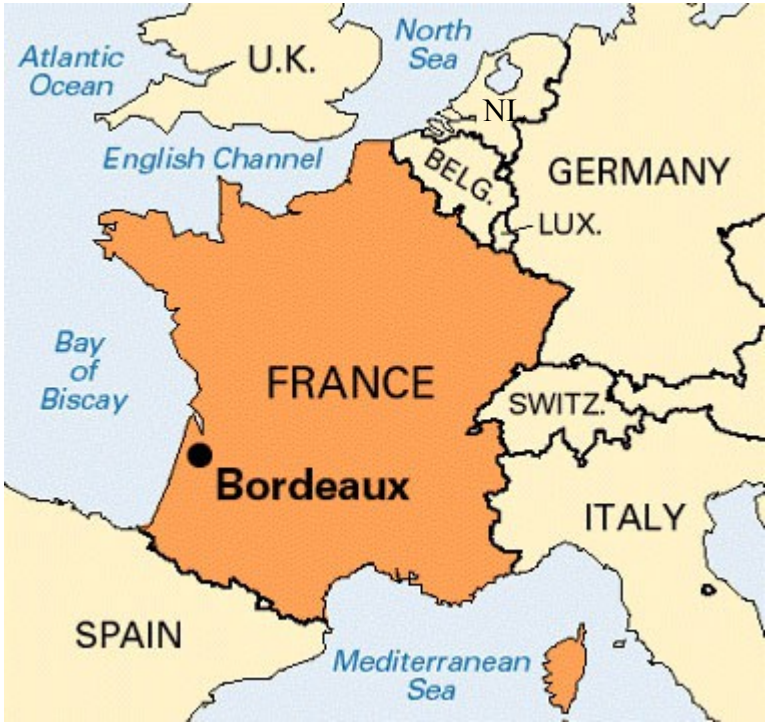
INRIA

centre de recherche BORDEAUX - SUD-OUEST

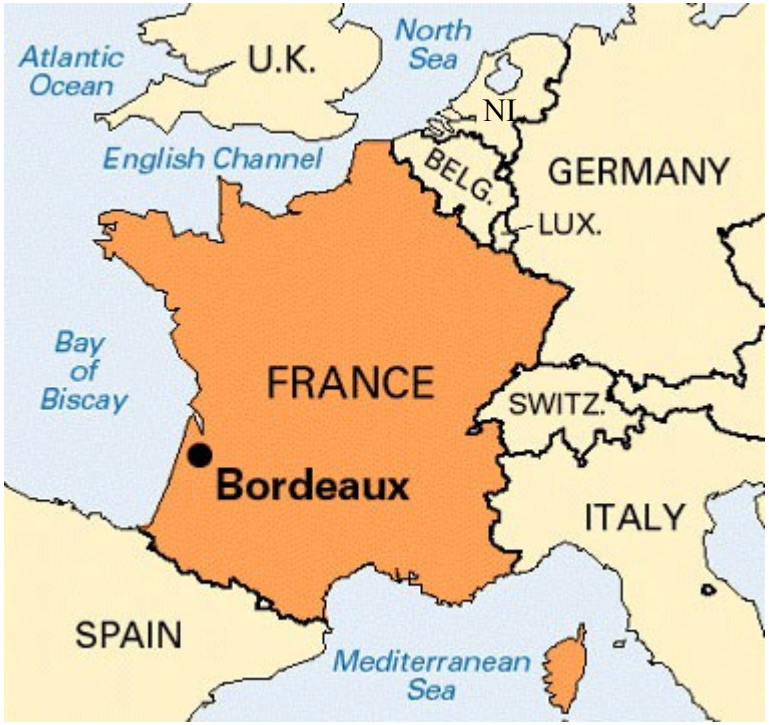
PEPPHER Programmability & Portability

SEVENTH FRAMEWORK PROGRAMME

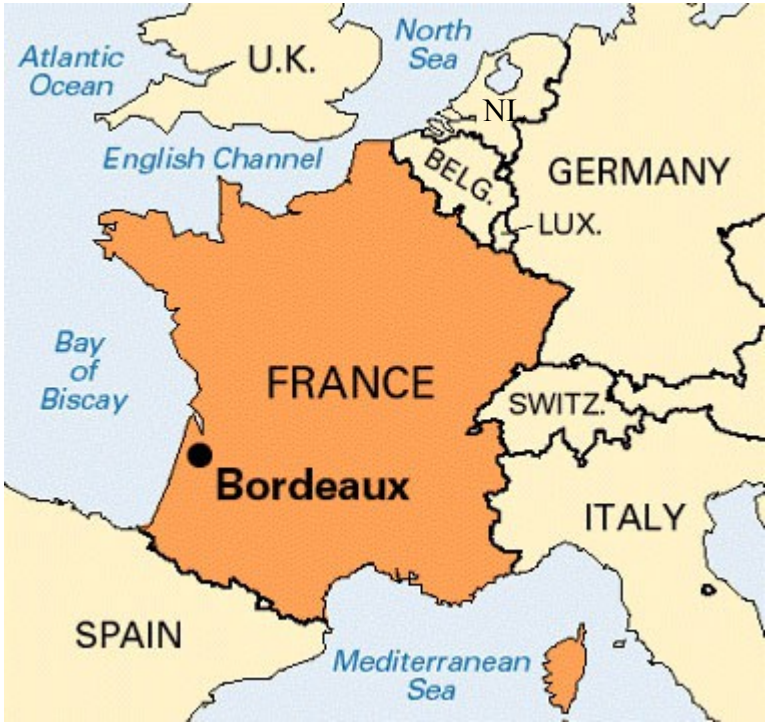
The RUNTIME Team



The RUNTIME Team



The RUNTIME Team



Doing Parallelism for centuries !

The RUNTIME Team

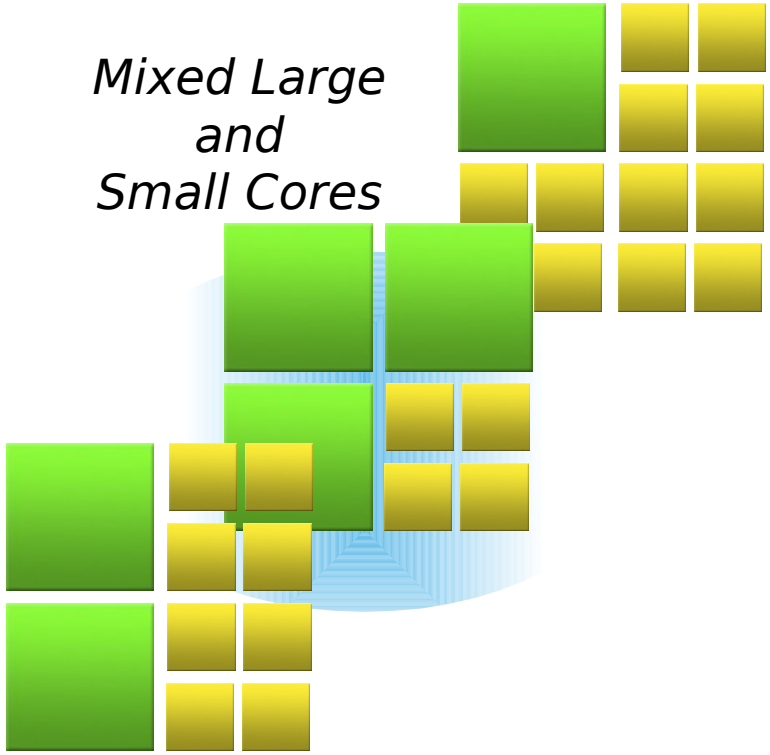
Research directions

- High Performance Runtime Systems for Parallel Architectures
  - "*Runtime Systems perform dynamically what cannot be not statically*"
- 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
- See http://runtime.bordeaux.inria.fr/ for more information

Introduction

Toward heterogeneous multi-core architectures

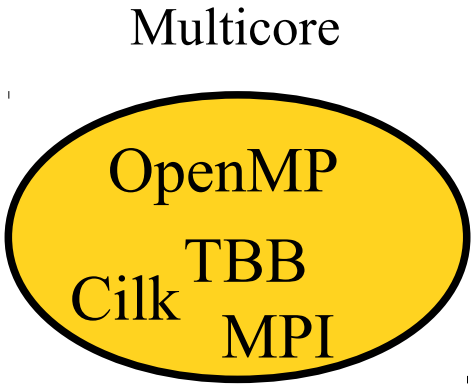
- Multicore is here
  - Hierarchical architectures
  - Manycore is coming
  - Power is a major concern
- Architecture specialization
  - Now
    - Accelerators (GPGPUs, FPGAs)
    - Coprocessors (Cell's SPUs)
  - In the (near?) Future
    - Many simple cores
    - A few full-featured cores

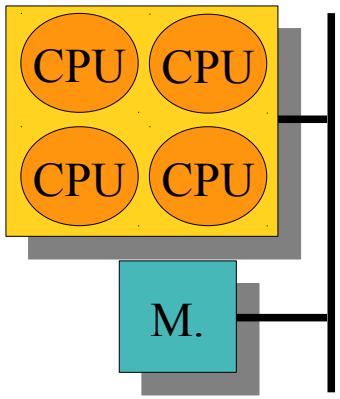


Introduction

How to program these architectures?

- Multicore programming
  - pthreads, OpenMP, TBB, ...

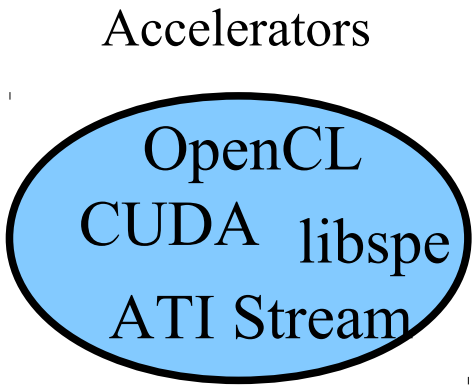


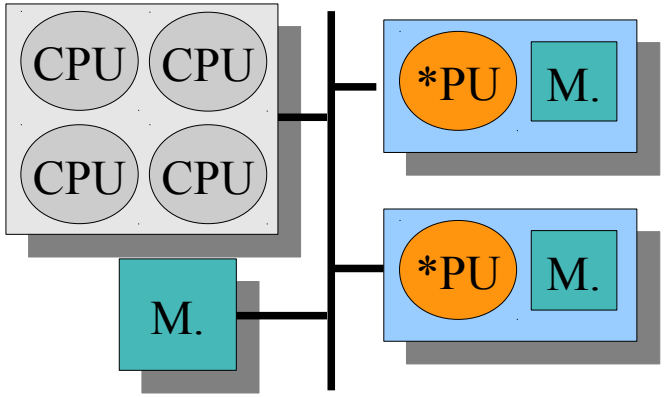


Introduction

How to program these architectures?

- Multicore programming
  - pthreads, OpenMP, TBB, ...
- Accelerator programming
  - Consensus on OpenCL?
  - (Often) Pure offloading model

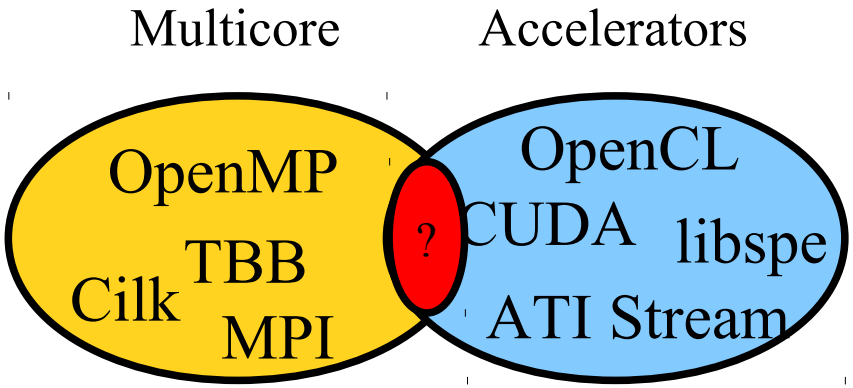


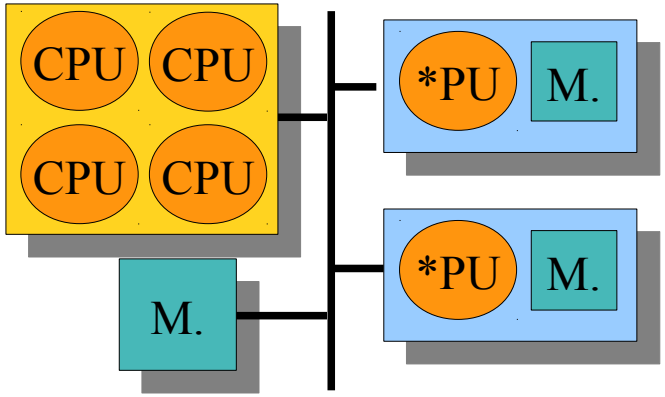


Introduction

How to program these architectures?

- Multicore programming
  - pthreads, OpenMP, TBB, ...
- Accelerator programming
  - Consensus on OpenCL?
  - (Often) Pure offloading model
- Hybrid models?
  - **Take advantage of all resources** ☺
  - **Complex interactions** ☹

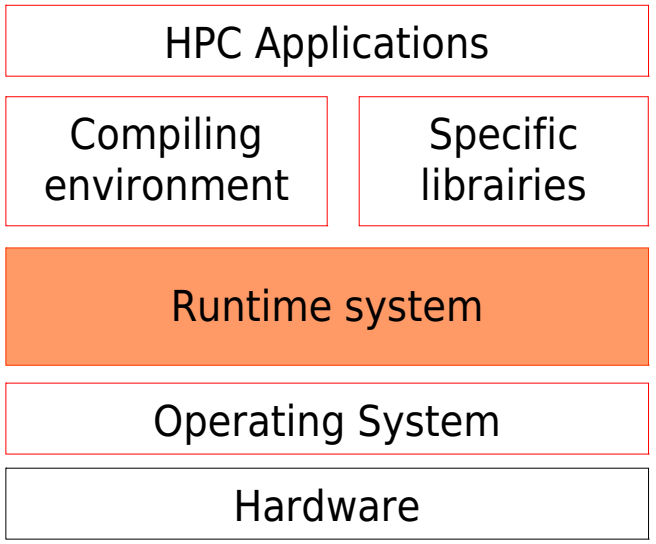




Introduction

Challenging issues at all stages

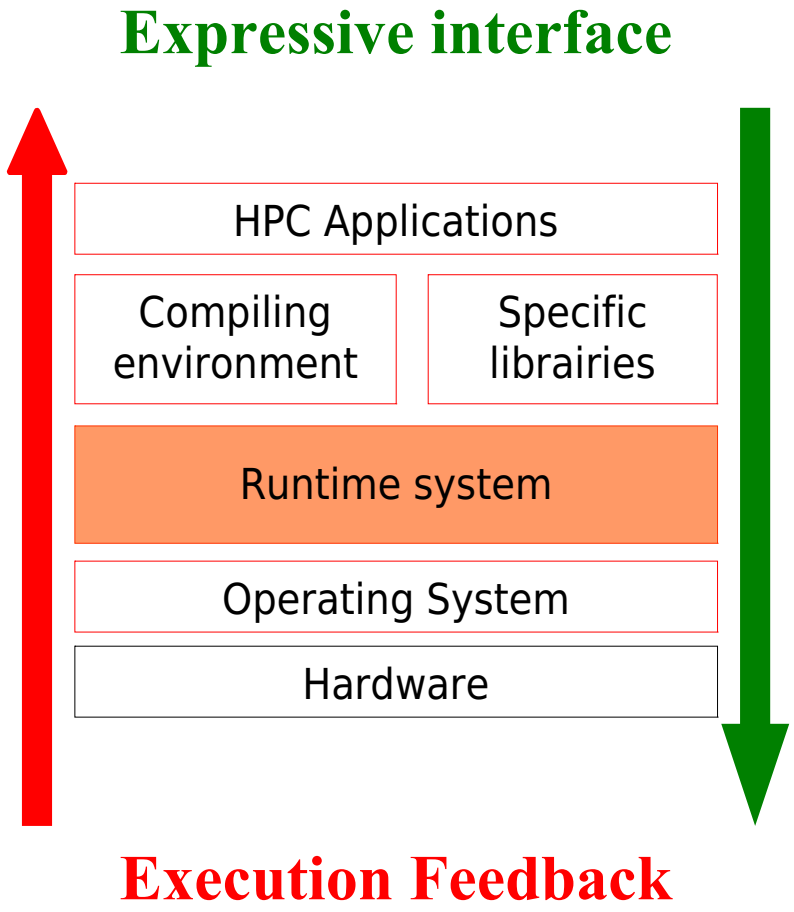
- Applications
  - Programming paradigm
  - BLAS kernels, FFT, …
- Compilers
  - Languages
  - Code generation/optimization
- Runtime systems
  - Resources management
  - Task scheduling
- Architecture
  - Memory interconnect



Introduction

Challenging issues at all stages

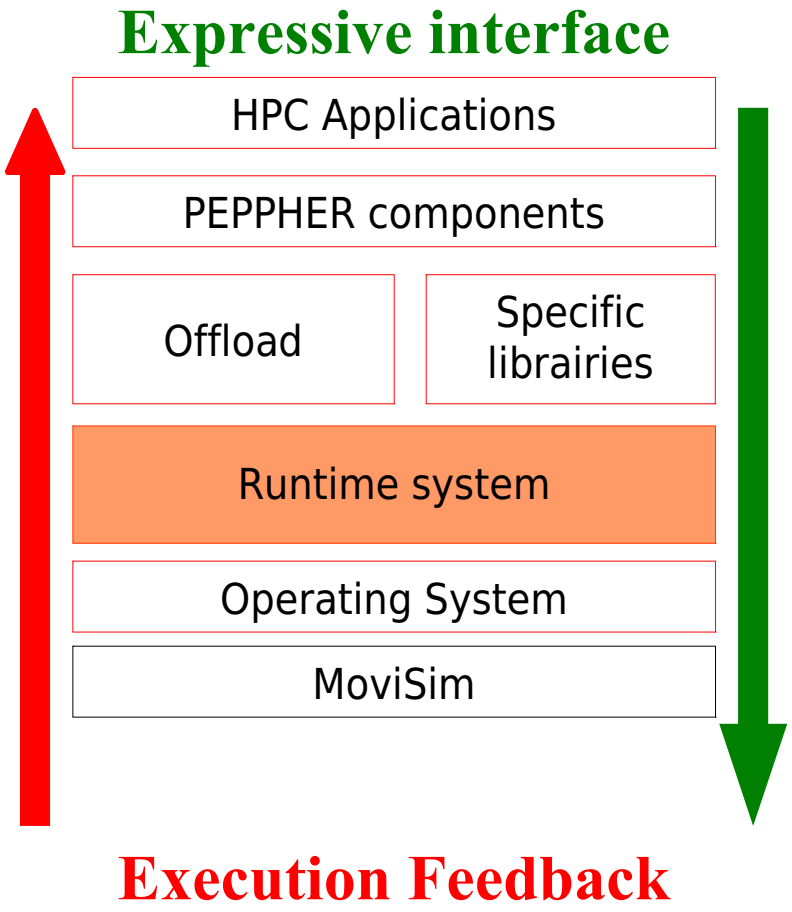
- Applications
  - Programming paradigm
  - BLAS kernels, FFT, …
- Compilers
  - Languages
  - Code generation/optimization
- Runtime systems
  - Resources management
  - Task scheduling
- Architecture
  - Memory interconnect



Introduction

Challenging issues at all stages

- Applications
  - Programming paradigm
  - BLAS kernels, FFT, …
- Compilers
  - Languages
  - Code generation/optimization
- Runtime systems
  - Resources management
  - Task scheduling
- Architecture
  - Memory interconnect



Outline

- Overview of StarPU
- Programming interface
- Task & data management

- Task scheduling
- MAGMA+PLASMA example
- Performance analysis tools
- Experimental features

