Introduction to StarPU

Samuel Thibault, Olivier Aumage

INRIA STORM Team
EXA2PRO H2020 FETHPC project

Overview of StarPU

Overview of StarPU

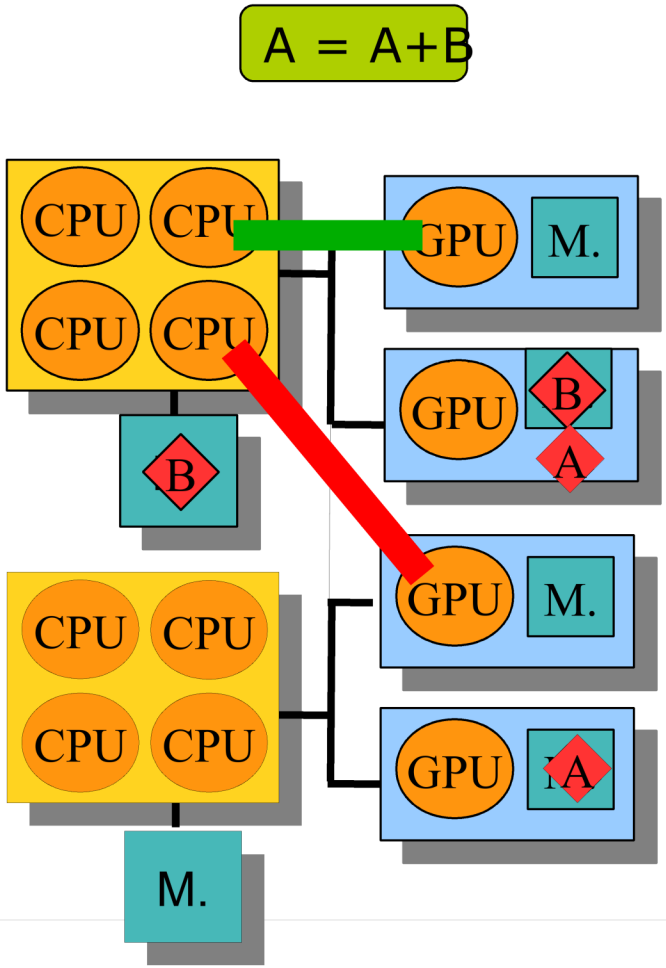
Rationale

Task scheduling

- Dynamic
- On all kinds of PU
  - General purpose
  - Accelerators/specialized

Memory transfer

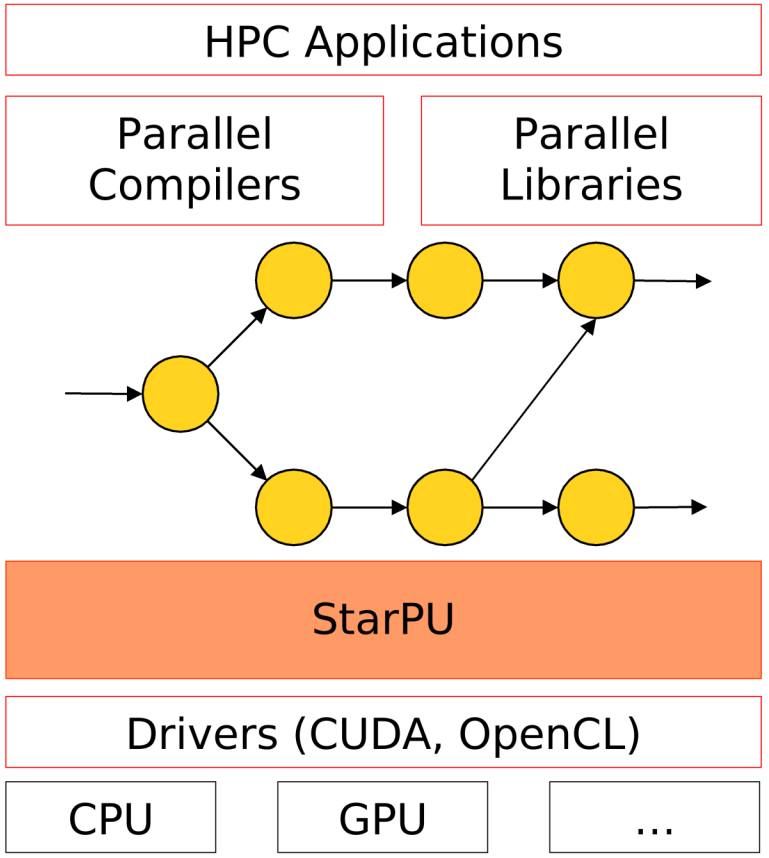
- Eliminate redundant transfers
- Software VSM (Virtual Shared Memory)



The StarPU runtime system

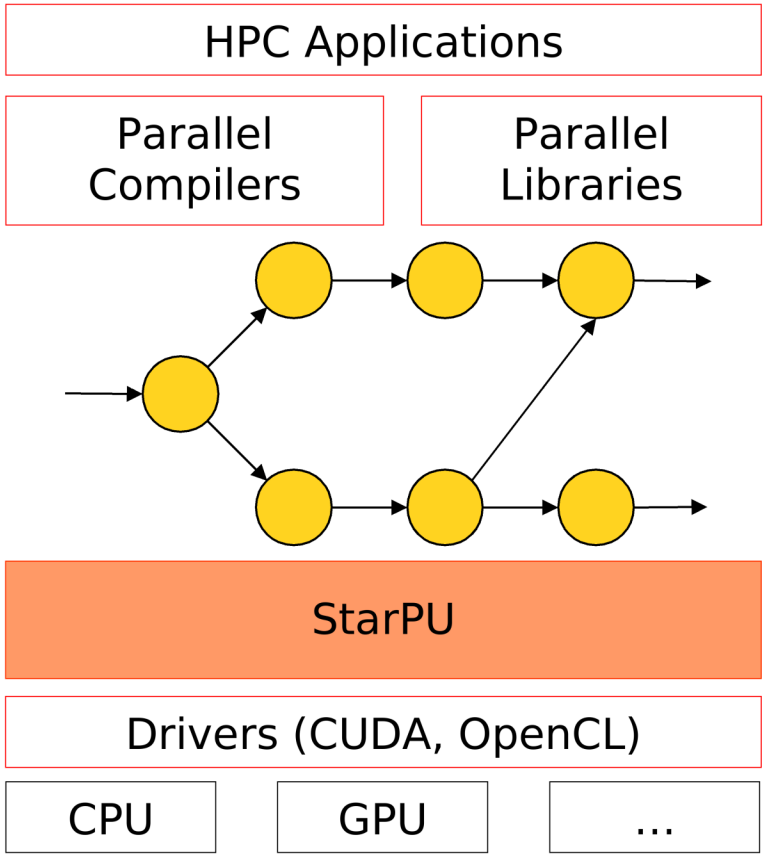
The need for runtime systems

- "do dynamically what can't be done statically anymore"
- Compilers and libraries generate (graphs of) tasks
  - Additional information is welcome!
- StarPU provides
  - Task scheduling
  - Memory management