- Conclusion

Overview of StarPU

Overview of StarPU

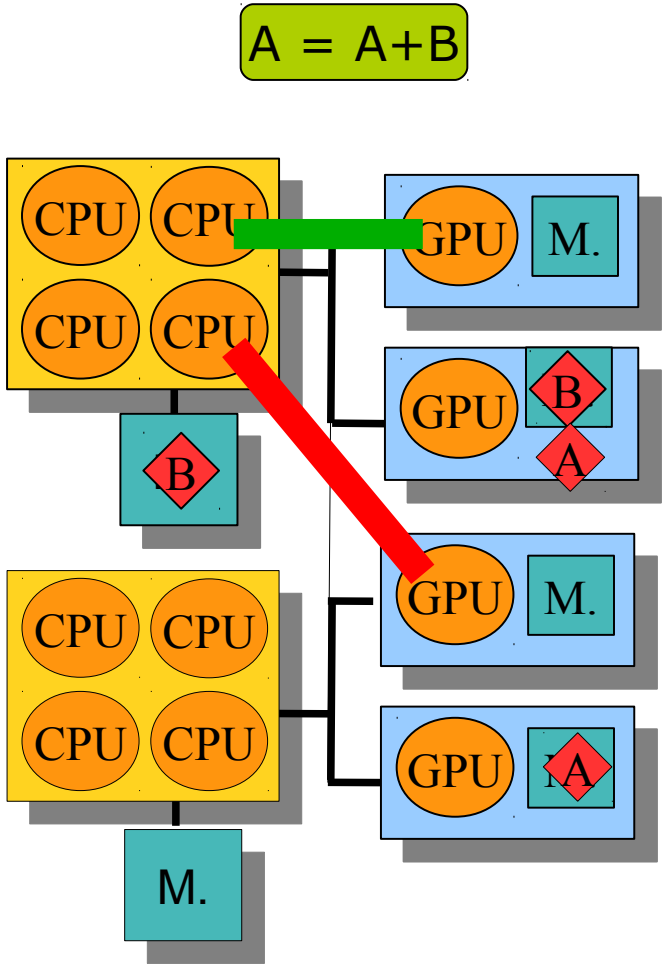
Rationale

Dynamically schedule tasks on all processing units

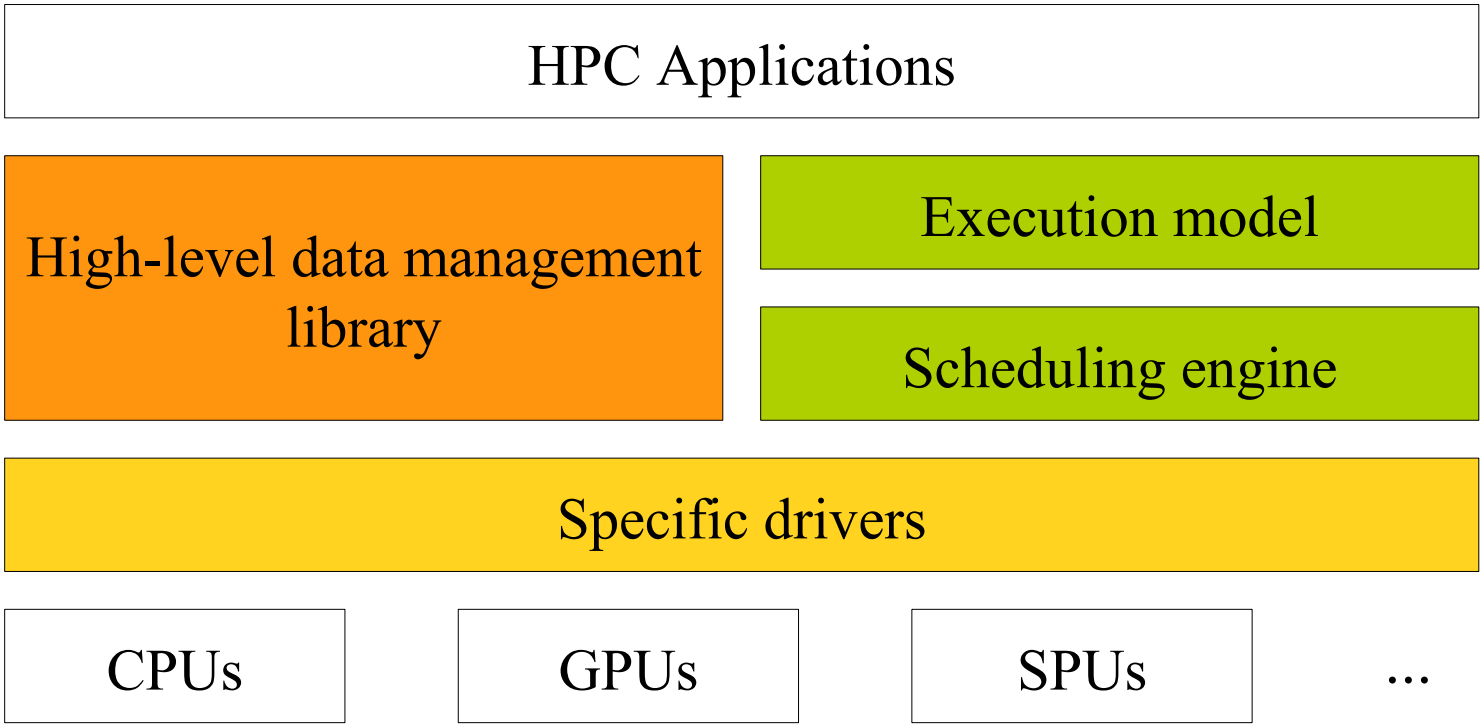
- See a pool of heterogeneous processing units

Avoid unnecessary data transfers between accelerators

- Software VSM for heterogeneous machines



The StarPU runtime system



Mastering CPUs, GPUs, SPUs … ***PUs**

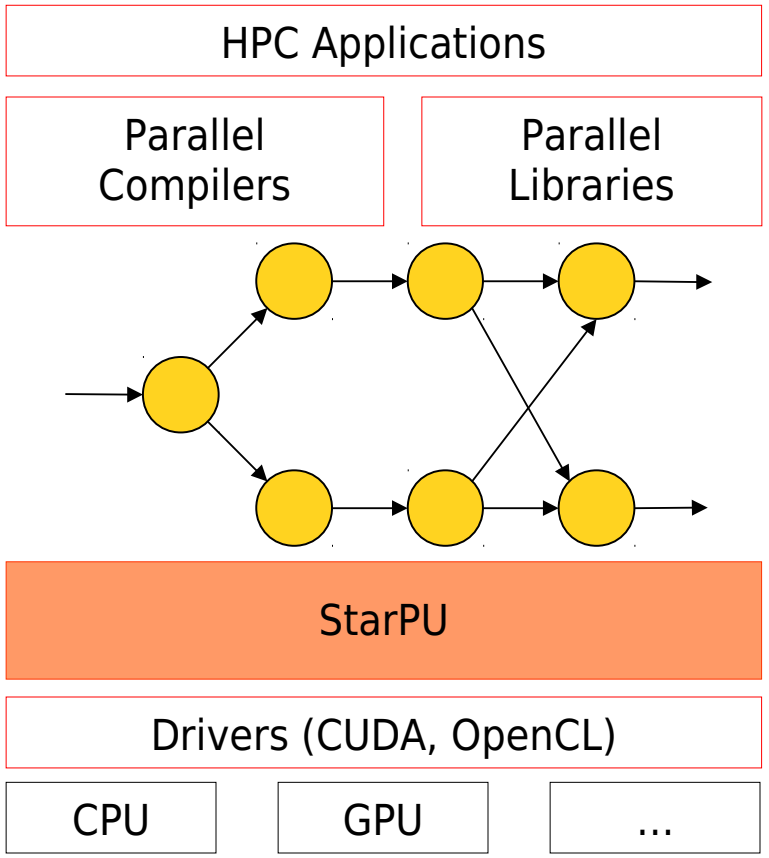
The StarPU runtime system

The need for runtime systems

- "do dynamically what can't be done statically anymore"

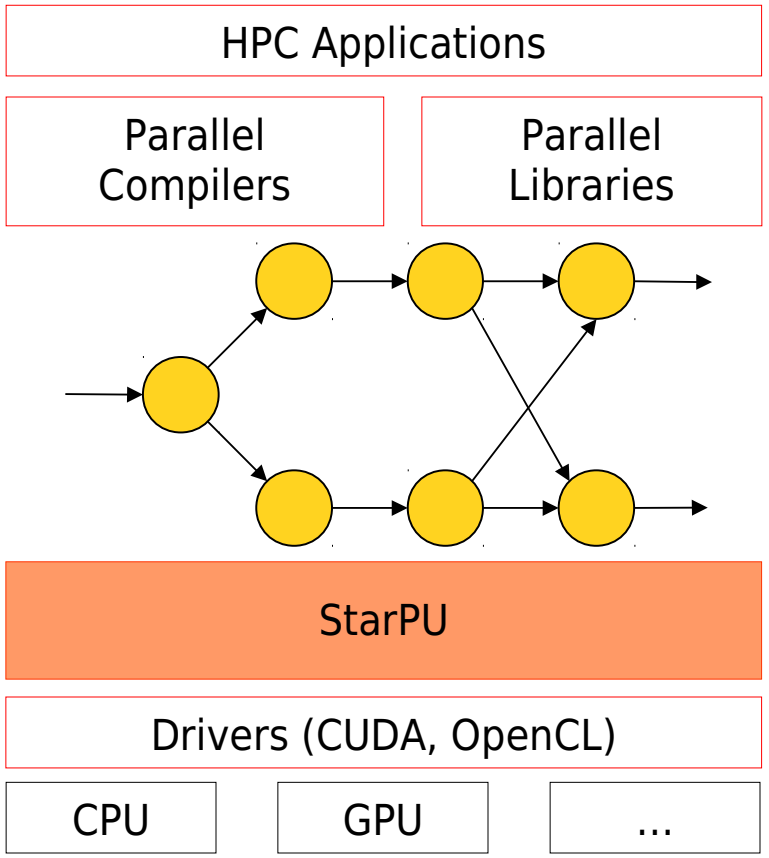
- StarPU provides
  - Task scheduling
  - Memory management

- Compilers and libraries generate (graphs of) parallel tasks
  - Additional information is welcome!



Data 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

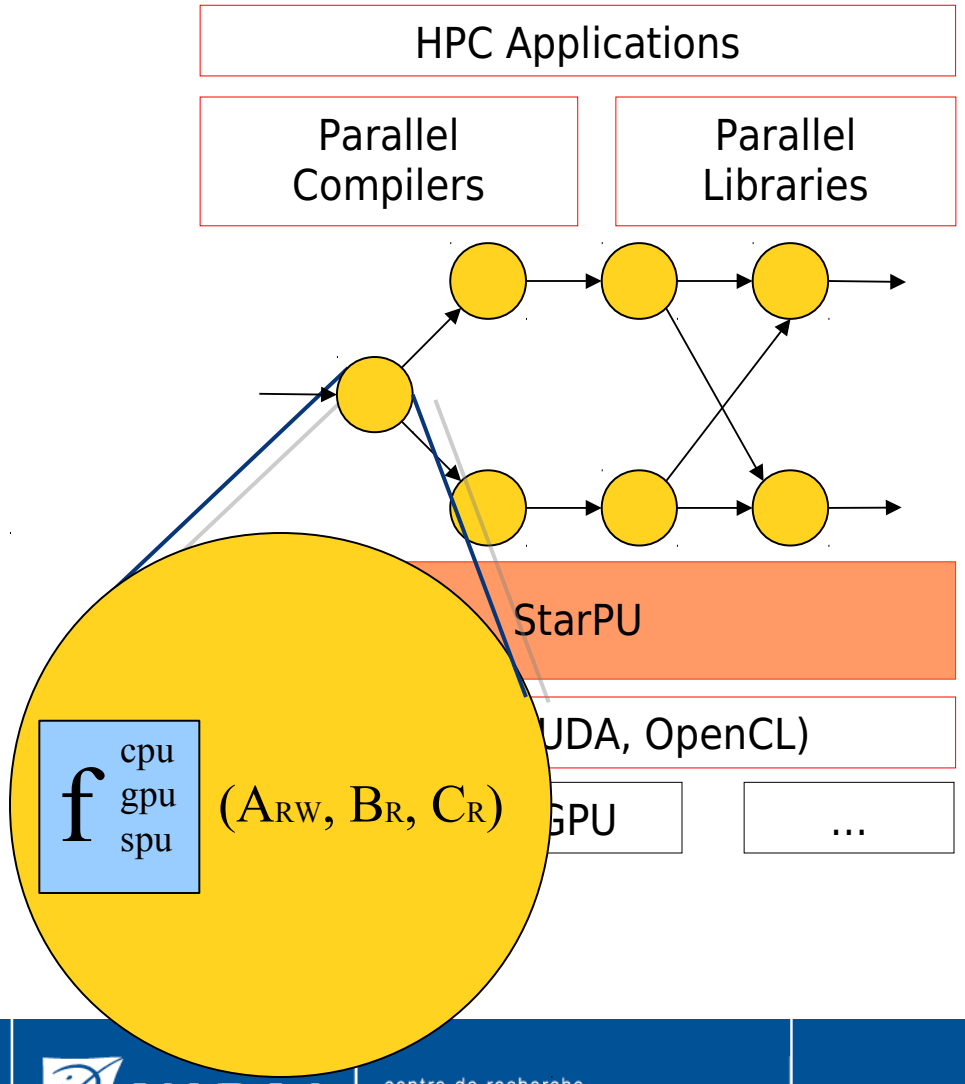


The StarPU runtime system

Task scheduling

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

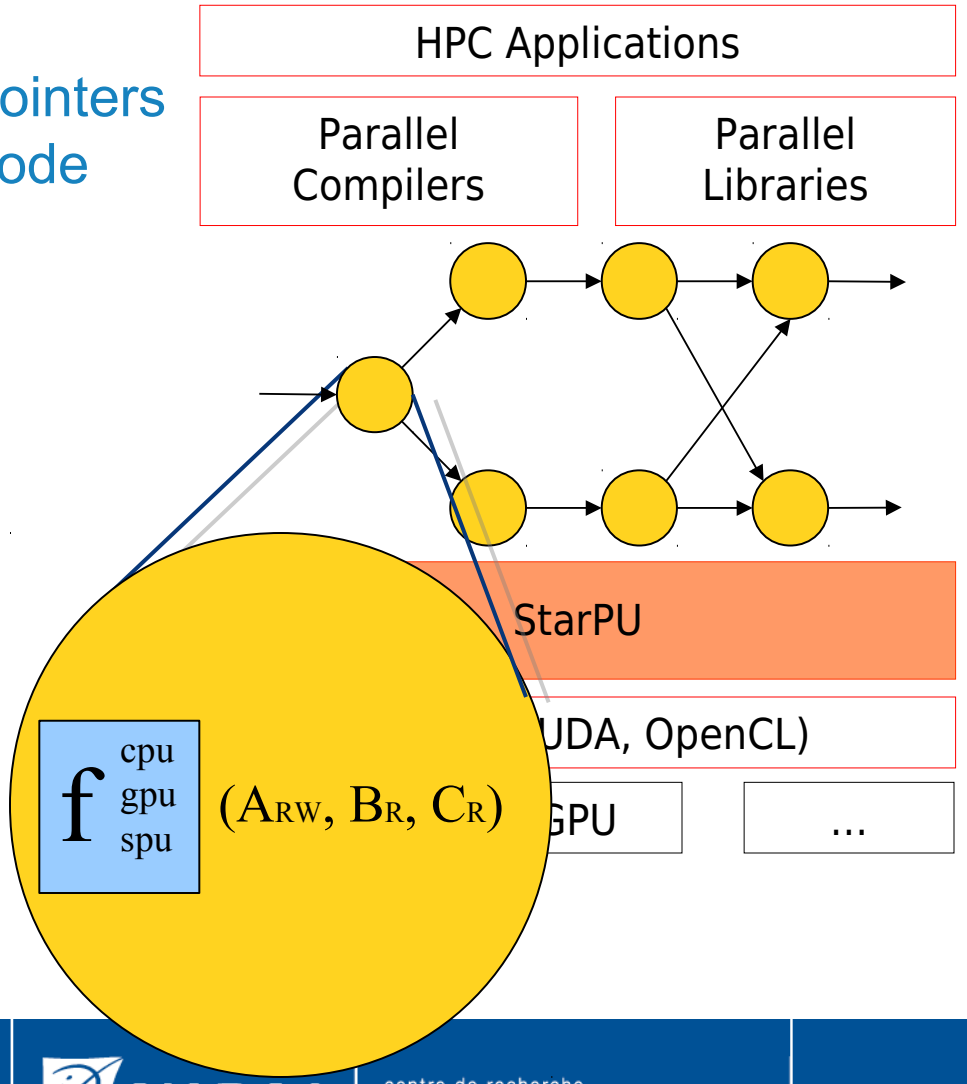
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



The StarPU runtime system

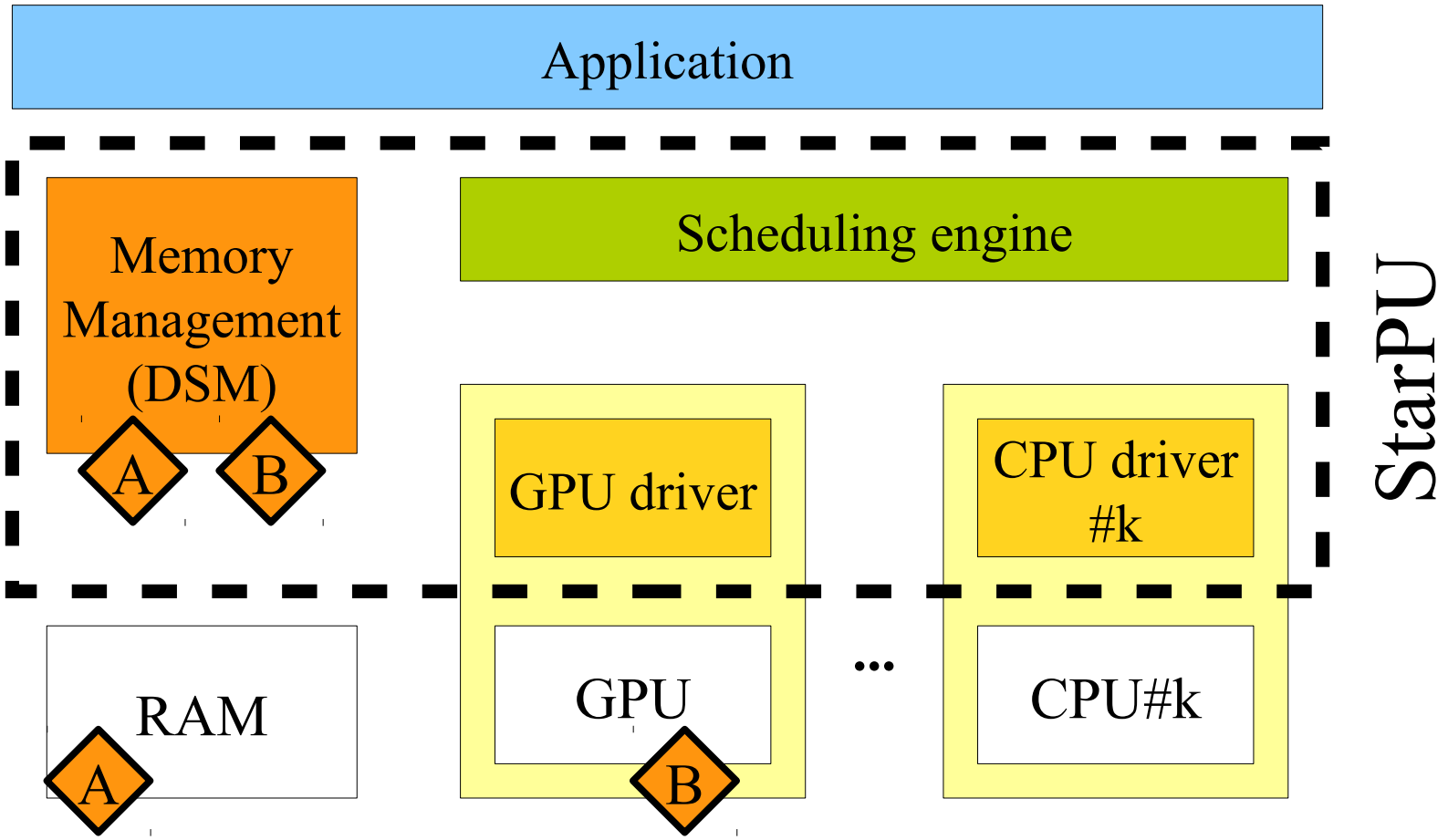
Task scheduling

- Who generates the code ?
  - StarPU Task = ~function pointers
  - StarPU doesn't generate code
- Libraries era
  - PLASMA + MAGMA
  - FFTW + CUFFT...
- Rely on compilers
  - PGI accelerators
  - CAPS HMPP...



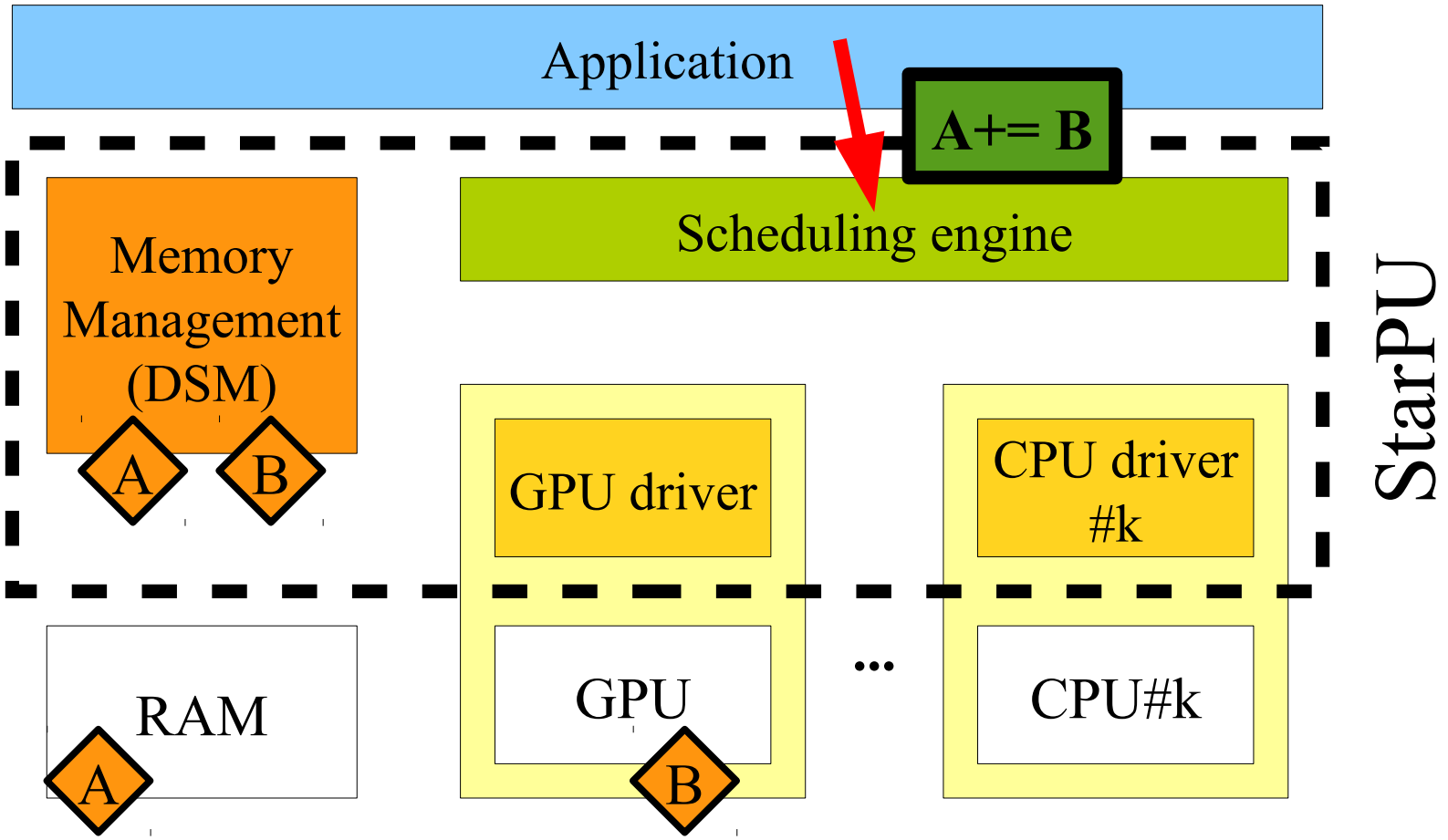
The StarPU runtime system

Execution model



The StarPU runtime system

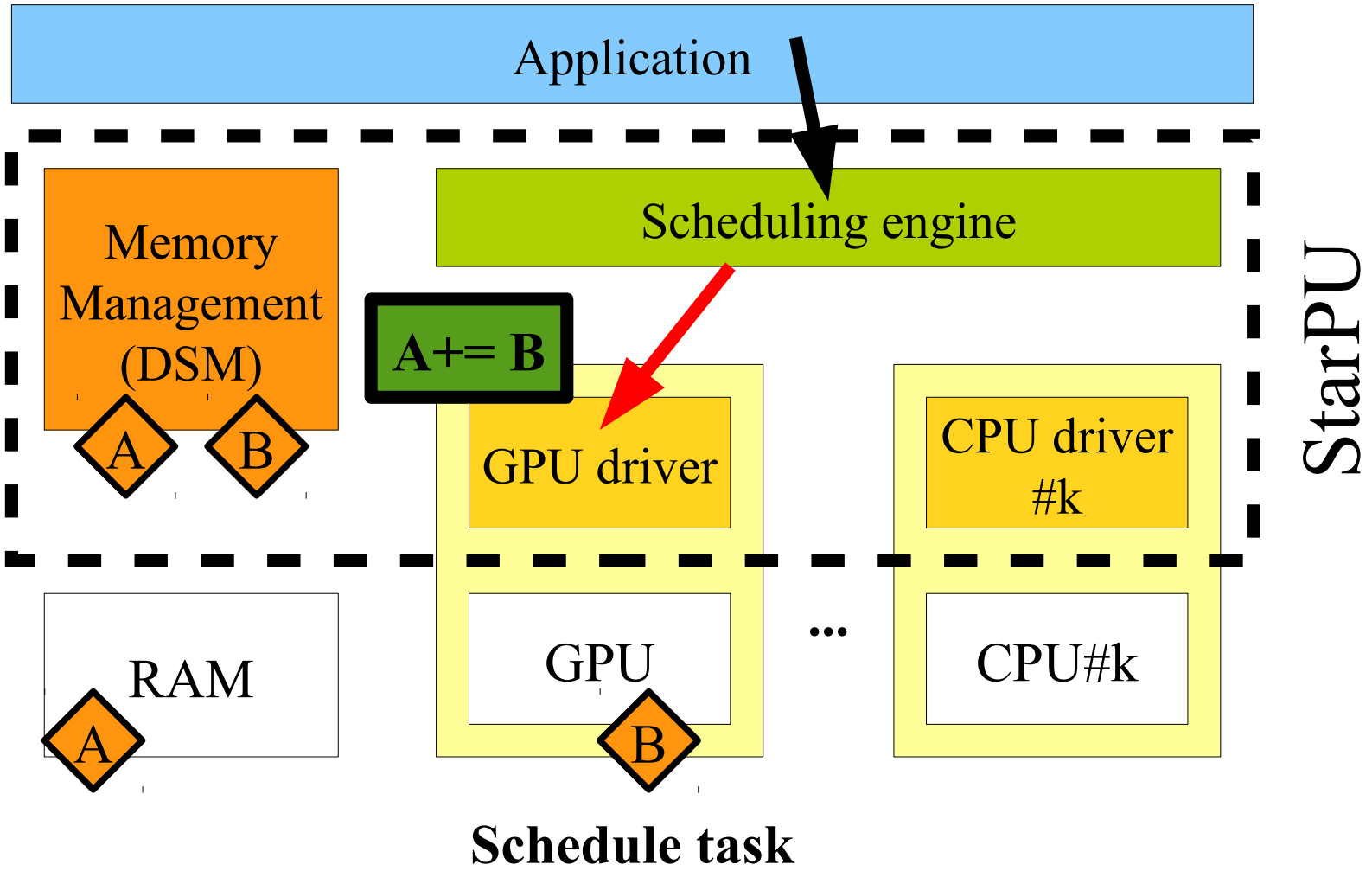
Execution model



Submit task « A += B »