Data management

- StarPU provides a Virtual Shared Memory (VSM) subsystem
  - Replication
  - Consistency
  - Single writer
    - Or reduction, ...
- Input & ouput of tasks = reference to VSM data

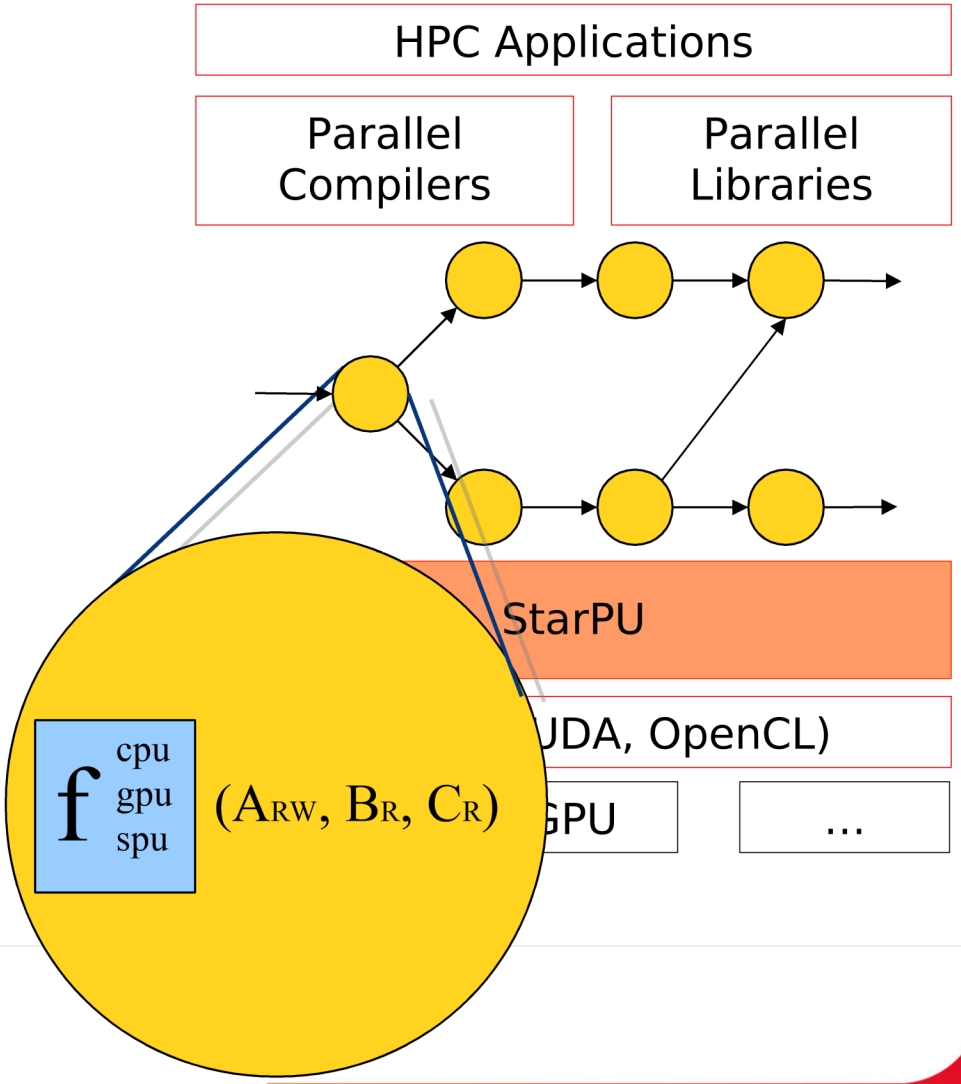


The StarPU runtime system

Task scheduling

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

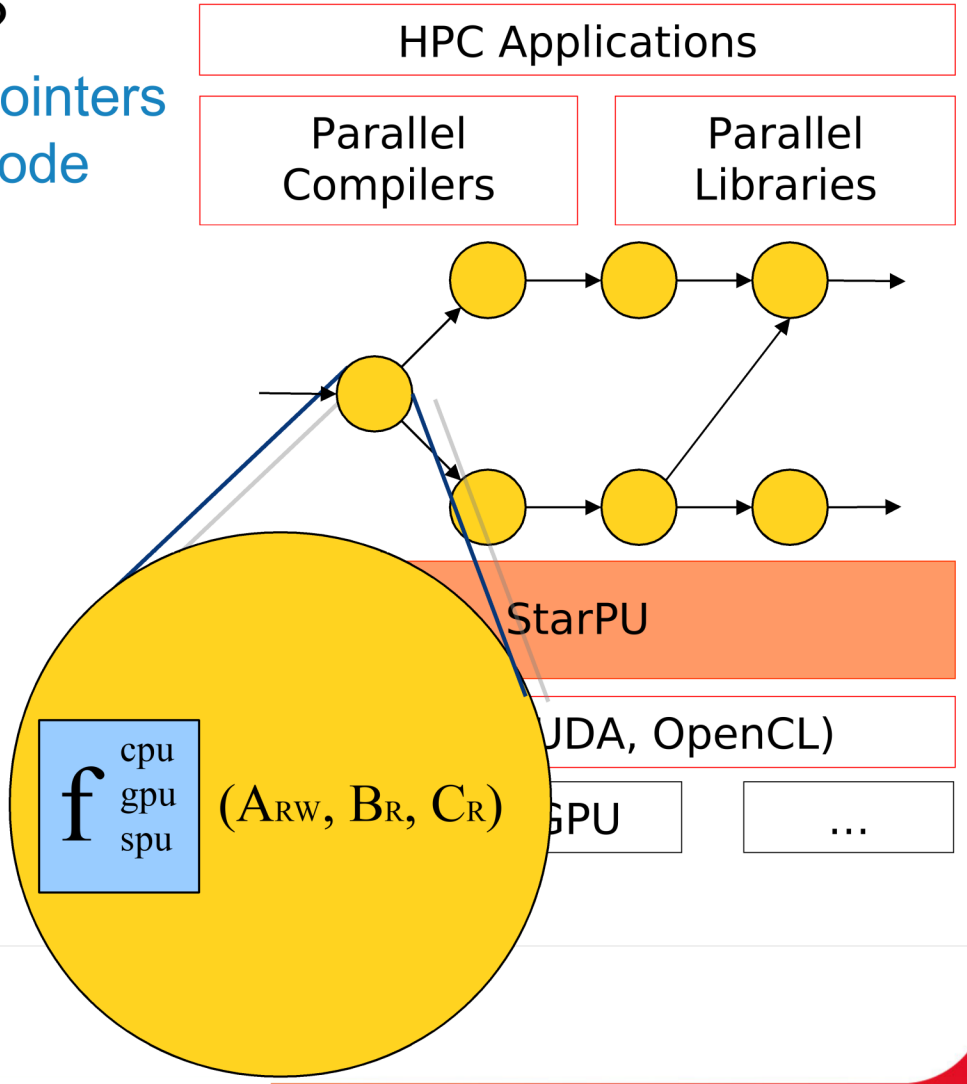
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