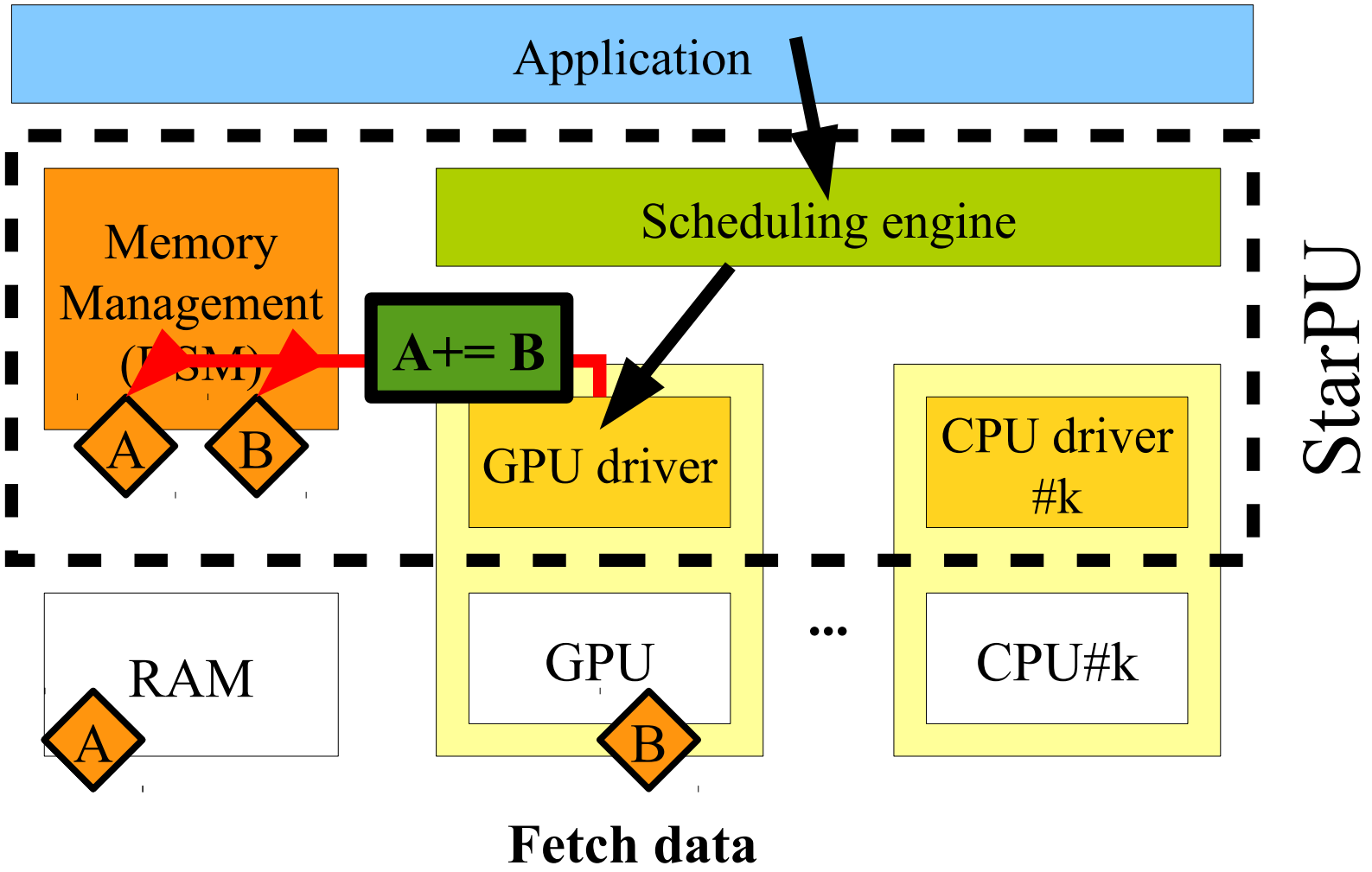
The StarPU runtime system

Execution model



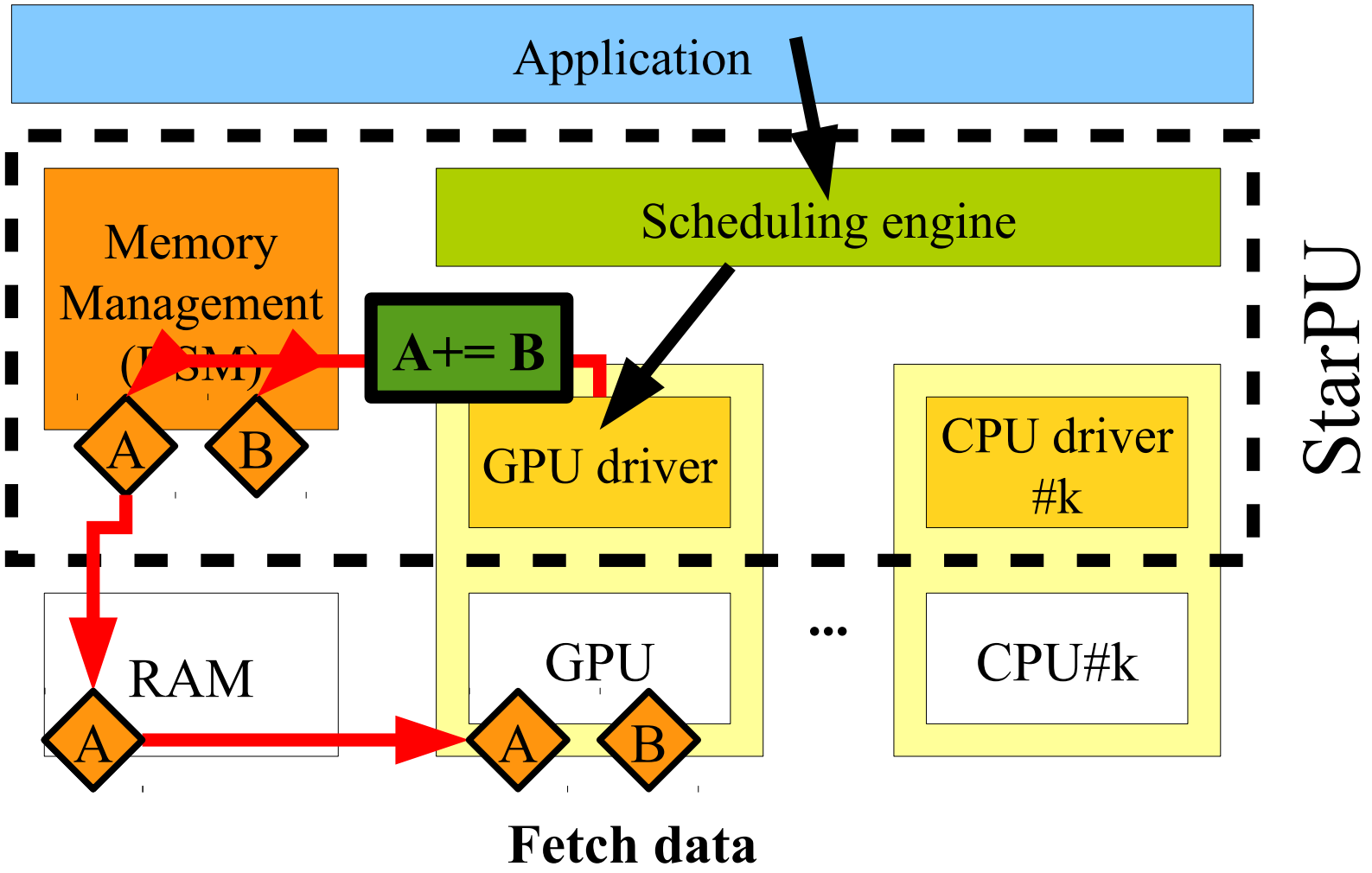
The StarPU runtime system

Execution model



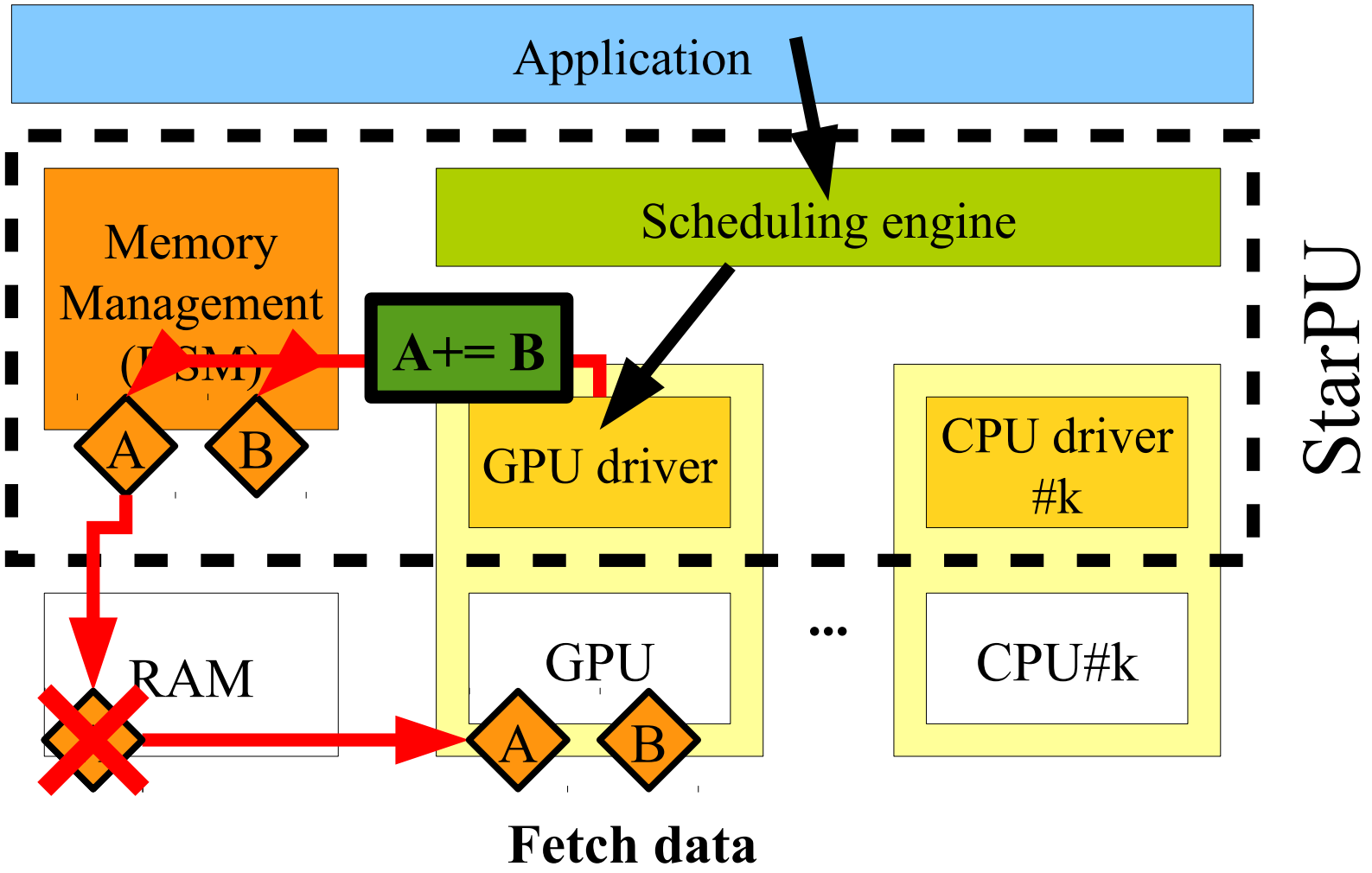
The StarPU runtime system

Execution model



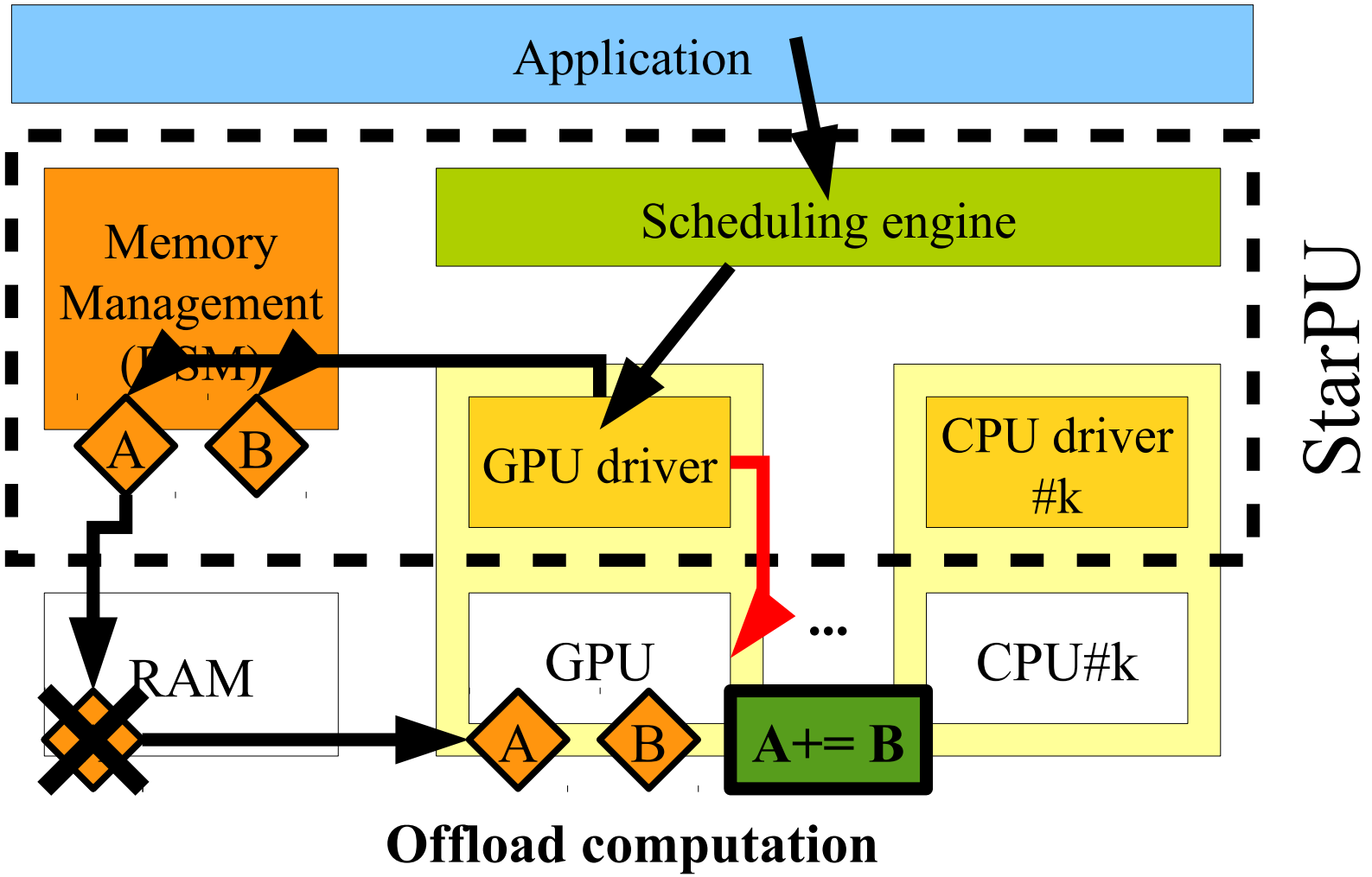
The StarPU runtime system

Execution model



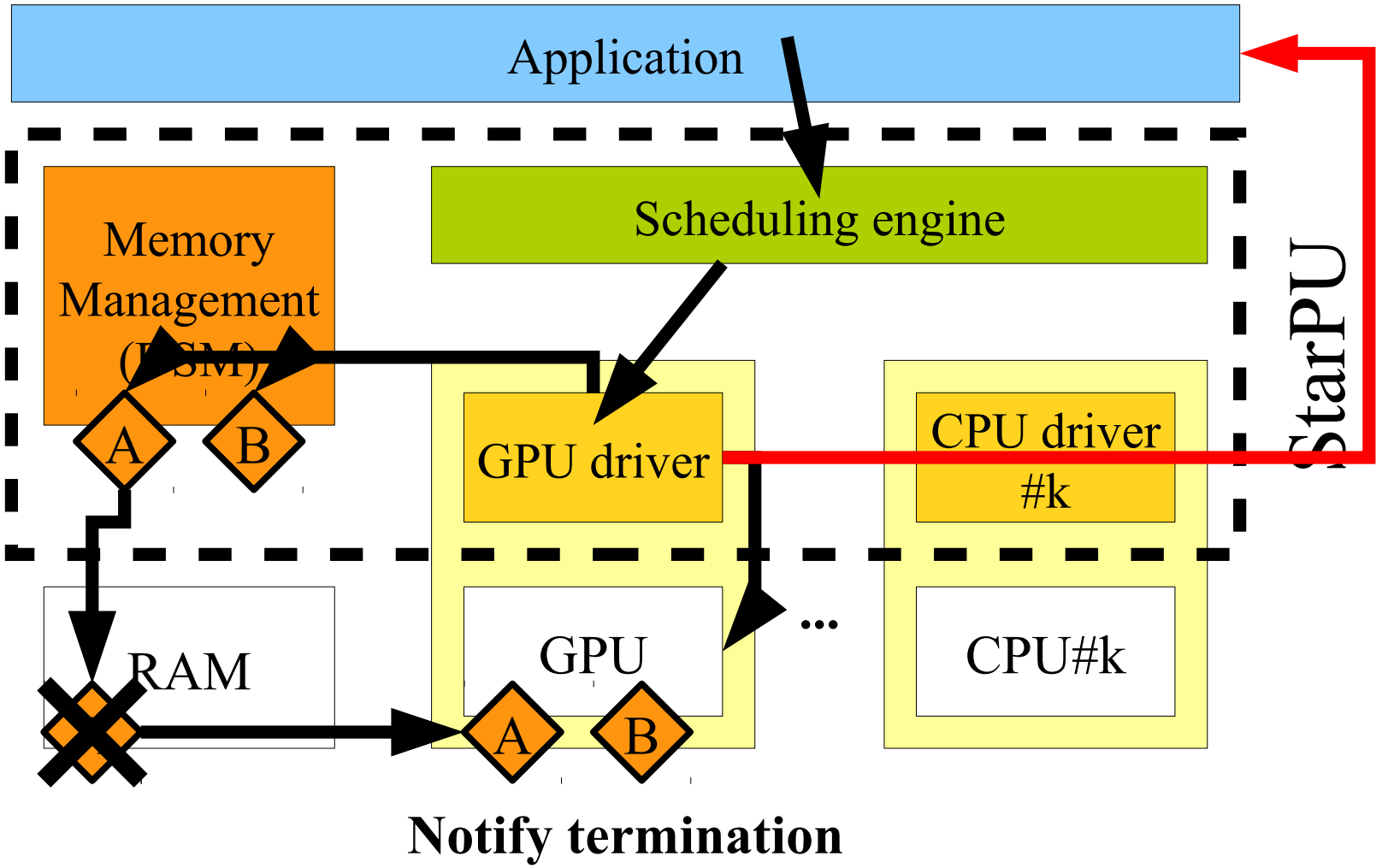
The StarPU runtime system

Execution model



The StarPU runtime system

Execution model



The StarPU runtime system

Development context

- History
  - Started about 4 years ago
  - StarPU main core ~ 20k lines of code
  - Written in C
  - 4 core developers
    - Cédric Augonnet, Samuel Thibault, Nathalie Furmento, Cyril Roelandt

- Open Source
  - Released under LGPL
  - Sources freely available
    - svn repository and nightly tarballs
    - See http://runtime.bordeaux.inria.fr/StarPU/
  - Open to external contributors

The StarPU runtime system

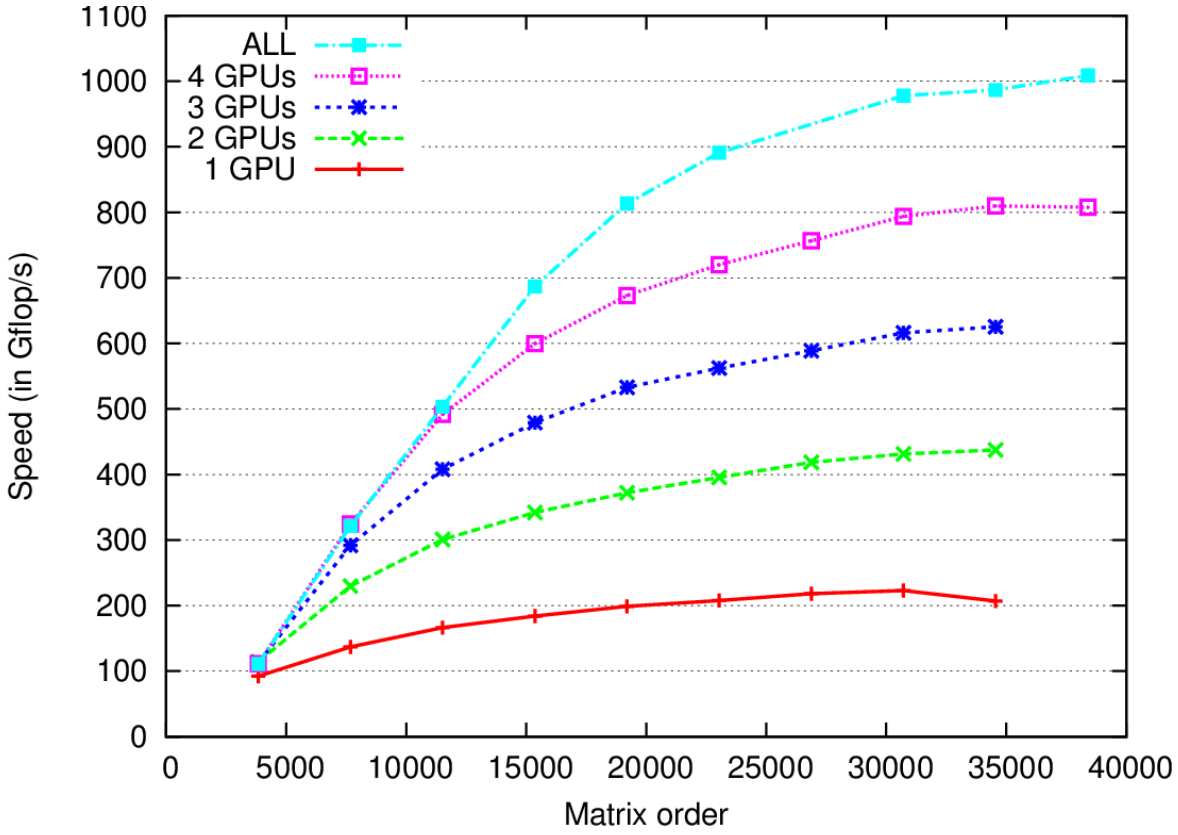
Supported platforms

- Supported architectures
  - Multicore CPUs (x86, PPC, ...)
  - NVIDIA GPUs
  - OpenCL devices (eg. AMD cards)
  - Cell processors (experimental)

- Supported Operating Systems
  - Linux
  - Mac OS
  - Windows

Performance teaser

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)



Programming interface

Scaling a vector

Launching StarPU

- Makefile flags
  - CFLAGS +=$(shell pkg-config --cflags libstarpu)
  - LDFLAGS+=$(shell pkg-config --libs libstarpu)
- Headers
  - #include <starpu.h>
- (De)Initialize StarPU
  - starpu_init(NULL);
  - starpu_shutdown();

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-versionned 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

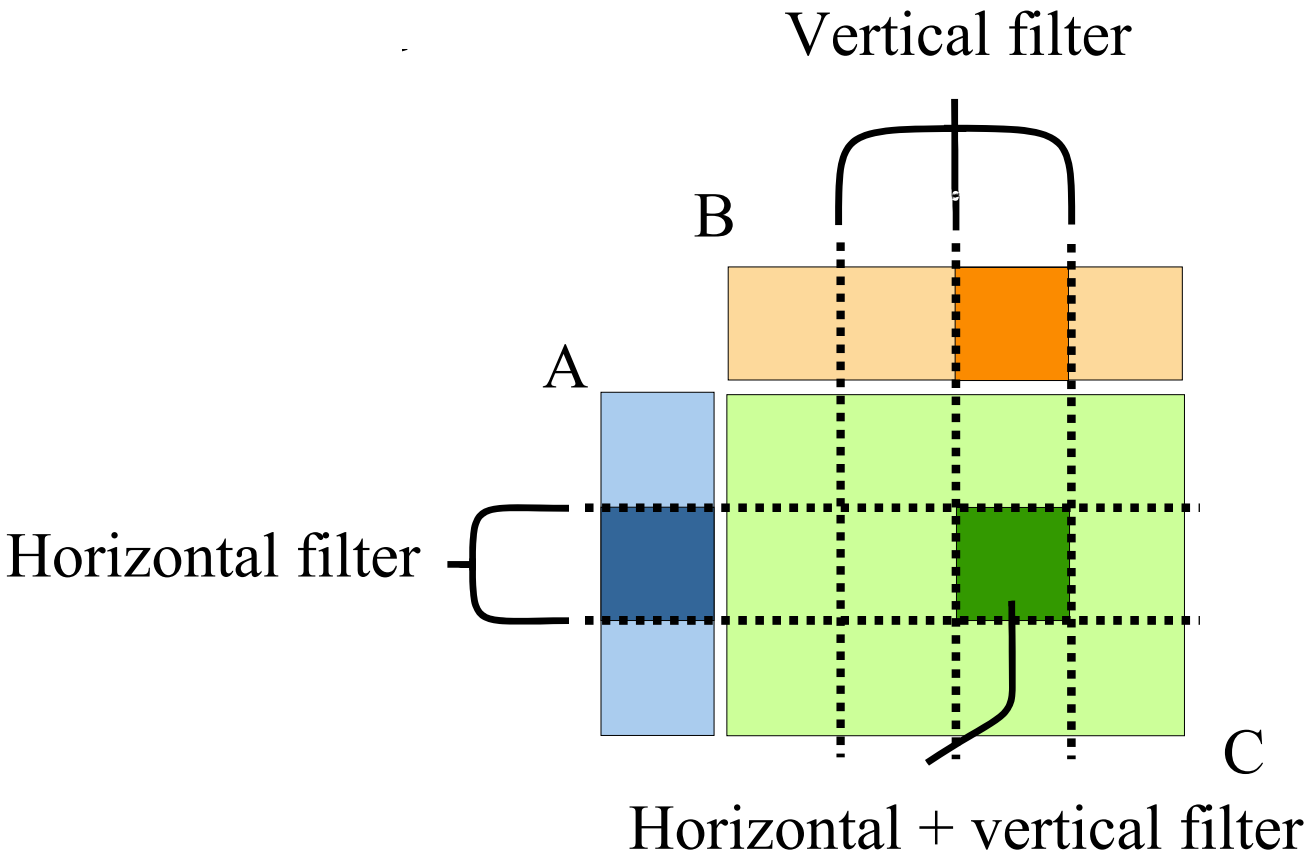
Defining a task, starpu_insert_task helper

- 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);
```

Partitioning data

- Example: matrix multiplication



Partitioning Data

- Partition matrices vertically / horizontally / both

```
struct starpu_data_filter vert = {
        .filter_func = starpu_vertical_block_filter_func,
        .nchildren = nslicesx
}
struct starpu_data_filter horiz = {
        .filter_func = starpu_block_filter_func,
        .nchildren = nslicesy
}
starpu_data_partition(B_handle, &vert);
starpu_data_partition(A_handle, &horiz);
starpu_data_map_filters(C_handle, 2, &vert, &horiz);
```

Partitioning Data

- Accessing parts

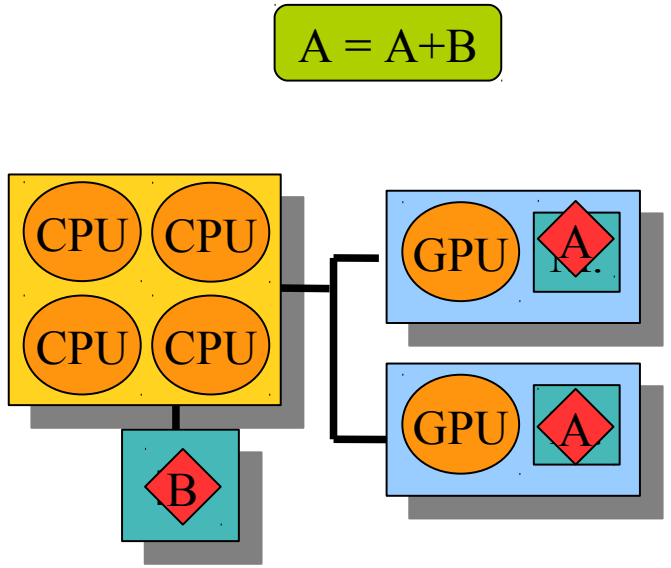
```
for (x = 0; x < nslicesx; x++) {
        for (y = 0; y < nslicesy; y++) {
            starpu_data_handle
                subA = starpu_data_get_sub_data(A_handle, 1, y),
                subB = starpu_data_get_sub_data(B_handle, 1, x),
                subC = starpu_data_get_sub_data(C_handle, 2, x, y);

            starpu_insert_task(&mult_cl,
                STARPU_R, subA, STARPU_R, subB,
                STARPU_RW, subC, 0);
        }
}
```

More details on Data Management

StarPU data interfaces

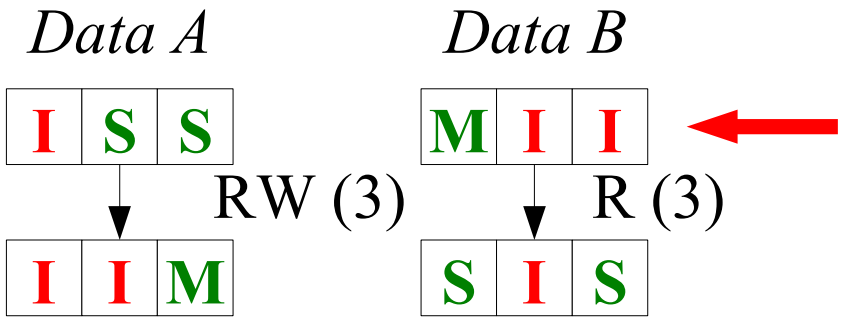
StarPU data coherency protocol

- Memory nodes
  - Each worker is associated to a node
  - Multiple workers may share a node



- Data coherency
  - Keep track of replicates
  - Discard invalid replicates

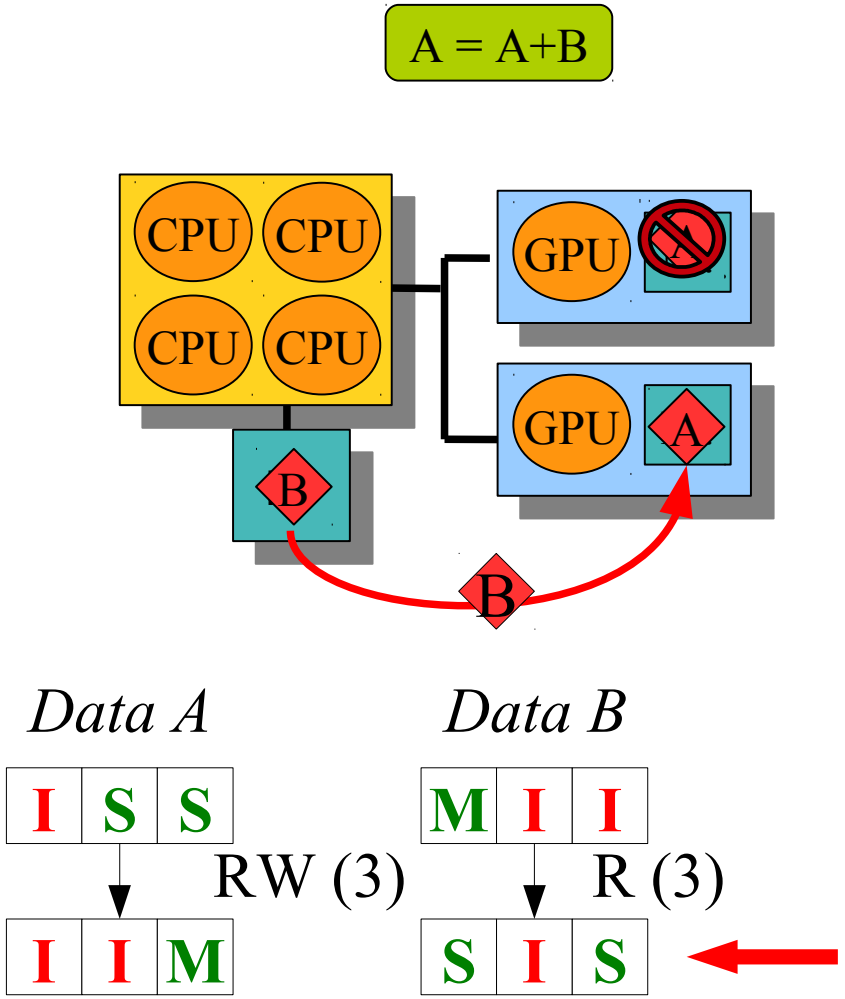
- MSI coherency protocol
  - M : Modified
  - S : Shared
  - I : Invalid



StarPU data interfaces

StarPU data coherency protocol

- Memory nodes
  - Each worker is associated to a node
  - Multiple workers may share a node
- Data coherency
  - Keep track of replicates
  - Discard invalid replicates
- MSI coherency protocol
  - M : Modified
  - S : Shared
  - I : Invalid



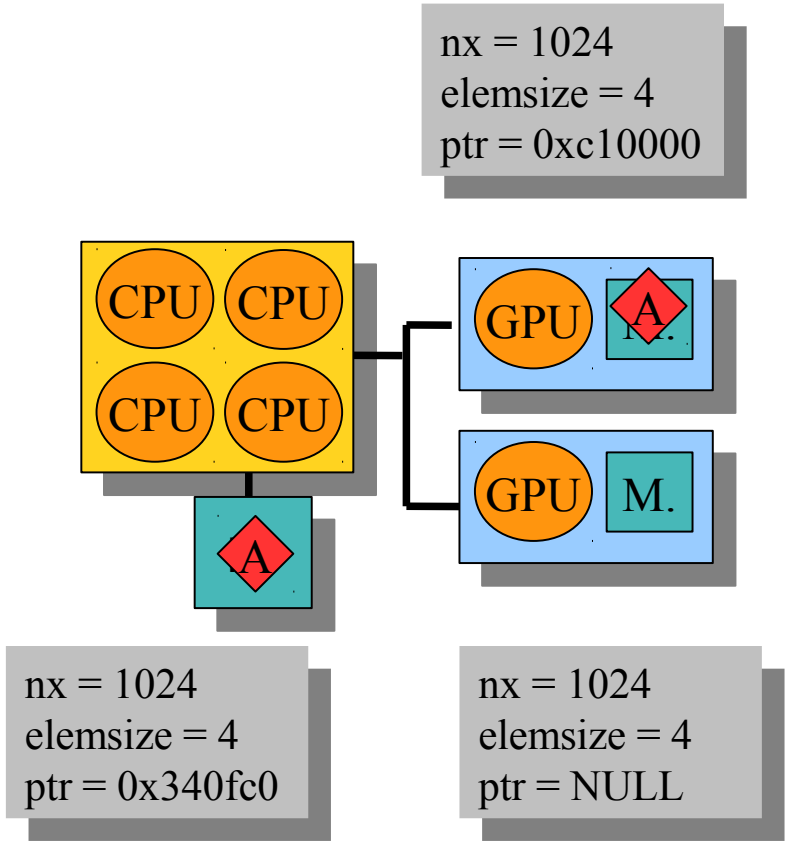
StarPU data interfaces

StarPU data interfaces

- Each piece of data is described by a structure
  - Example : vector interface

```
struct starpu_vector_interface_s {
        unsigned nx;
        unsigned elemsize;
        uintptr_t ptr;
}
```

  - Matrices formats, ...
  - StarPU ensures that interfaces are coherent
- StarPU tasks are passed pointers to these interfaces
  - Coherency protocol is independent from the type of interface

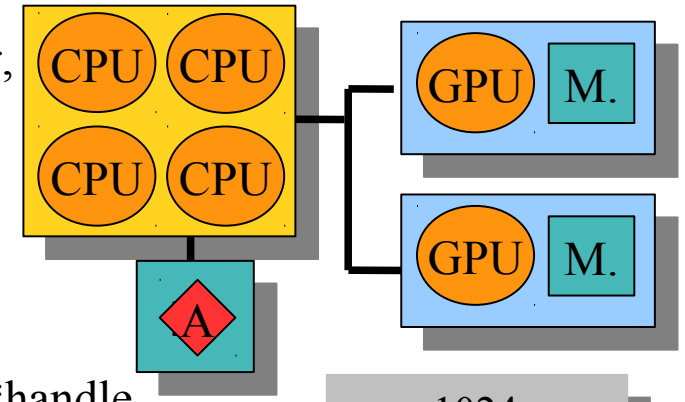


StarPU data interfaces

StarPU data interfaces

- Registering a piece of data
  - Generic method

```
starpu_data_register(starpu_data_handle *handleptr,
        uint32_t home_node, void *interface,
        struct starpu_data_interface_ops_t *ops);
```



  - Wrappers are available for existing interfaces

```
starpu_variable_data_register(starpu_data_handle *handle,
        uint32_t home_node,
        uintptr_t ptr, size_t elemsize);
```

nx = 1024
elemsize = 4
ptr = 0x340fc0

```
starpu_vector_data_register(starpu_data_handle *handle,
        uint32_t home_node,
        uintptr_t ptr, uint32_t nx, size_t elemsize);
```

```
starpu_csr_data_register(starpu_data_handle *handle, uint32_t home_node,
        uint32_t nnz, uint32_t nrow, uintptr_t nzval, uint32_t *colind,
        uint32_t *rowptr, uint32_t firstentry, size_t elemsize);
```

More details on Task Management

Task management

Task API

- Create tasks
  - Dynamically allocated by starpu_task_create
  - Otherwise, initialized by starpu_task_init
- Submit a task
  - starpu_task_submit(task)
    - blocking if task->synchronous = 1
- Wait for task termination
  - starpu_task_wait(task);
  - starpu_task_wait_for_all();
- Destroy tasks
  - starpu_task_destroy(task);
    - automatically called if task->destroy = 1
  - starpu_task_deinit(task);

Task management

The task structure

- struct starpu_task

- Task description
  - struct starpu_codelet_t *cl
  - void *cl_arg : constant data area passed to the codelet
  - Buffers array (accessed data + access mode)

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

  - void (*callback_func)(void *);
    - void *callback_arg;
    - Should not be a blocking call !
  - Extra hints for the scheduler
    - eg. priority level

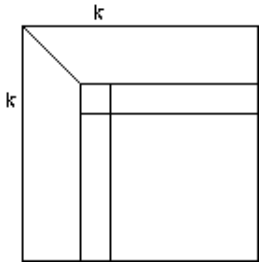
Task management

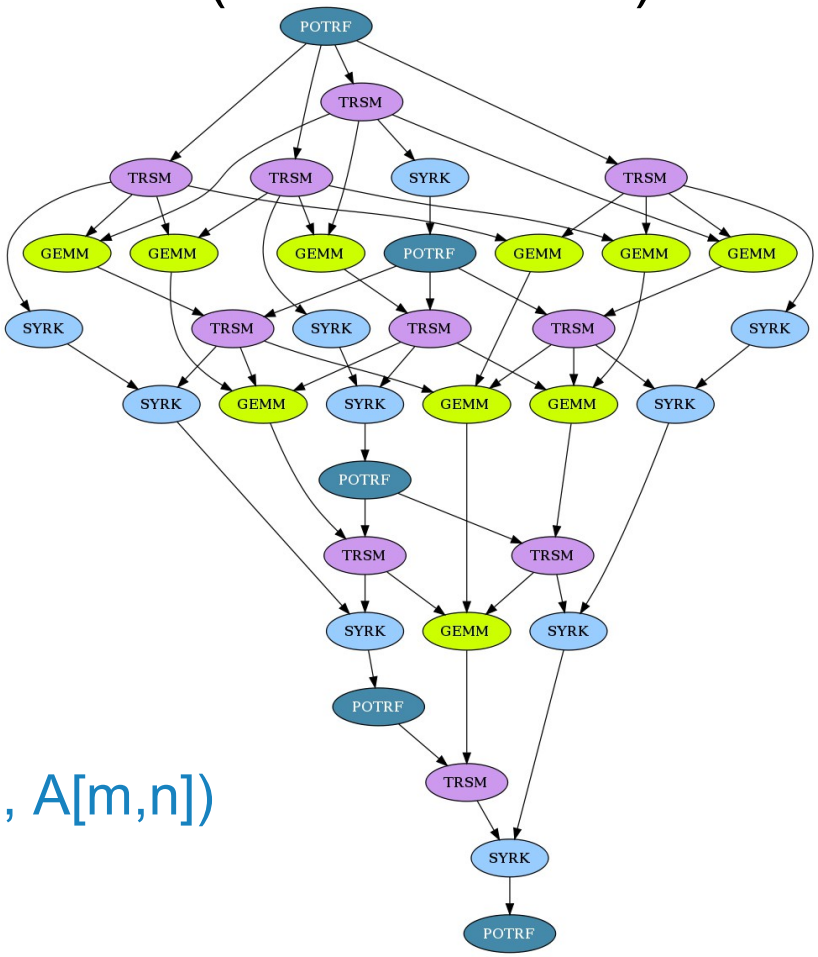
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)
  - For (k = 0 .. tiles – 1)
    {

```
POTRF(A[k,k])
for (m = k+1 .. tiles – 1)
    TRSM(A[k,k], A[m,k])
for (n = k+1 .. tiles – 1)
    SYRK(A[n,k], A[n,n])

for (n = k+1 .. tiles – 1)
    for (m = k+1 .. tiles – 1)
        GEMM(A[m,k], A[n,k], A[m,n])
```

    }





1st hands-on session

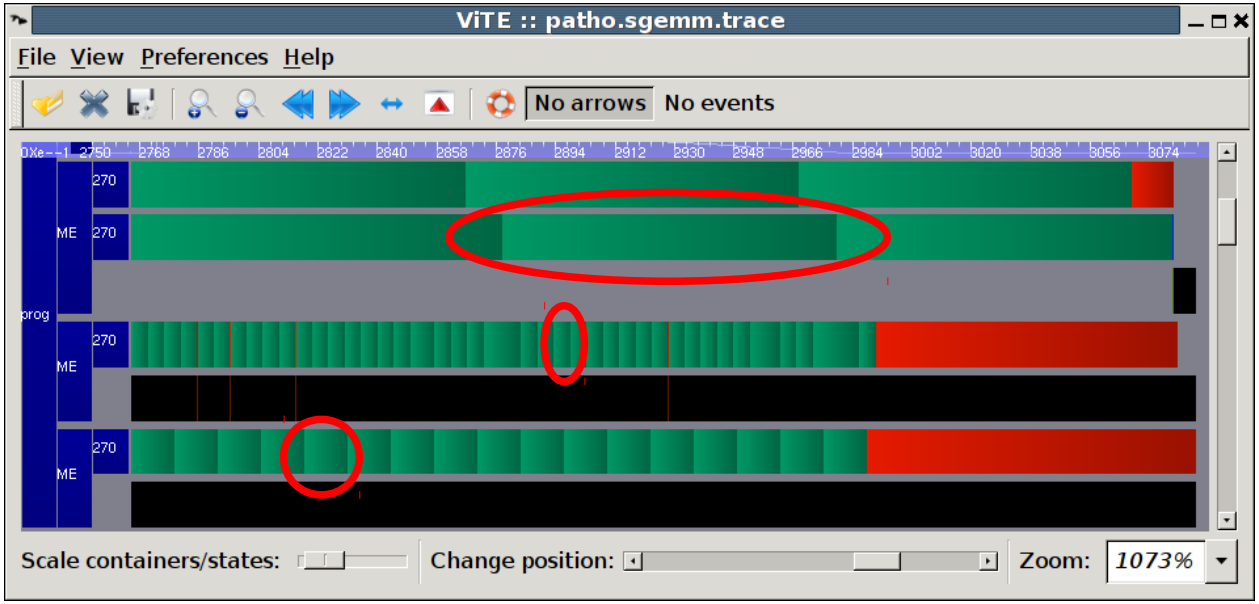
Task Scheduling

Why do we need task scheduling ?

Blocked Matrix multiplication

Things can go (really) wrong even on trivial problems !

- Static mapping ?
  - Not portable, too hard for real-life problems
- Need Dynamic Task Scheduling
  - Performance models



2 Xeon cores

Quadro FX5800

Quadro FX4600

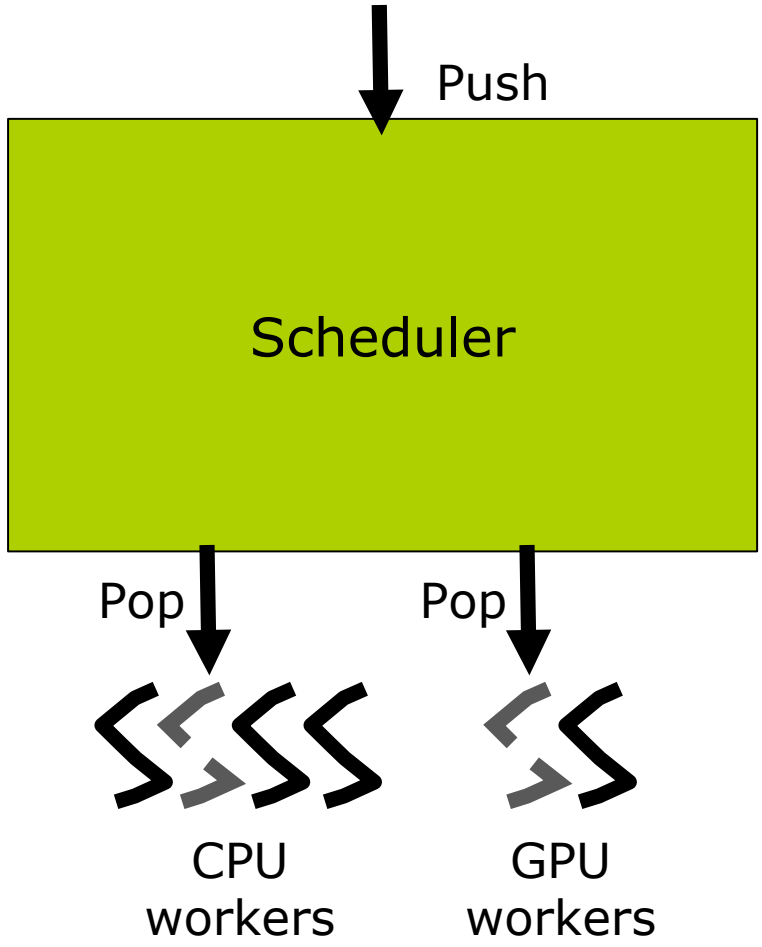
Task scheduling

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 poll for work ("pop")

Various scheduling policies, can even be user-defined



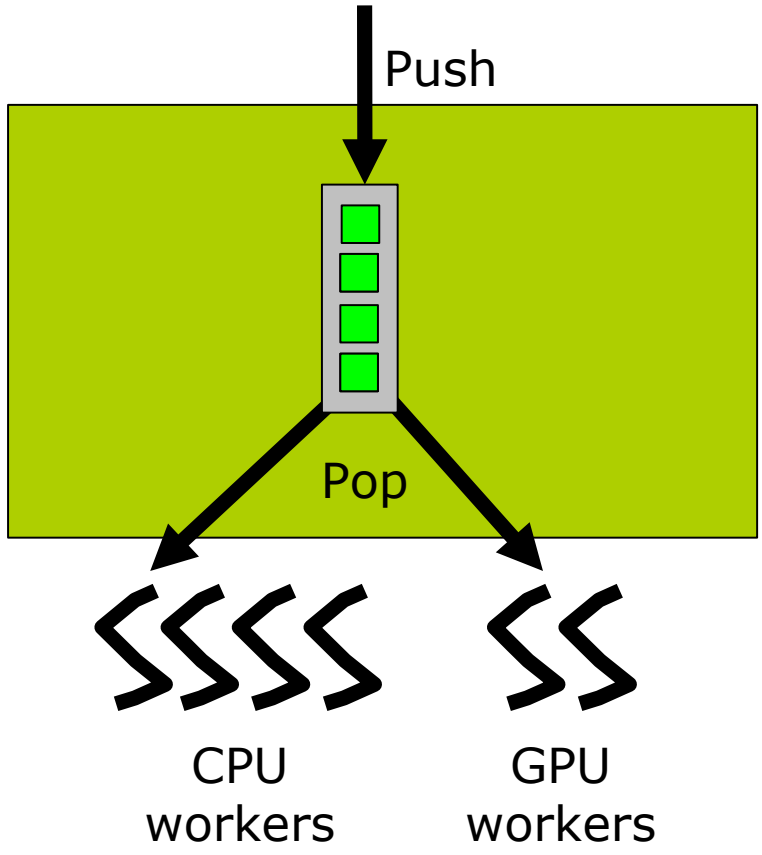
Task scheduling

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 poll for work (“pop”)

Various scheduling policies, can even be user-defined



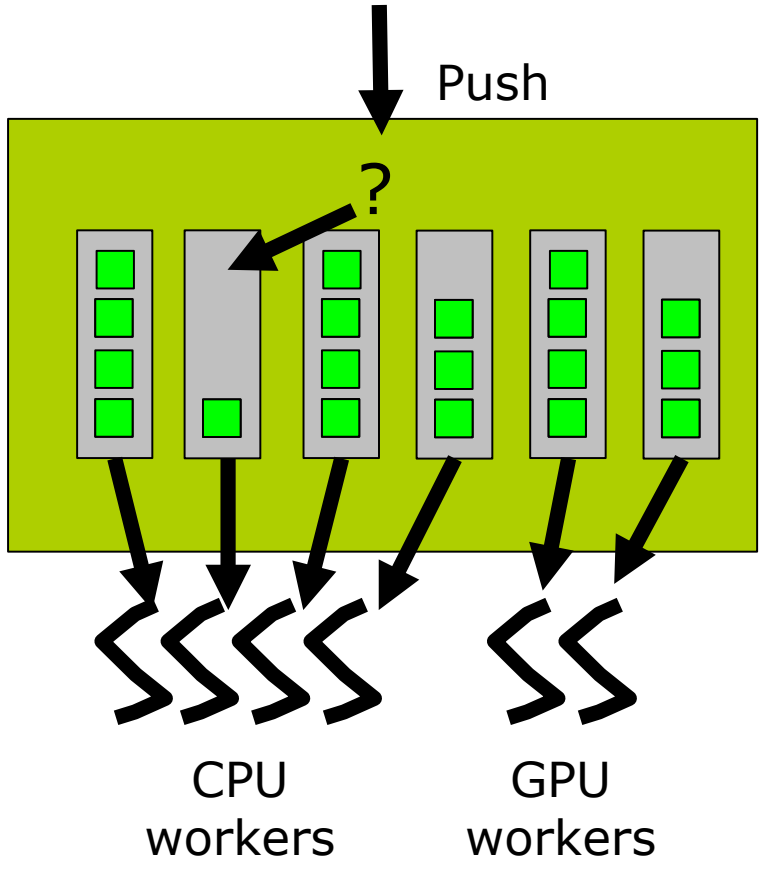
Task scheduling

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 poll for work (“pop”)

Various scheduling policies, can even be user-defined



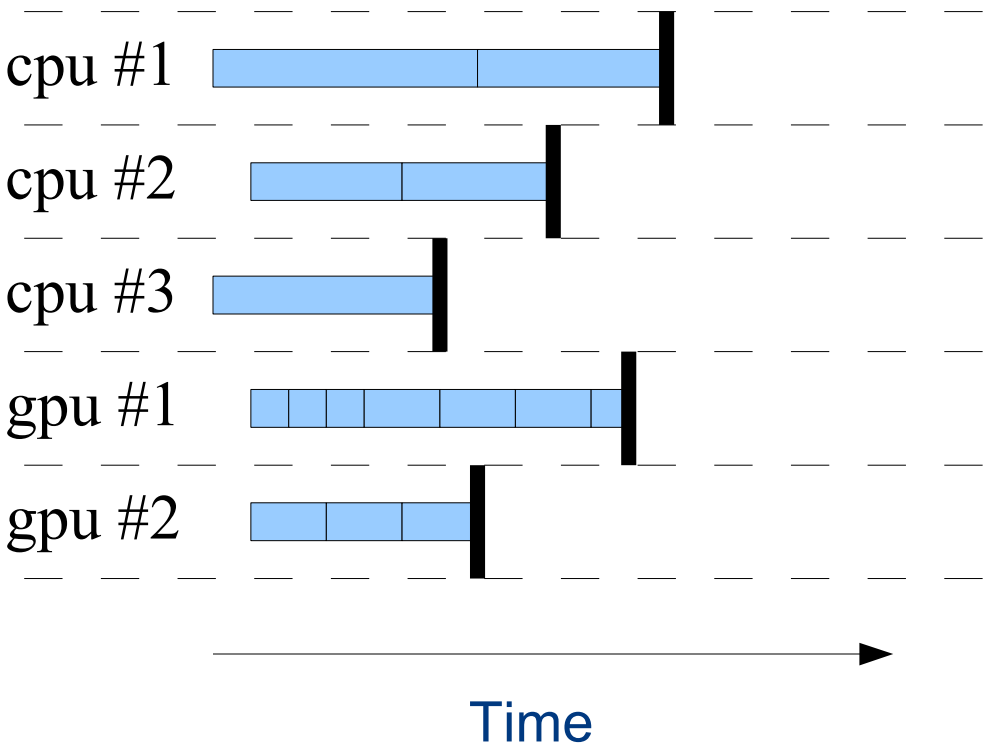
History-based performance model