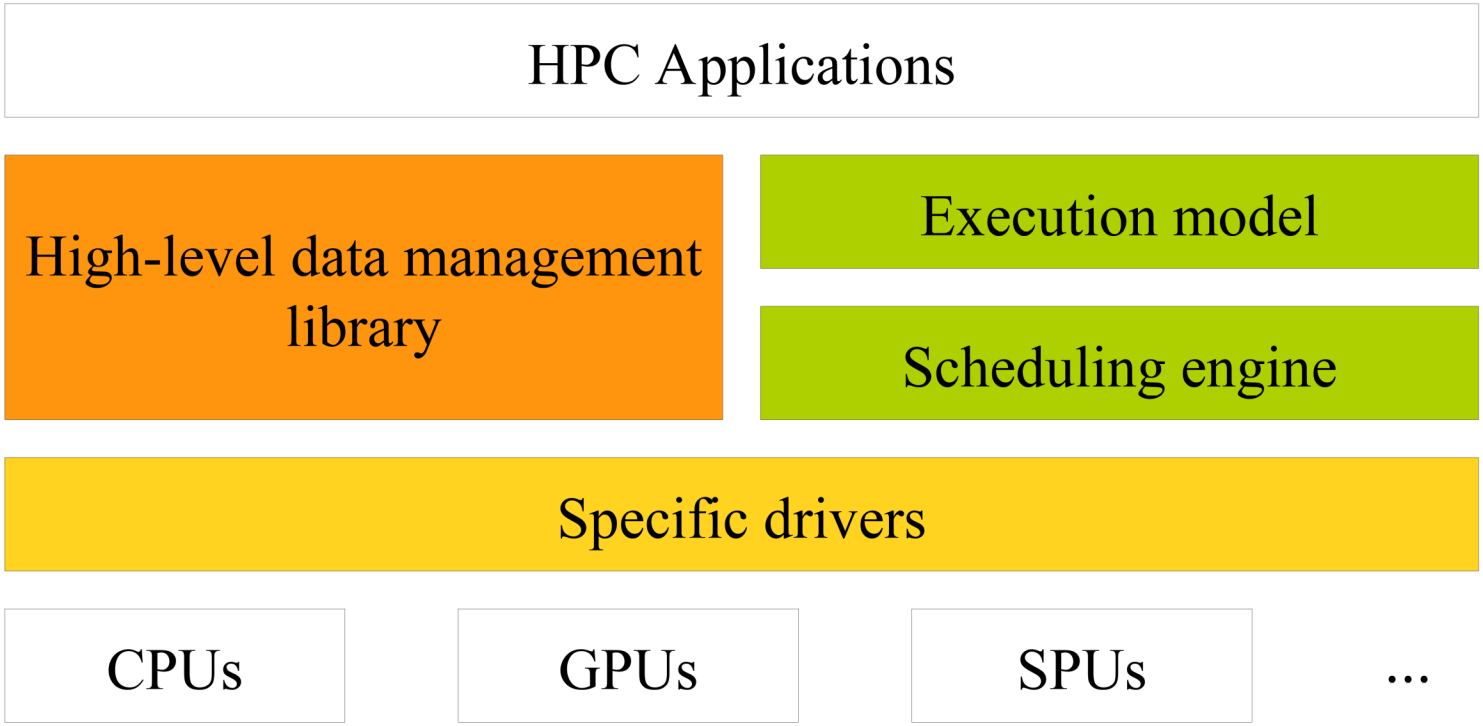
The StarPU runtime system

Task scheduling

- Who generates the code ?
  - StarPU Task ~= function pointers
  - StarPU doesn't generate code
- Libraries era
  - PLASMA + MAGMA
  - FFTW + CUFFT…
  - Variants management
- Rely on compilers