```
struct starpu_perfmodel_t cl_model = {
    .type = STARPU_HISTORY_BASED,
    .symbol = "my_codelet",
};
starpu_codelet scal_cl = {
    .where = STARPU_CPU | ...
    .cpu_func = scal_cpu_func,
...
    .model = &cl_model
};
```

Also STARPU_REGRESSION_BASED, STARPU_NL_REGRESSION_BASED, or explicit

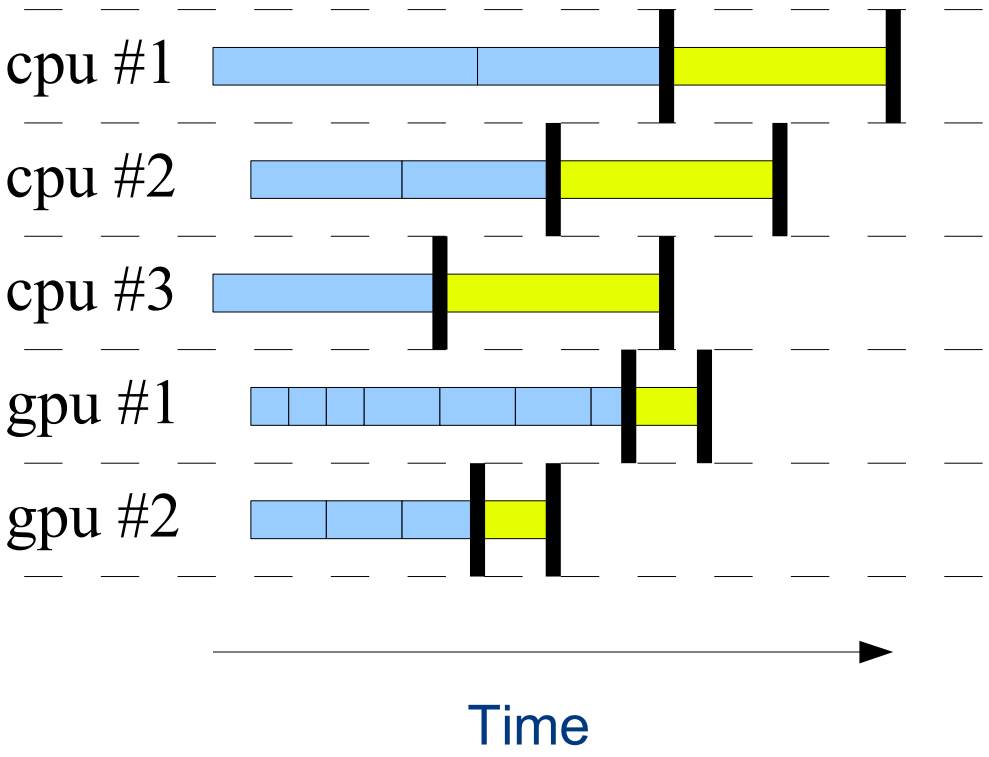
Prediction-based scheduling
Load balancing

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



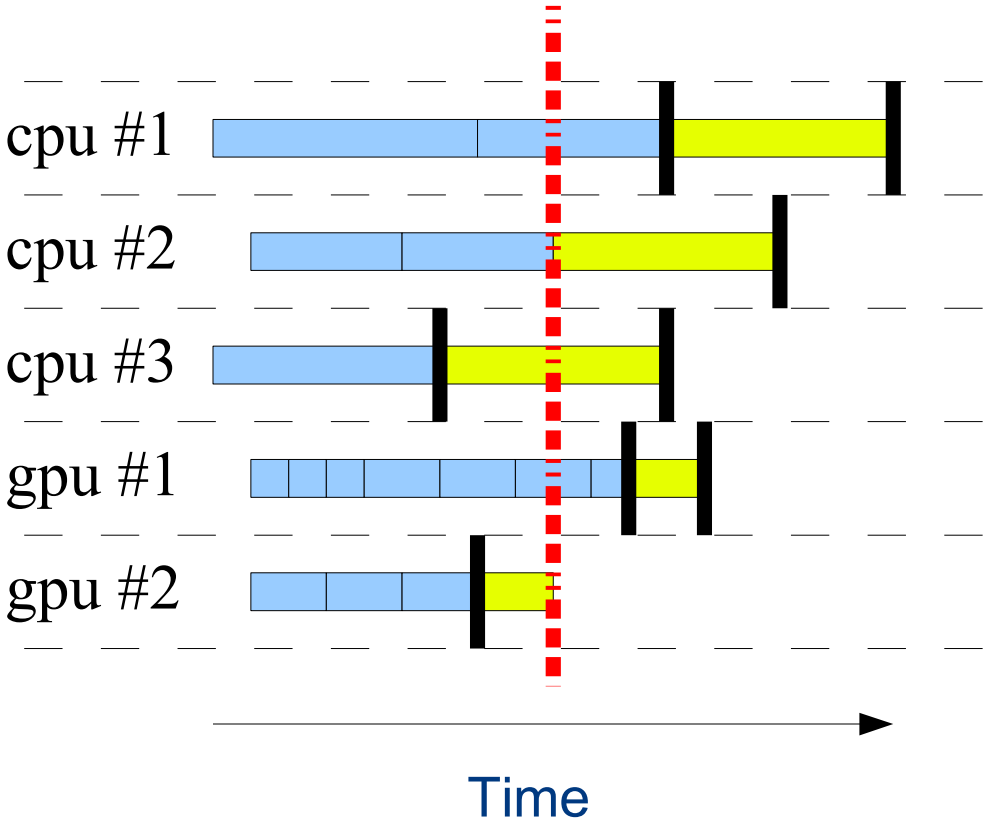
Prediction-based scheduling
Load balancing

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



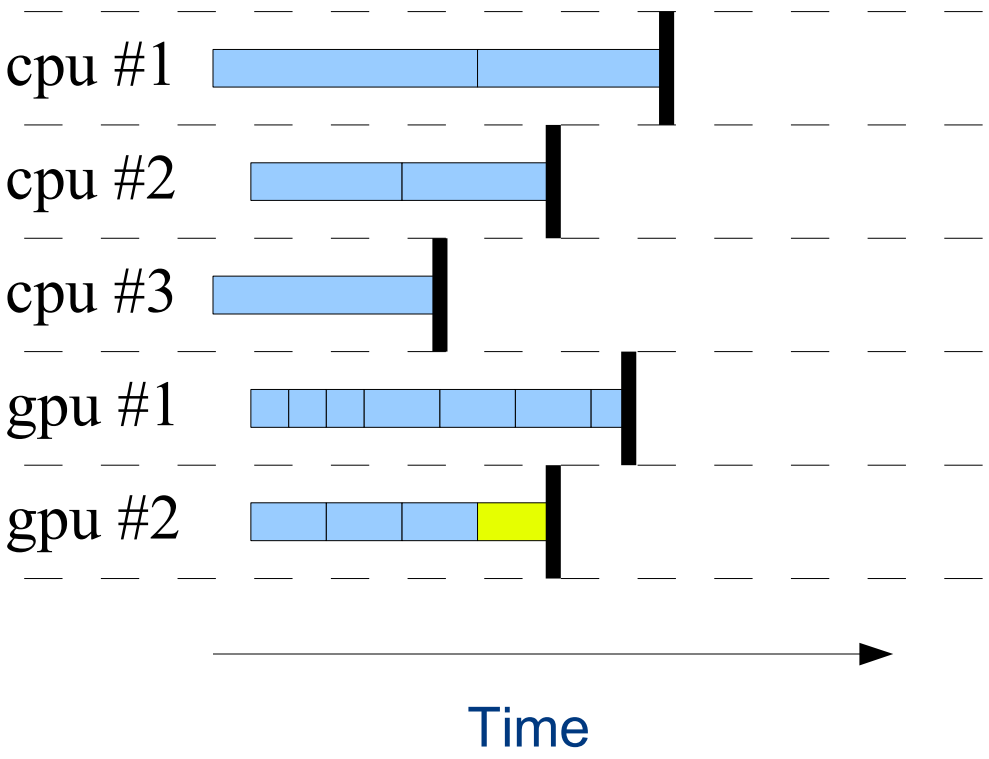
Prediction-based scheduling
Load balancing

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



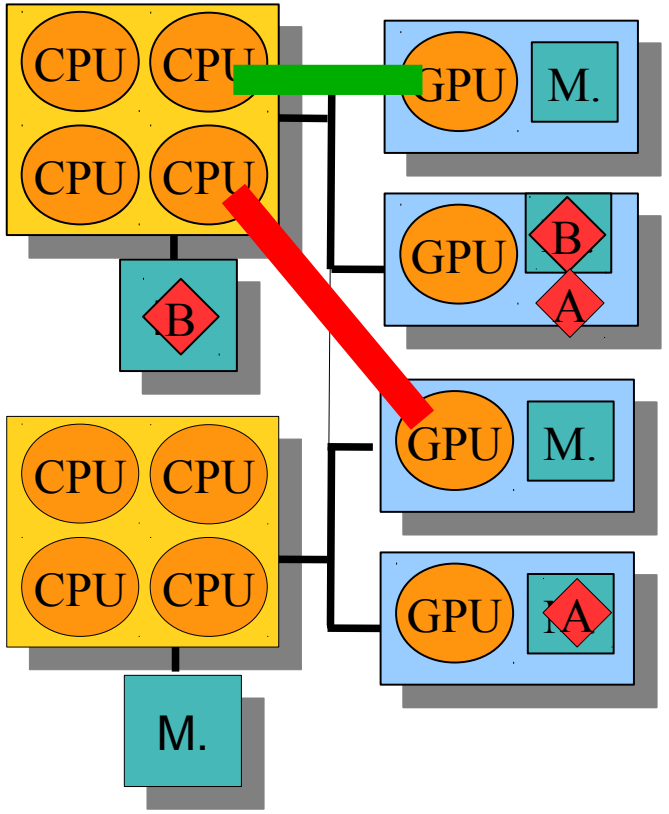
Prediction-based scheduling
Load balancing

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



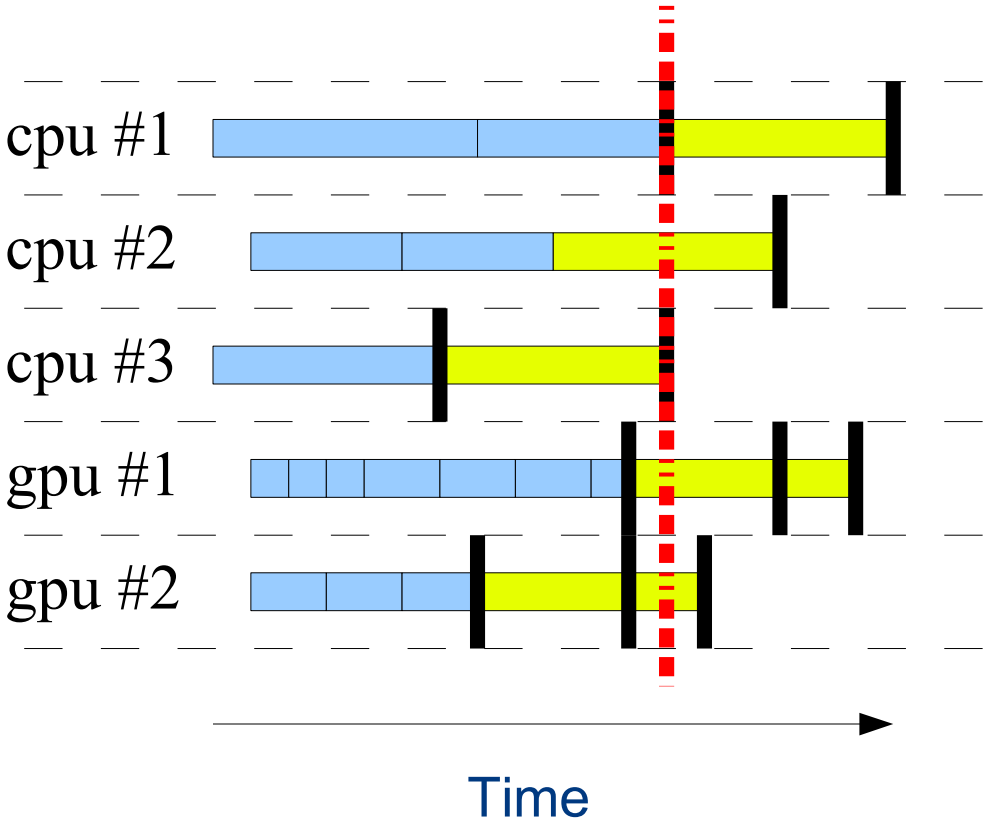
Predicting data transfer overhead
Motivations

- Hybrid platforms
  - Multicore CPUs and GPUs
  - PCI-e bus is a precious ressource
- Data locality vs. Load balancing
  - Cannot avoid all data transfers
  - Minimize them
- StarPU keeps track of
  - data replicates
  - on-going data movements



Prediction-based scheduling
Load balancing

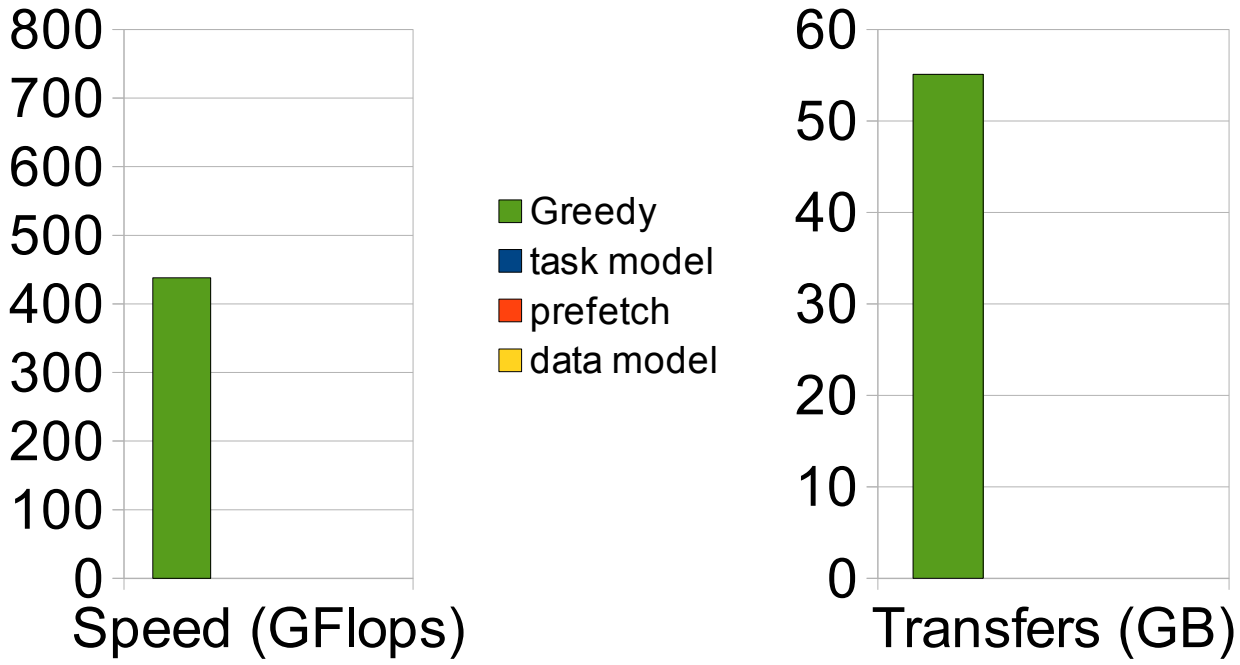
- Data transfer time
  - Sampling based on off-line calibration
- Can be used to
  - Better estimate overall exec time
  - Minimize data movements



Scheduling in a hybrid environment

Performance models

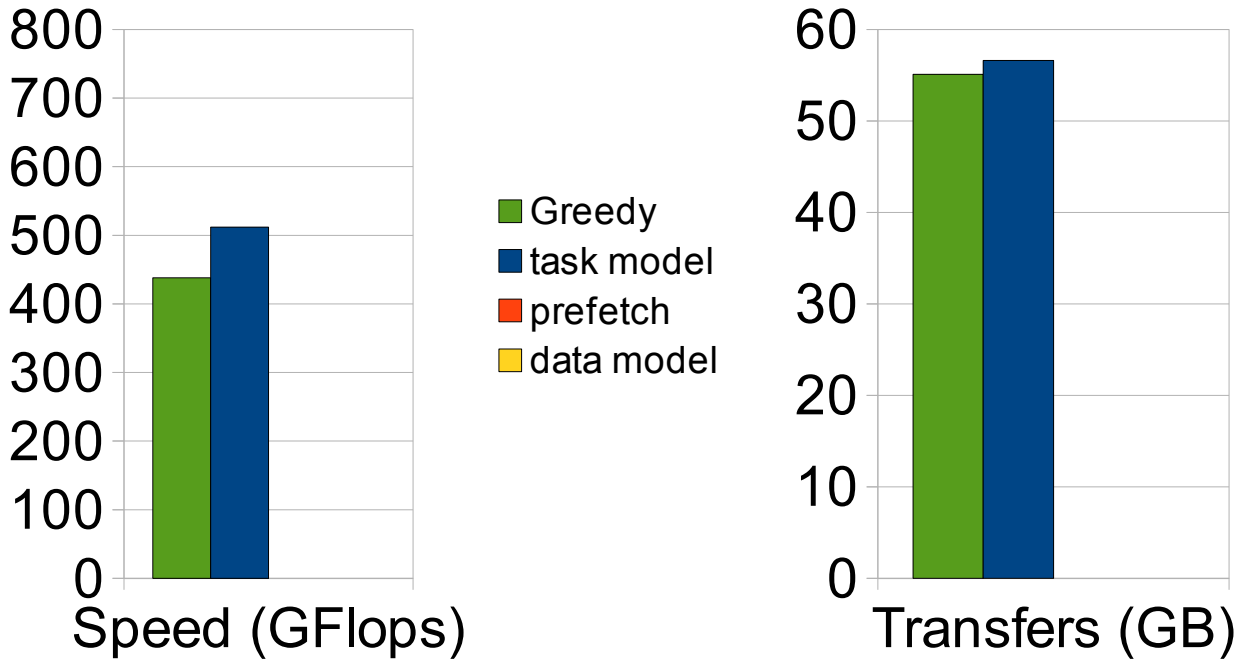
- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)



Scheduling in a hybrid environment

Performance models

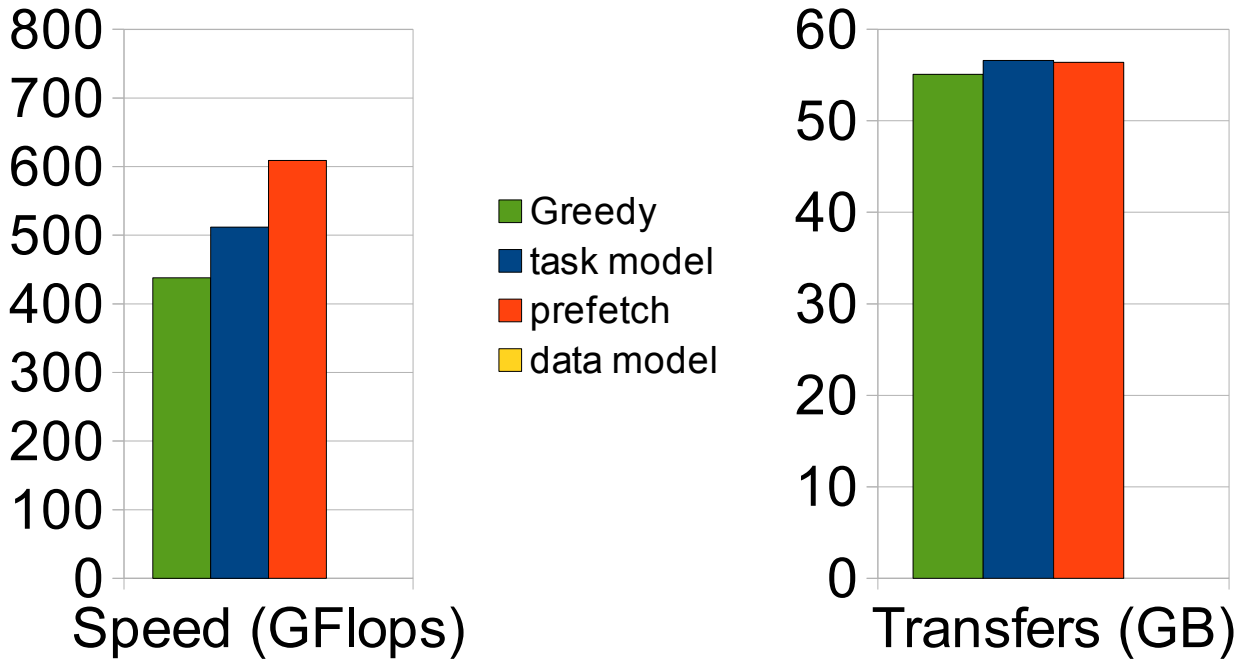
- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)



Scheduling in a hybrid environment

Performance models

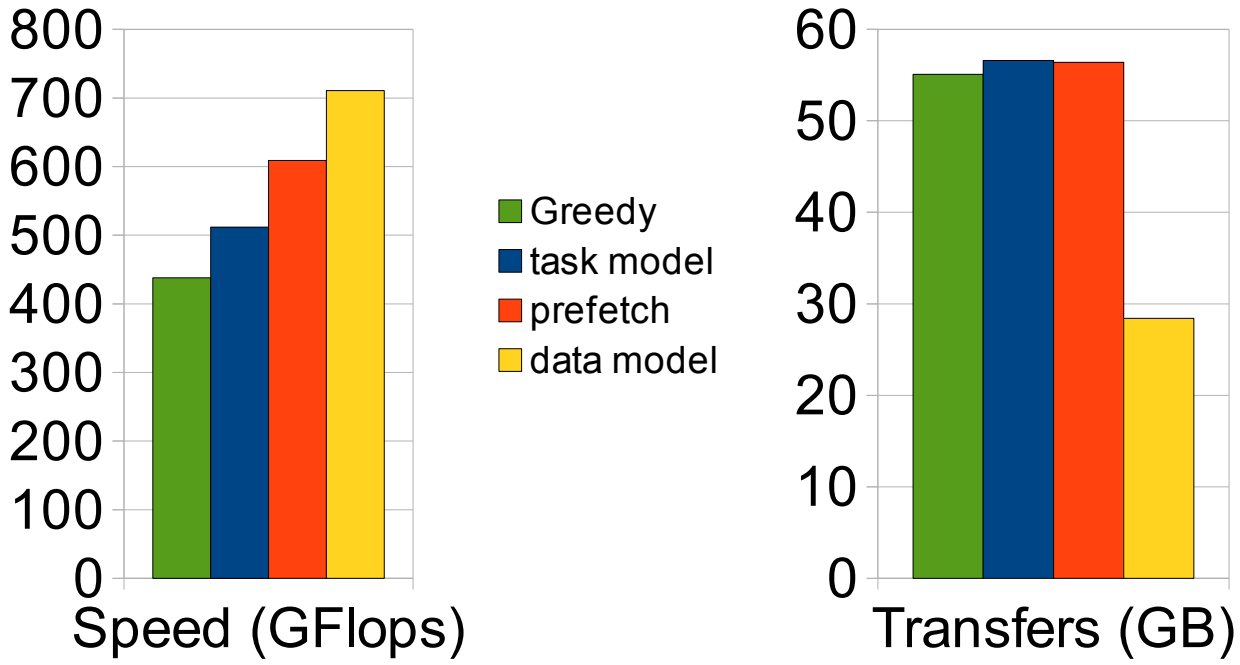
- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)



Scheduling in a hybrid environment

Performance models

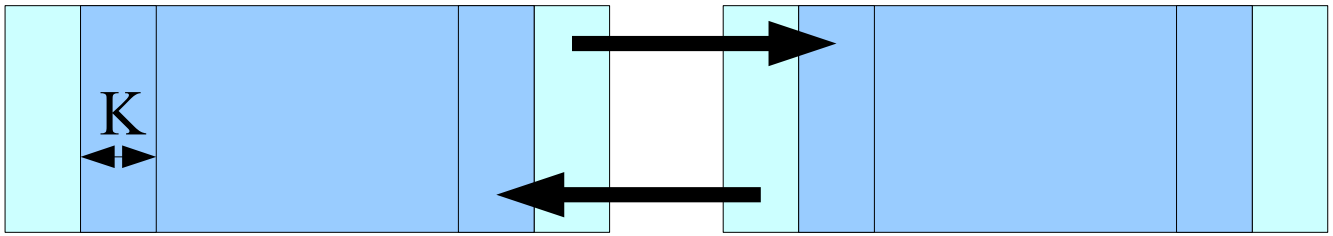
- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)



Stencil computation

Our algorithm

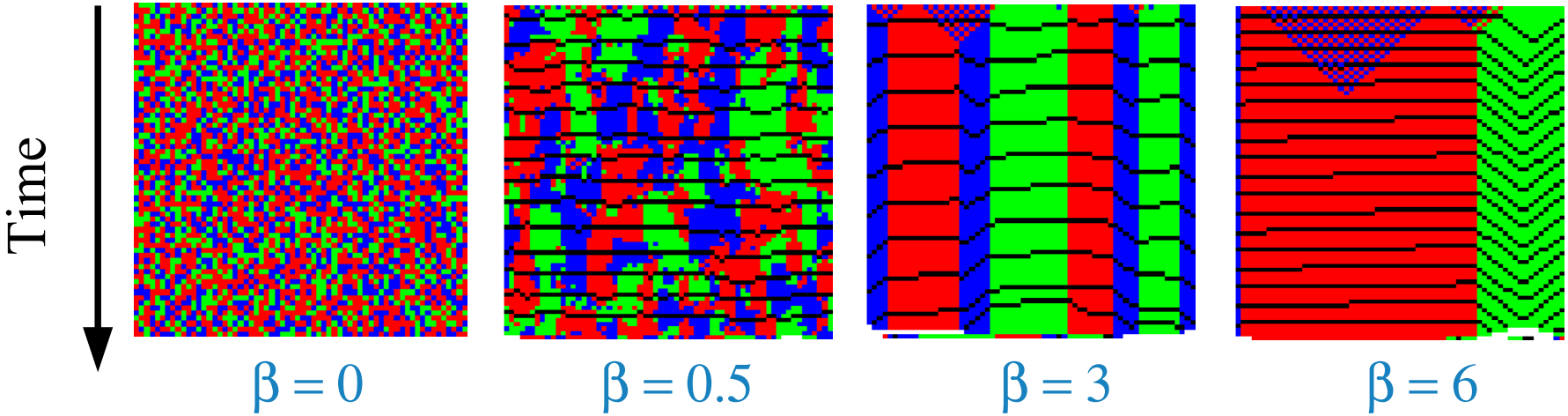
- 3D 27-point stencil kernel
  - Straighforward kernel implementation
    - CUDA + CPU