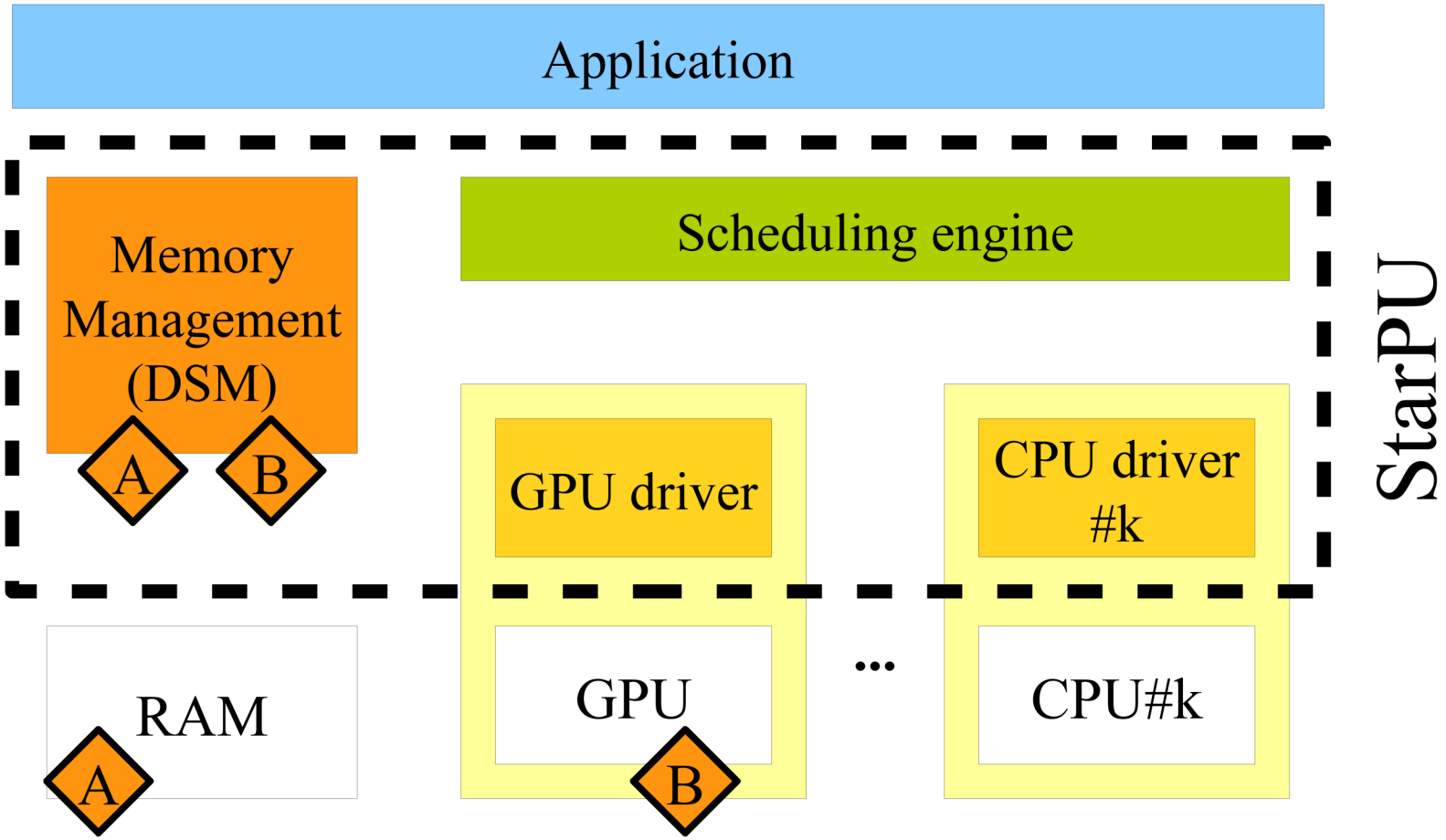
The StarPU runtime system



Mastering CPUs, GPUs, SPUs … ***PUs → StarPU**

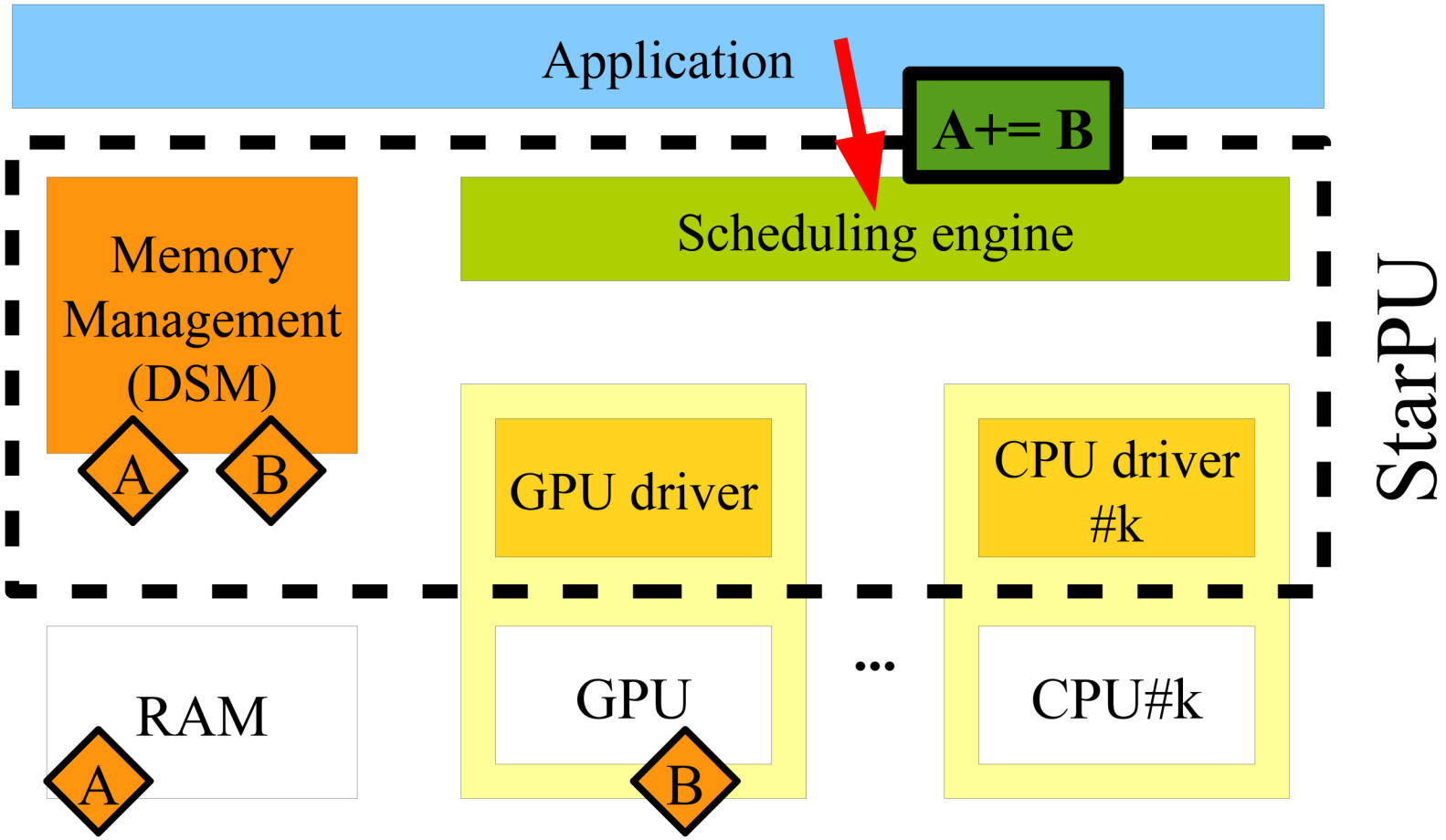
The StarPU runtime system

Execution model



The StarPU runtime system

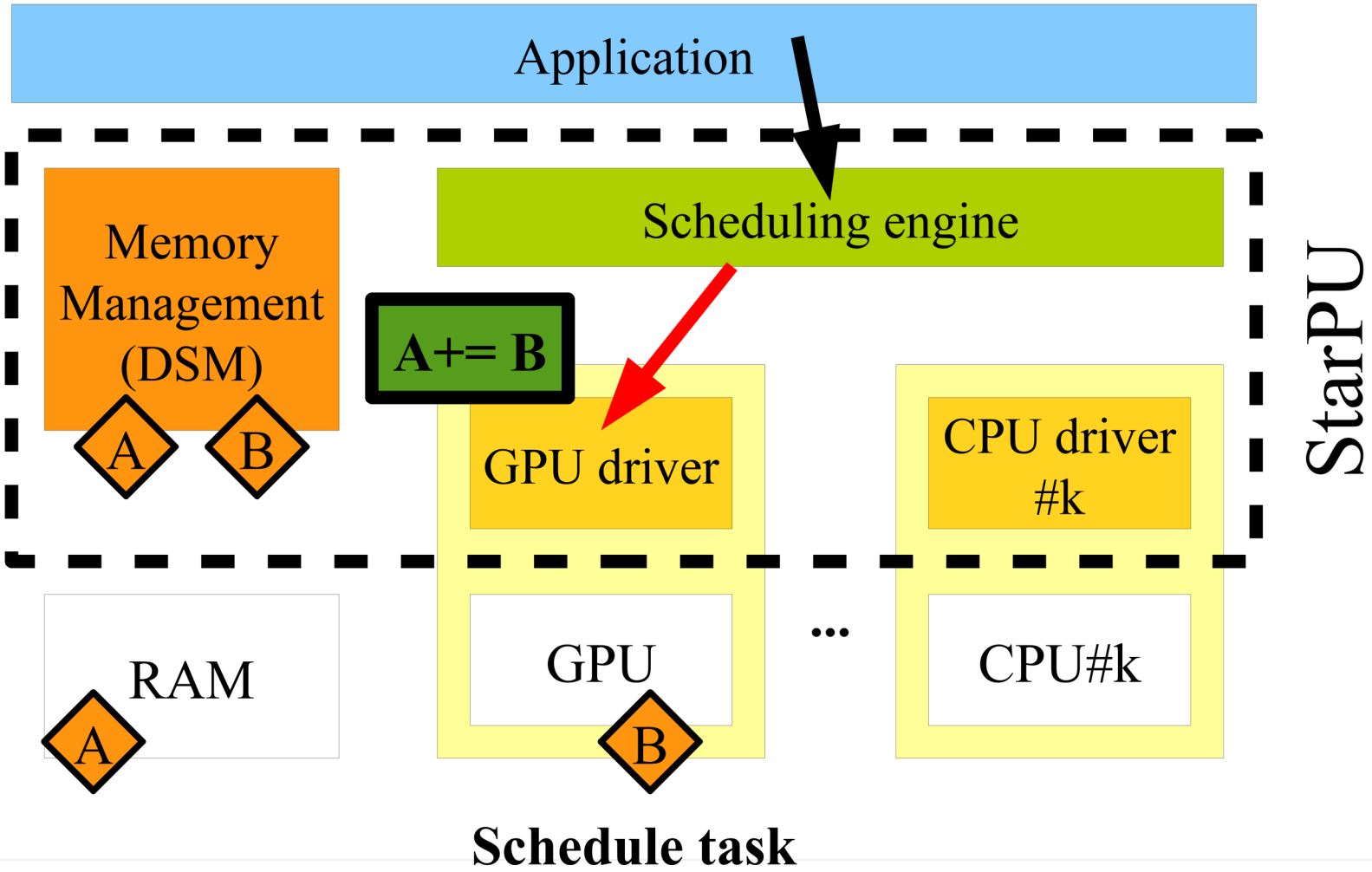
Execution model



Submit task « A += B »