    - 3D Torus : no boundary conditions
  - Alternate two layers
- Parallelization : 1D distribution of 3D blocks
  - 2D and 3D are also doable
  - Blocks boundaries = shadow cells



Stencil computation

256 x 4096 x 4096 , 64 blocks

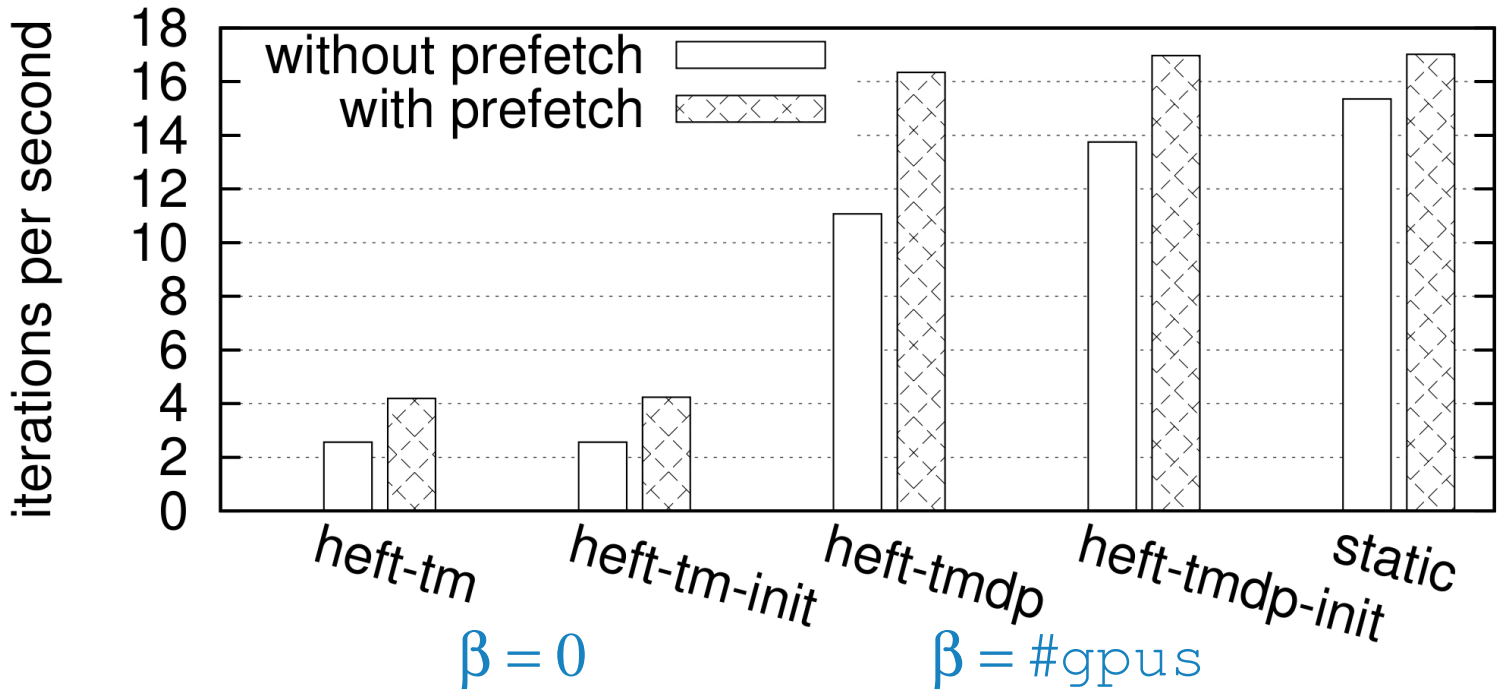
- Load balancing vs. Data locality
  - « dmda » ~ minimize (Tcompute + β Ttransfer)
  - 1 GPU = 1 color
  - Display which GPU did the computation



- Both load balancing and data locality are needed
- No need to statically map the blocks

Stencil computation

- Impact of scheduling policy
  - 3 GPUs (FX5800) – no CPU used :-(
  - 256 x 4096 x 4096 : 64 blocks



Mixing PLASMA and MAGMA with StarPU

« SPLAGMA »

Cholesky & QR decompositions

Mixing PLASMA and MAGMA with StarPU

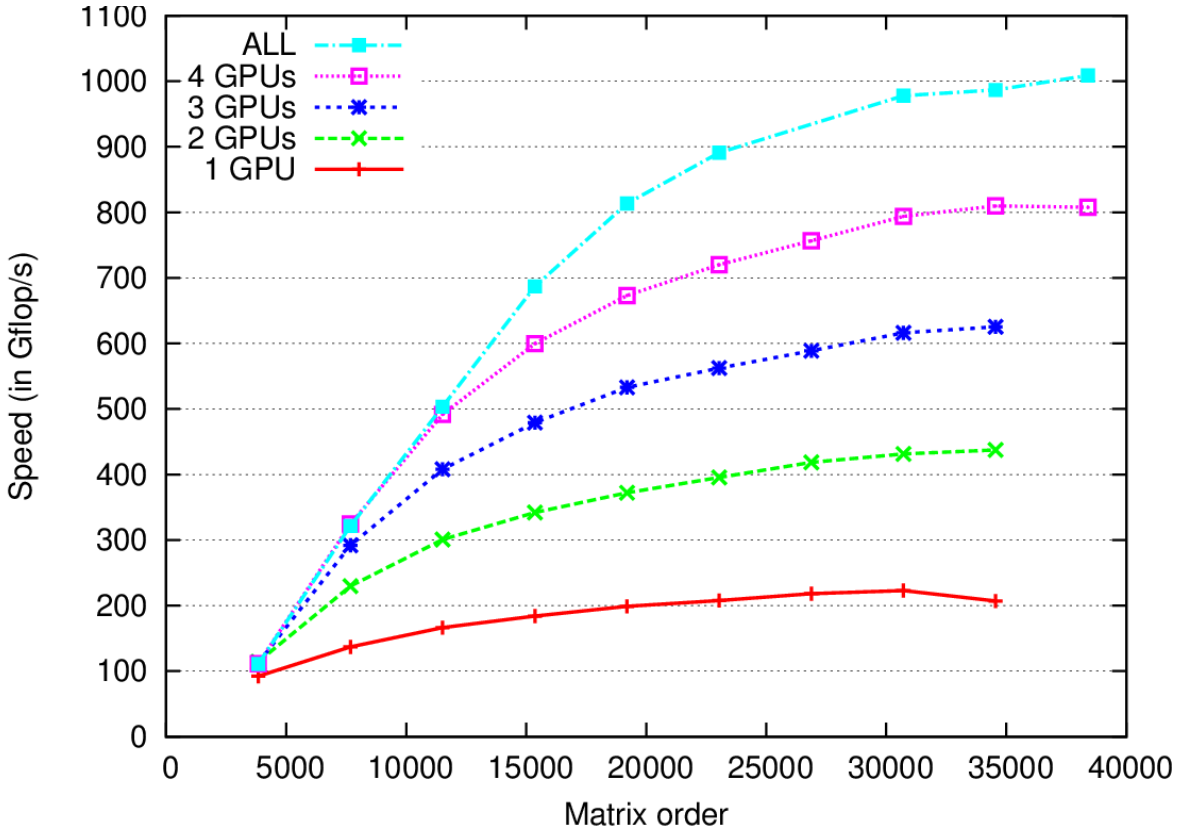
- State of the art algorithms
  - PLASMA (Multicore CPUs)
    - Dynamically scheduled with Quark
  - MAGMA (Multiple GPUs)
    - Hand-coded data transfers
    - Static task mapping

- General SPLAGMA design
  - Use PLASMA algorithm with « magnum tiles »
  - PLASMA kernels on CPUs, MAGMA kernels on GPUs
  - Bypass the QUARK scheduler

- Programmability
  - Cholesky: ~half a week
  - QR : ~2 days of works
  - Quick algorithmic prototyping

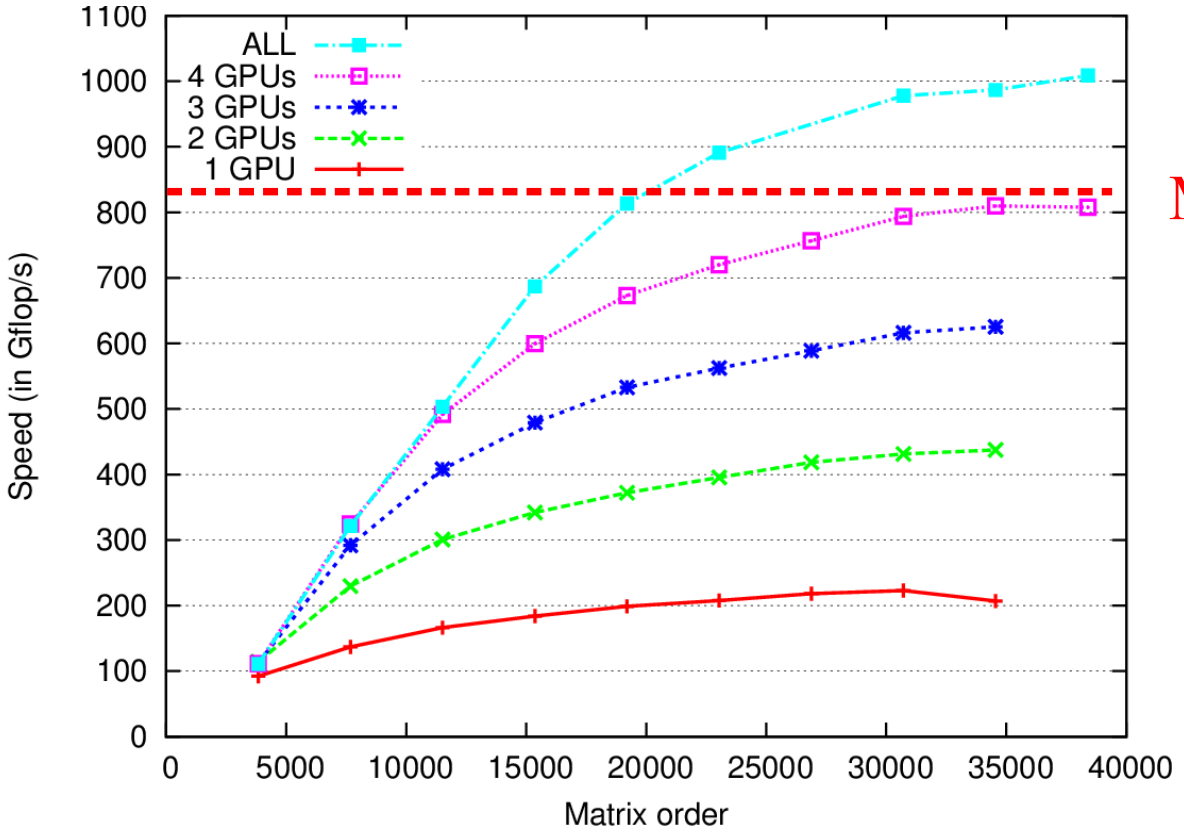
Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)



Mixing PLASMA and MAGMA with StarPU

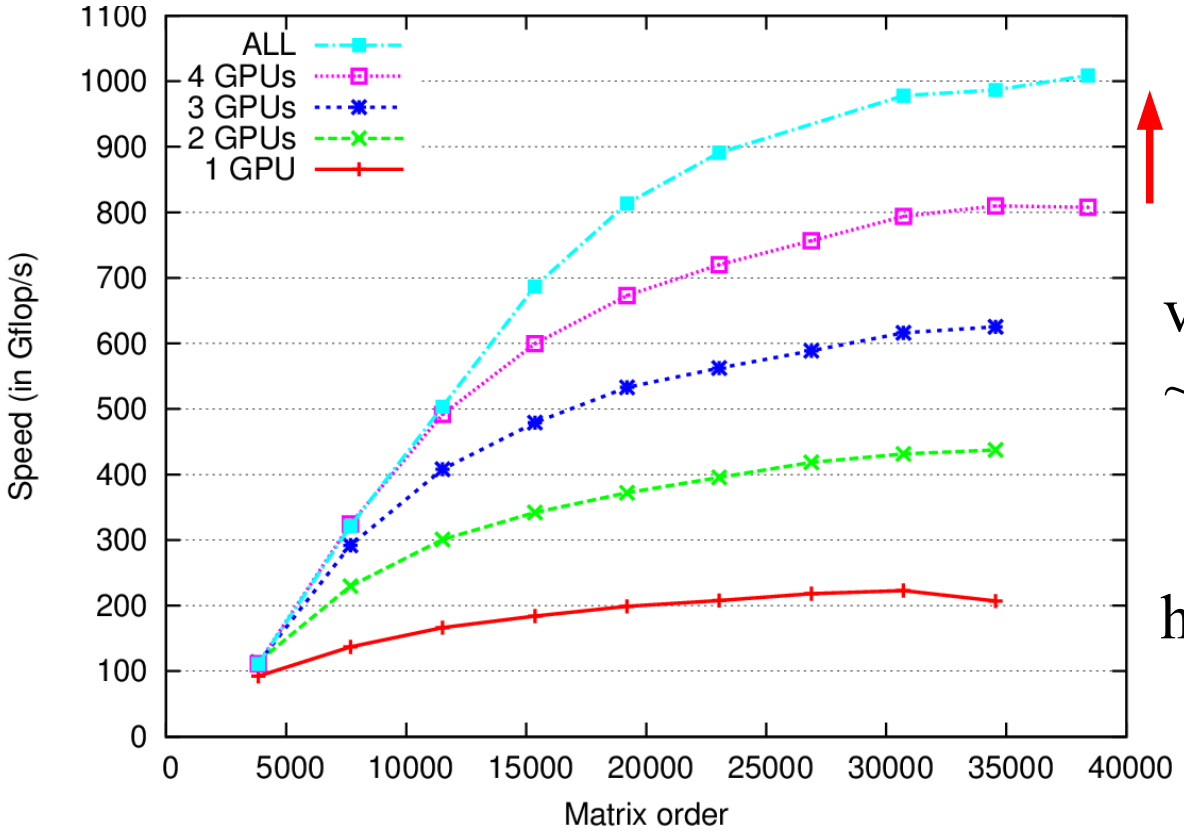
- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)



MAGMA

Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)



+12 CPUs
~200GFlops

vs theoretical
~150Gflops !

Thanks to
heterogeneity

Mixing PLASMA and MAGMA with StarPU

- « Super-Linear » efficiency in QR?
  - Kernel efficiency
    - sgeqrt
      - CPU: 9 Gflops GPU: 30 Gflops (Speedup : ~3)
    - stsqrt
      - CPU: 12Gflops GPU: 37 Gflops (Speedup: ~3)
    - somqr
      - CPU: 8.5 Gflops GPU: 227 Gflops (Speedup: ~27)
    - Sssmqr
      - CPU: 10Gflops GPU: 285Gflops (Speedup: ~28)
  - Task distribution observed on StarPU
    - sgeqrt: 20% of tasks on GPUs
    - Sssmqr: 92.5% of tasks on GPUs
  - Taking advantage of heterogeneity !
    - Only do what you are good for
    - Don't do what you are not good for

Performance analysis tools

Task distribution