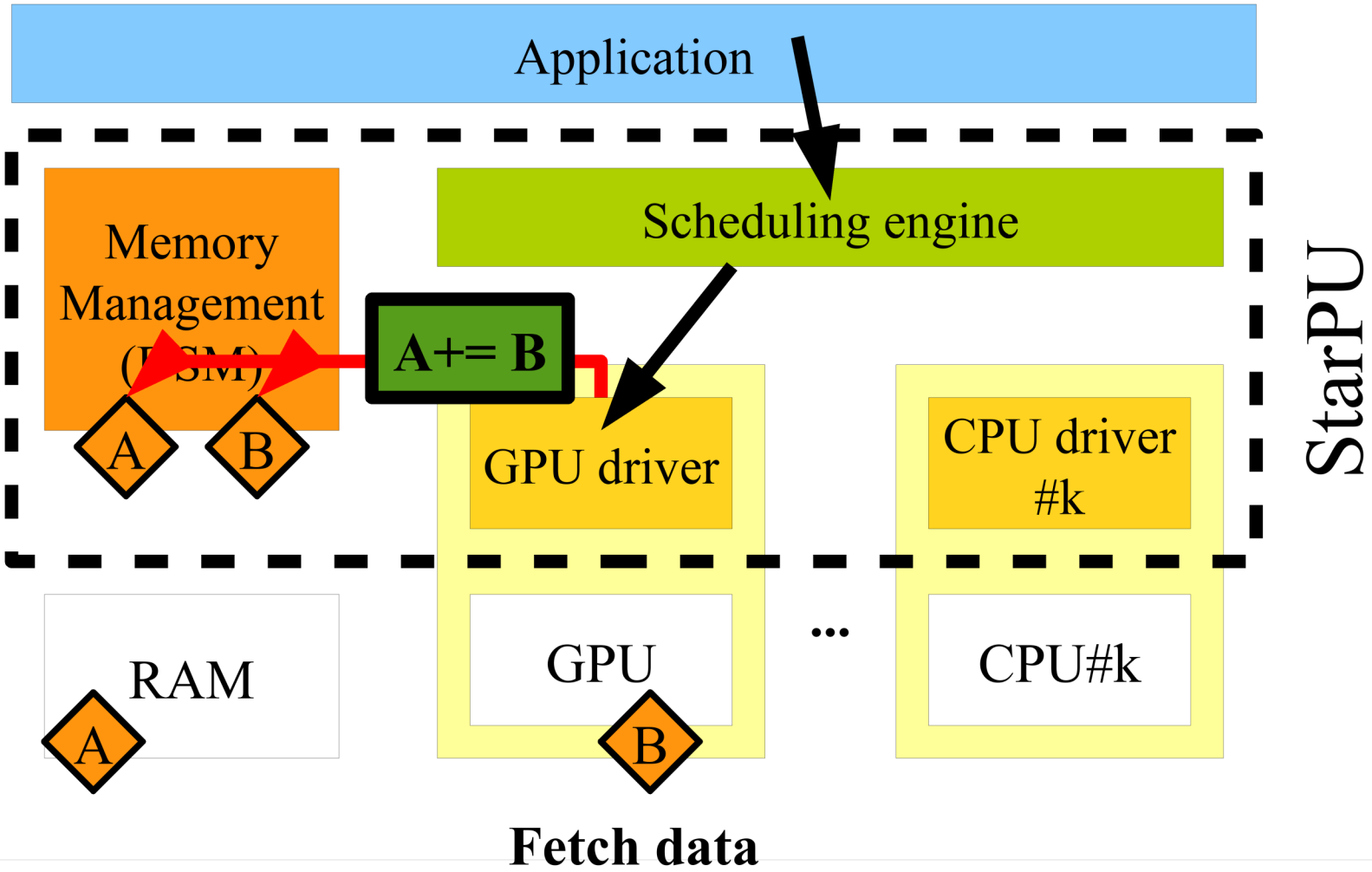
The StarPU runtime system

Execution model



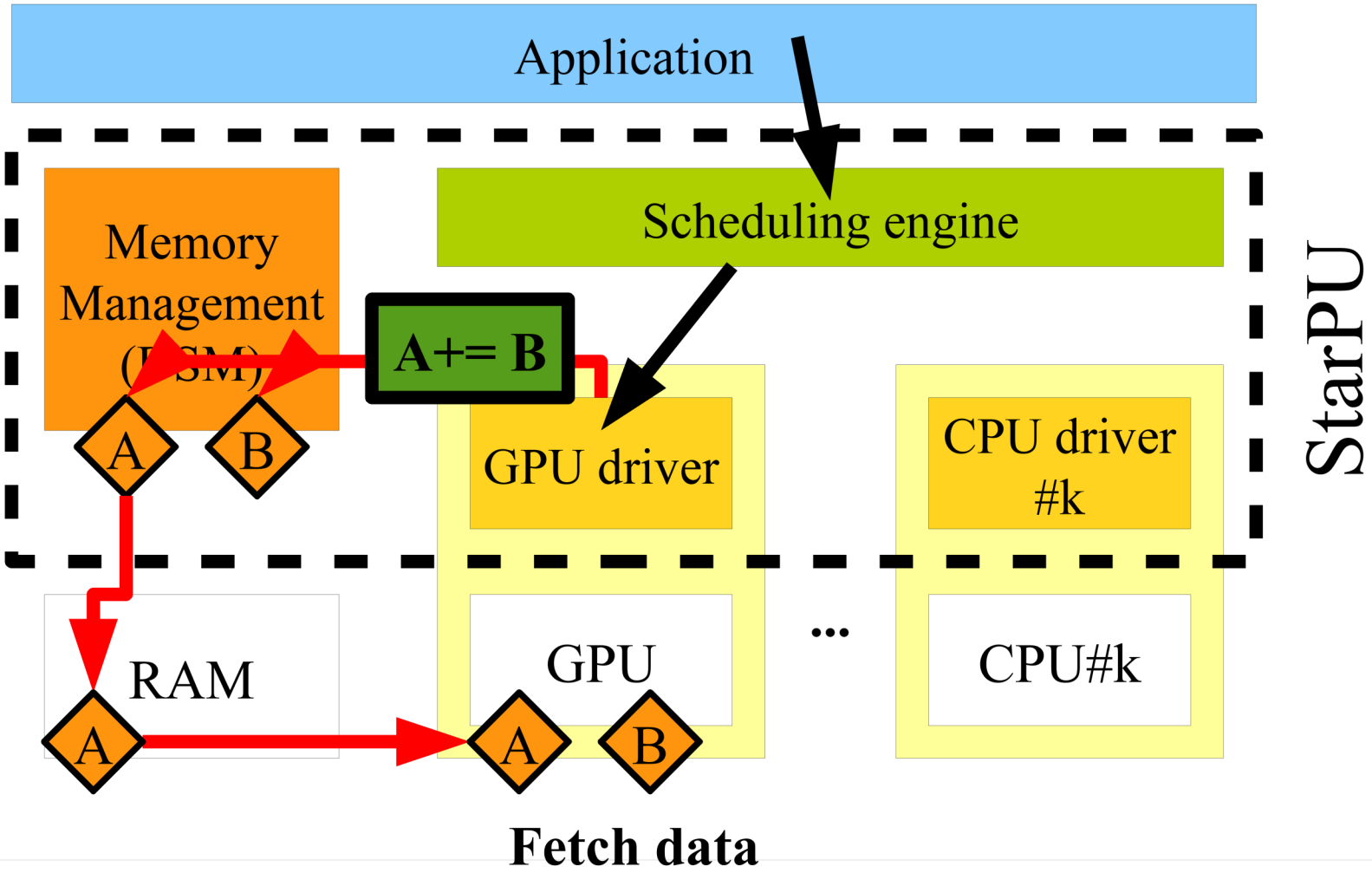
The StarPU runtime system

Execution model



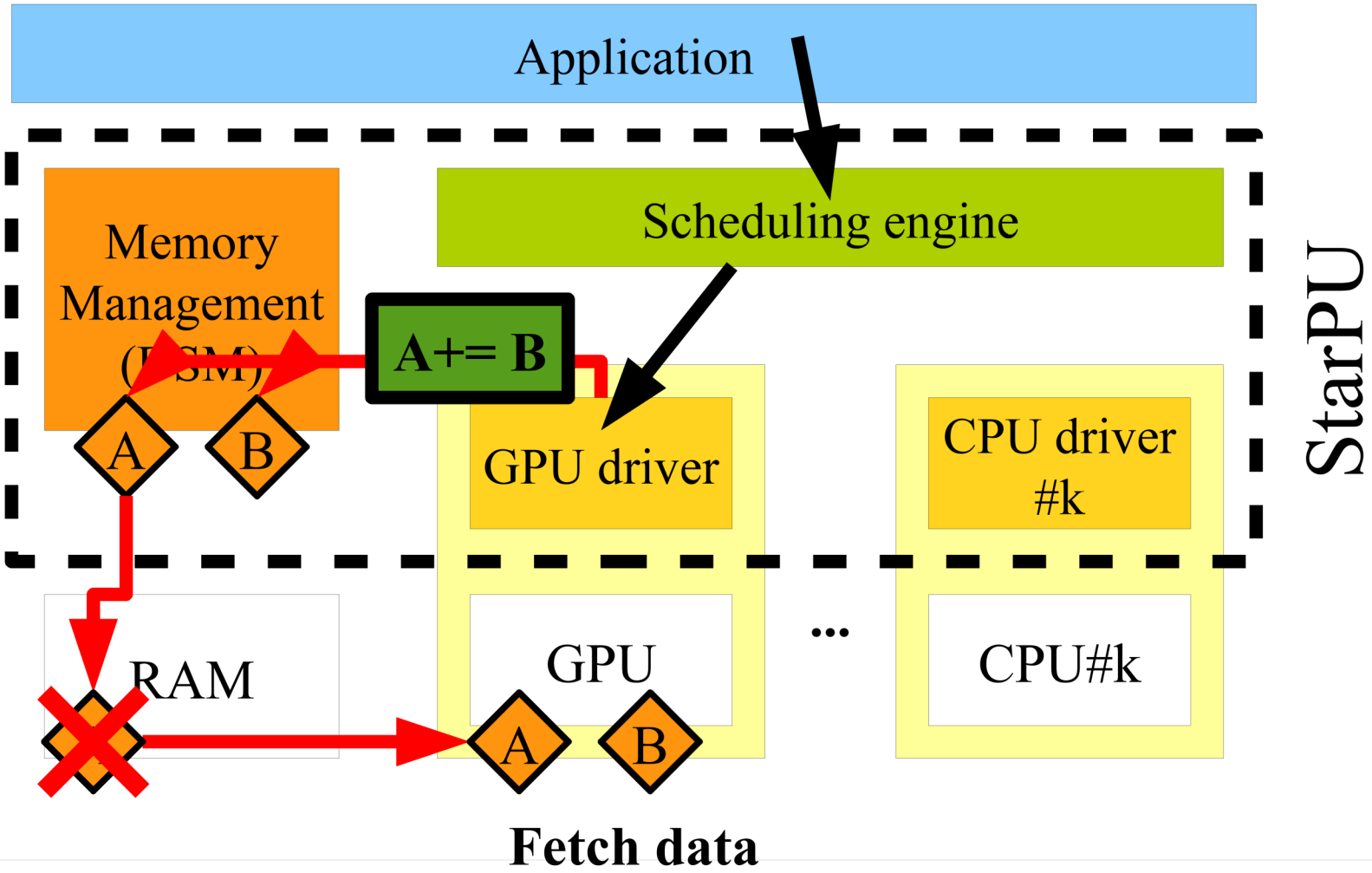
The StarPU runtime system

Execution model



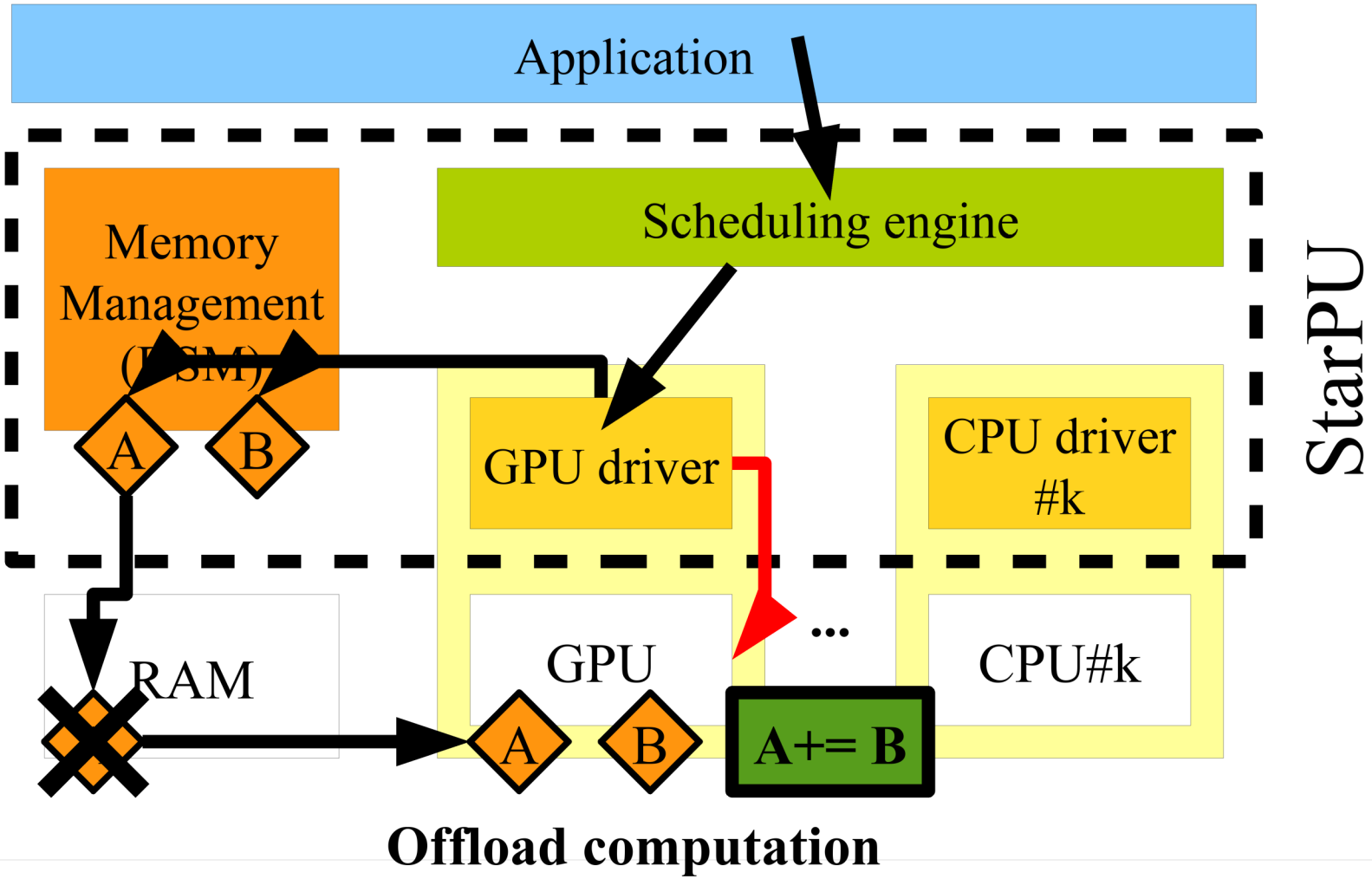
The StarPU runtime system

Execution model



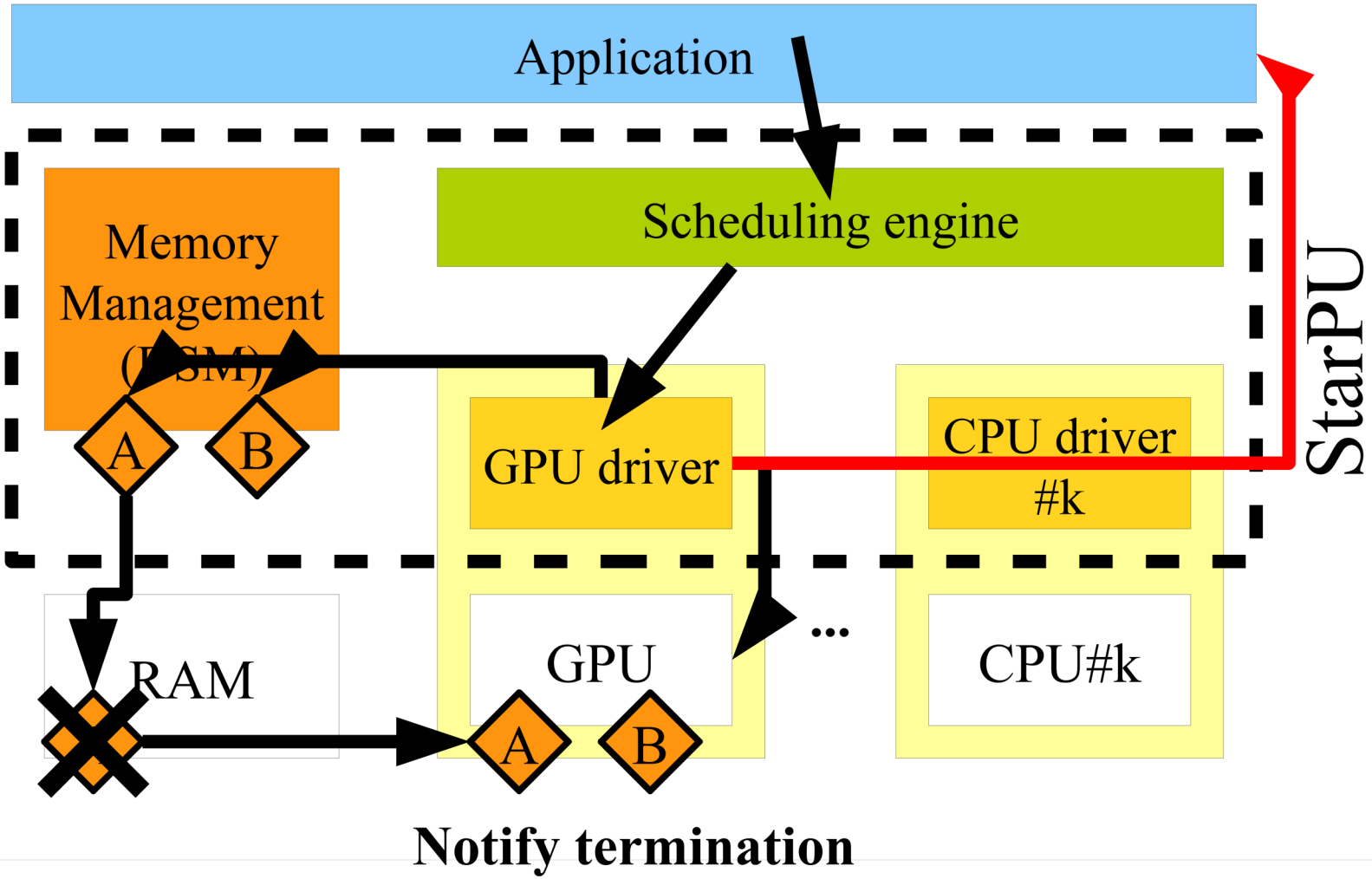
The StarPU runtime system

Execution model



The StarPU runtime system

Execution model



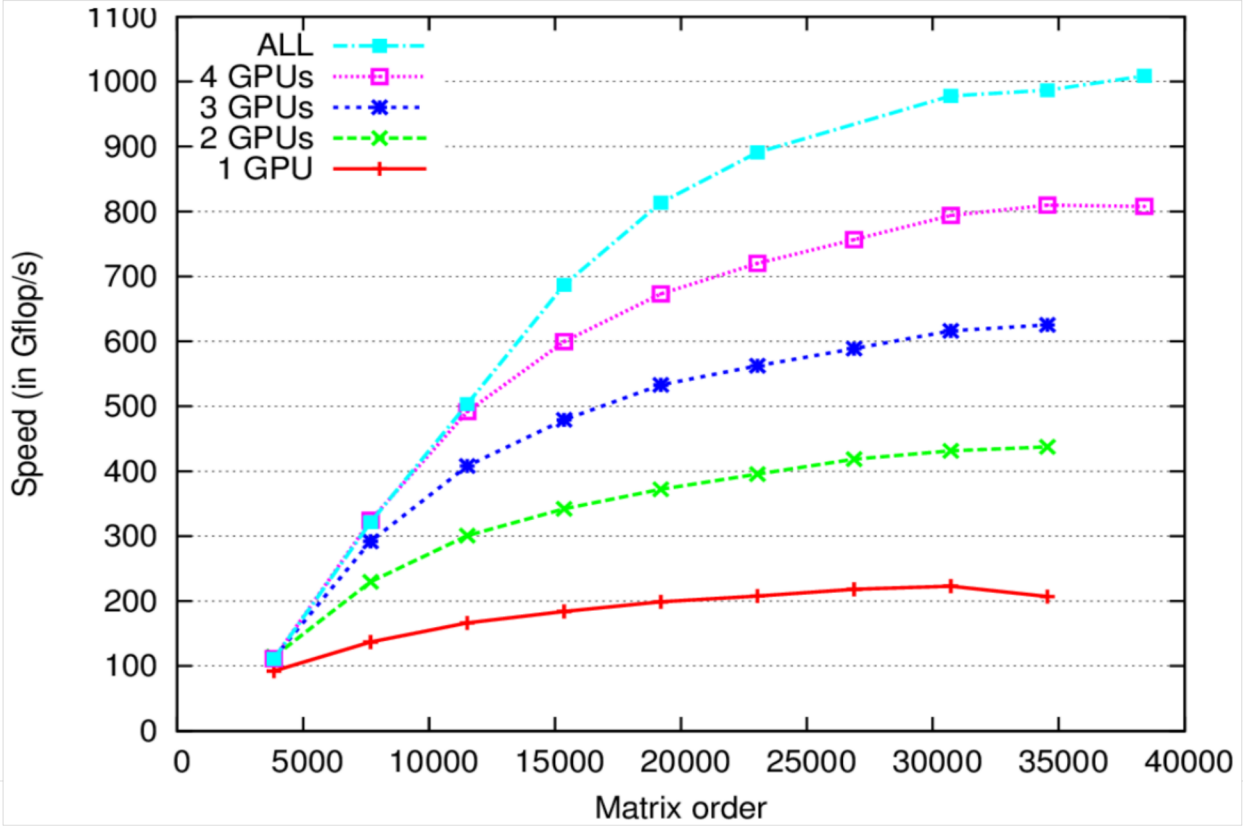
The StarPU runtime system

Supported platforms

- Supported architectures
  - Multicore CPUs (x86, PPC, ...)
  - NVIDIA GPUs
  - OpenCL devices (eg. AMD cards)
  - Intel Xeon Phi (MIC), Intel SCC (decommissioned)
  - Kalray MPPA (experimental)
  - Cell processors (experimental) [SAMOS'09]
- Supported Operating Systems
  - Linux
  - Mac OS
  - Windows

Performance teaser

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