```
$ STARPU_WORKER_STATS=1 ./examples/mult/sgemm
Time: 34.78 ms
GFlop/s: 24.12

Worker statistics:
******************
CUDA 0 (Quadro FX 5800)                 264 task(s)
CUDA 1 (Quadro FX 5800)                 237 task(s)
CUDA 2 (Quadro FX 5800)                 237 task(s)
CPU 0                                   177 task(s)
CPU 1                                   175 task(s)
CPU 2                                   168 task(s)
CPU 3                                   177 task(s)
```

Bus usage

```
$ STARPU_BUS_STATS=1 ./examples/mult/sgemm
Time: 35.71 ms
GFlop/s: 23.49

Data transfer statistics:
*************************
0 -> 1  2.52 MB 1.32MB/s        (transfers : 161 - avg 0.02 MB)
1 -> 0  2.39 MB 1.26MB/s        (transfers : 153 - avg 0.02 MB)
0 -> 2  3.12 MB 1.64MB/s        (transfers : 200 - avg 0.02 MB)
2 -> 0  3.00 MB 1.58MB/s        (transfers : 192 - avg 0.02 MB)
0 -> 3  3.03 MB 1.59MB/s        (transfers : 194 - avg 0.02 MB)
3 -> 0  2.91 MB 1.53MB/s        (transfers : 186 - avg 0.02 MB)
Total transfers: 16.97 MB
```

Energy consumption

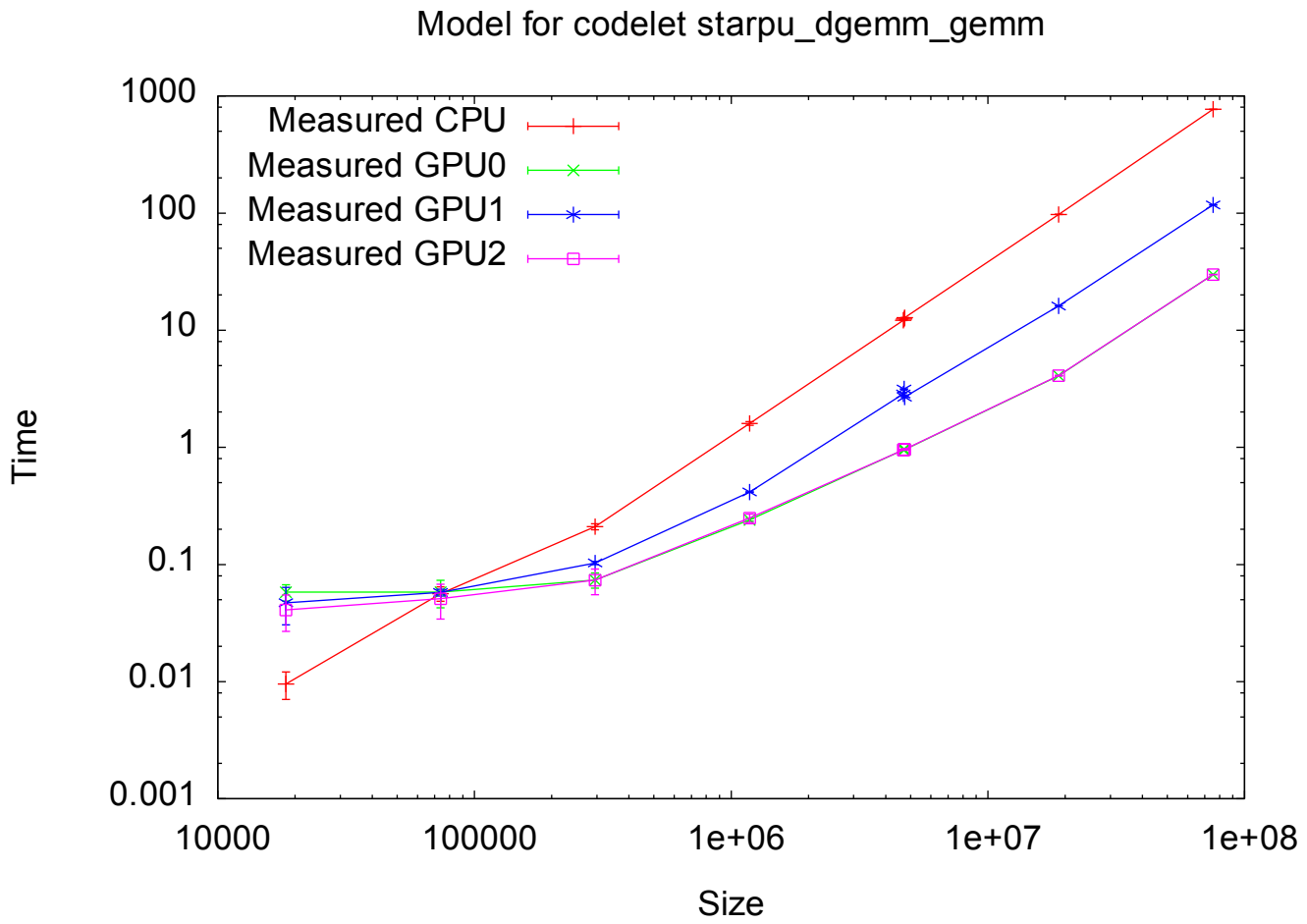
```
$ STARPU_WORKER_STATS=1 STARPU_PROFILING=1 ./examples/stencil/stencil
OpenCL 0 (Quadro FX 5800)
773 task(s)
total: 409.60 ms executing: 340.51 ms sleeping: 0.00
5040.000000 J consumed
OpenCL 1 (Quadro FX 5800)
767 task(s)
total: 409.62 ms executing: 346.28 ms sleeping: 0.00
10280.000000 J consumed
OpenCL 2 (Quadro FX 5800)
756 task(s)
total: 409.63 ms executing: 343.72 ms sleeping: 0.00
14880.000000 J consumed
```

Performance models

```
$ starpu_perfmodel_display -l
file: <starpu_sgemm_gemm>
$ starpu_perfmodel_display -s starpu_sgemm_gemm
performance model for cpu
# hash         size            mean            dev             n
880805ba       49152           1.233333e+02    1.063576e+01    1612
8bd4e11d       2359296         1.331984e+04    6.971079e+02    635
performance model for cuda_0
# hash         size            mean            dev             n
880805ba       49152           2.743658e+01    2.178427e+00    496
8bd4e11d       2359296         6.207991e+02    6.941988e+00    307
```

Performance models plot

```
$ starpu_perfmodel_plot -s starpu_dgemm_gemm
$ gnuplot starpu_dgemm_gemm.gp
```

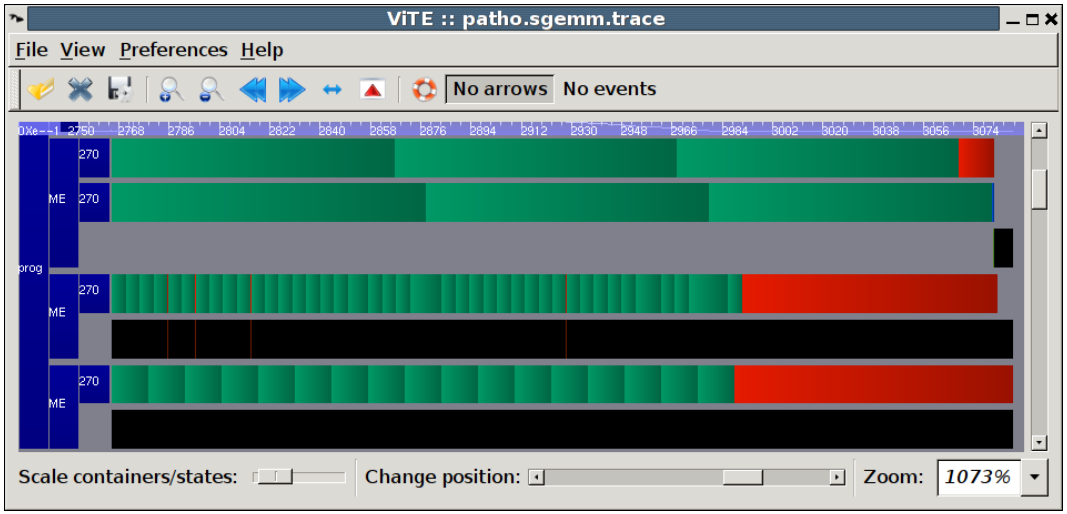


Offline performance analysis

Visualize execution traces

- Generate a Pajé trace
  - A file of the form /tmp/prof_file_user_<your login> should have been created
  - Call fxt_tool -i /tmp/prof_file_user_yourlogin
    - A paje.trace file should be generated in current directory

- Vite trace visualization tool
  - Freely available from http://vite.gforge.inria.fr/ (open source !)
  - vite paje.trace



2 Xeon cores

Quadro FX5800

Quadro FX4600

Experimental features

Reduction mode

- Contribution from a series of tasks into a single buffer
  - e.g. Dot product, Matrix multiplication, Histogram, …

- New data access mode: REDUX
  - Similar to OpenMP's reduce() keyword
  - Looks like R/W mode from the point of view of tasks
  - Tasks actually access transparent per-PU buffer
    - initialized by user-provided “init” function
  - User-provided “reduction” function used to reduce into single buffer when switching back to R or R/W mode.
    - Can be optimized according to machine architecture

- Preliminary results: x3 acceleration on Conjugate Gradiant application

How about MPI + StarPU?

- Save programmers the burden of rewriting their MPI code
  - Keep the same MPI flow
  - Work on StarPU data instead of plain data buffers.
- StarPU provides support for sending data over MPI
  - starpu_mpi_send/recv, isend/irecv, ...
    - Equivalents of MPI_Send/Recv, Isend/Irecv,... but working on StarPU data
    - Plus _submit versions
  - Automatically handles all needed CPU/GPU transfers
  - Automatically handles task/communications dependencies
  - Automatically overlaps MPI communications, CPU/GPU communications, and CPU/GPU computations
    - Thanks to the data transfer requests mechanism

MPI ping-pong example

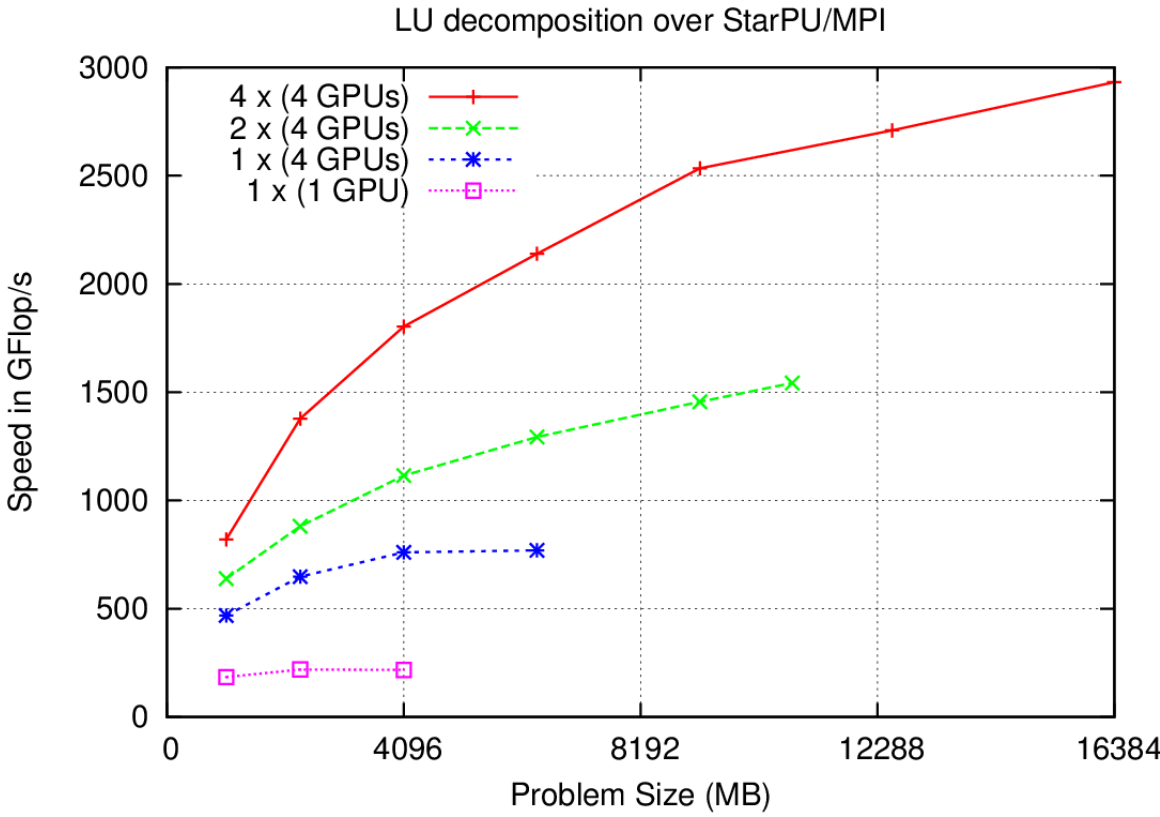
```
for (loop = 0 ; loop < NLOOPS; loop++) {
        if ( !(loop == 0 && rank == 0))
            MPI_Recv(&data, prev_rank, …) ;

        increment(&data);

        if ( !(loop == NLOOPS-1 && rank == size-1))
            MPI_Send(&data, next_rank, …) ;
}
```

StarPU-MPI ping-pong example

```
for (loop = 0 ; loop < NLOOPS; loop++) {
        if ( !(loop == 0 && rank == 0))
            starpu_mpi_irecv_submit(data_handle, prev_rank, …) ;

        task = starpu_task_create() ;
        task->cl = &increment_codelet ;
        task->buffers[0].handle = data_handle ;
        task->buffers[0].mode = STARPU_RW ;
        starpu_task_submit(task) ;

        if ( !(loop == NLOOPS-1 && rank == size-1))
            starpu_mpi_isend_submit(data_handle, next_rank, …) ;
}
starpu_task_wait_for_all() ;
```

MPI results with LU

- LU decomposition
  - MPI+multiGPU
  - 4 x 4 GPUs (GT200)
- Static MPI distribution
  - 2D block cyclic
  - ~SCALAPACK
  - No pivoting !



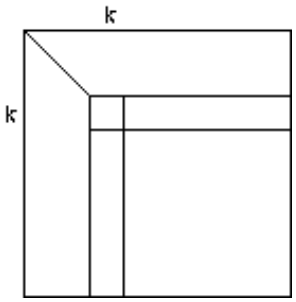
- Currently porting UTK's MAGMA + PLASMA

MPI version of starpu_insert_task
MPI VSM

- Data distribution over MPI nodes decided by application
- But data coherency extended to the MPI level
  - Automatic starpu_mpi_send/recv calls for each task
- Similar to a DSM, but granularity is whole data and whole task
- All nodes execute the whole algorithm
- Actual task distribution according to data being written to

Sequential-looking code !

MPI version of starpu_insert_task
MPI VSM – 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();
```

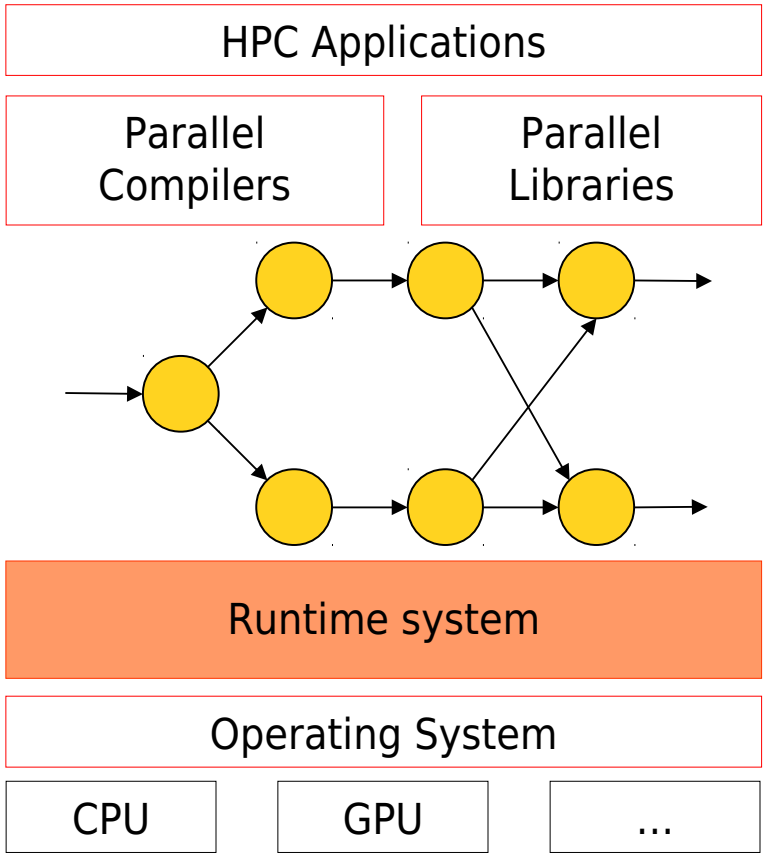
2nd hands-on session

Conclusion

Summary

- StarPU
  - Freely available under LGPL
- Task Scheduling
  - Required on hybrid platforms
  - Performance modeling
    - Tasks and data transfer
  - Results very close to hand-tuned scheduling
- Used for various computations
  - Cholesky, QR, LU, FFT, stencil, Gradient Conjugate,...

http://starpu.gforge.inria.fr



Conclusion

Future work

- Granularity is a major concern
  - Finding the optimal block size ?
    - Offline parameters auto-tuning
    - Dynamically adapt block size
  - Parallel CPU tasks
    - OpenMP, TBB, PLASMA // tasks
    - How to dimension parallel sections ?
  - Divisible tasks
    - Who decides to divide tasks ?

http://starpu.gforge.inria.fr/

Conclusion

Future work

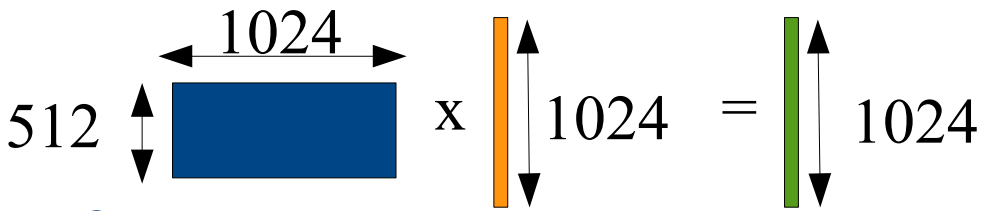
- Granularity is a major concern
  - Finding the optimal block size ?
    - Offline parameters auto-tuning
    - Dynamically adapt block size
  - Parallel CPU tasks
    - OpenMP, TBB, PLASMA // tasks
    - How to dimension parallel sections ?
  - Divisible tasks
    - Who decides to divide tasks ?

Thanks for your attention !

http://starpu.gforge.inria.fr/

Performance Models
Our History-based proposition

- Hypothesis
  - Regular applications
  - Execution time independent from data content
    - Static Flow Control
- Consequence
  - Data description fully characterizes tasks
  - Example: matrix-vector product



    - **Unique** Signature : ((1024, 512), 1024, 1024)
    - Per-data signature
      - CRC(1024, 512) = 0x951ef83b
    - Task signature
      - CRC(CRC(1024, 512), CRC(1024), CRC(1024)) = 0x79df36e2

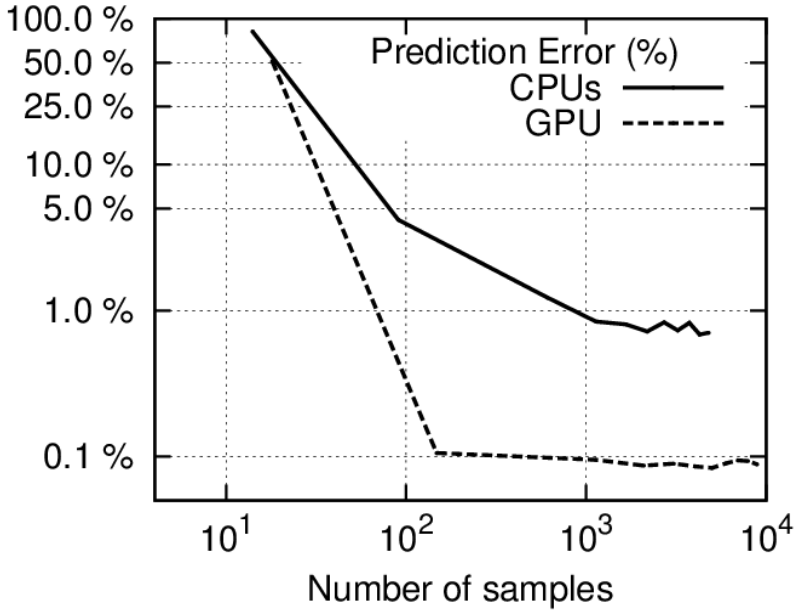
Performance Models
Our History-based proposition

- Generalization is easy
  - Task f(D1, … , Dn)
  - Data
    - Signature(Di) = CRC(p1, p2, … , pk)
  - Task ~ Series of data
    - Signature(D1, ..., Dn) = CRC(sign(D1), ..., sign(Dn))
- Systematic method
  - Problem independent
  - Transparent for the programmer
  - Efficient

Evaluation

Example: LU decomposition

	Speed (GFlop/s)	
	(16k x 16k)	(30k x 30k)
ref.	89.98 ±2.97	130.64 ±1.66
1st iter	48.31	96.63
2nd iter	103.62	130.23
3rd iter	103.11	133.50
≥ 4 iter	**103.92 ±0.46**	**135.90 ±0.64**



- Faster
- No code change !
- More stable

- Dynamic calibration
- Simple, but accurate