Programming interface

Building details

- Makefile flags
  - CFLAGS +=$(shell pkg-config --cflags starpu-1.3)
  - LDLIBS +=$(shell pkg-config --libs starpu-1.3)
- Headers
  - #include <starpu.h>
- (De)Initialize StarPU
  - starpu_init(NULL);
  - ...
  - starpu_shutdown();

Terminology

- Data handle
  - Designates data managed by StarPU
  - input/output for tasks
- Codelet
  - Gathers different implementation variants achieving the same computation
  - Instantiated as tasks
- Task
  - Relates the two
  - Instantiation of a codelet to be run over data handles
- Worker
  - Executes tasks

Scaling a vector

Defining a codelet (4)

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

```
starpu_codelet scal_cl = {
    .cpu_funcs = { scal_cpu_func, scal_cpu_func_sse },
    .cuda_funcs = { scal_cuda_func },
    .opencl_funcs = { scal_opencl_func },
    .nbuffers = 1,
    .modes = { STARPU_RW },
};
```

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]));
```

- Submit tasks….
- 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   = STARPU_VECTOR_GET_PTR(vector);

    float *factor = cl_arg;

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

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;

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
        );
```

Scaling a vector

Defining a task, OpenMP support from K'Star

- Define a task that scales the vector by a constant

```
float factor = 3.14;

#pragma omp task depend(inout:vector)
scal(vector, factor);
```

Summary

```
starpu_codelet_t cl = { .cpu_func = my_f, ... };
float array[NX];
        ...
starpu_data_handle vector_handle;
starpu_vector_data_register(&vector_handle, 0,
            array, NX, sizeof(vector[0]));
...
starpu_task_insert(&cl, STARPU_RW, vector_handle, 0);
...
starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);
```

Scaling a vector

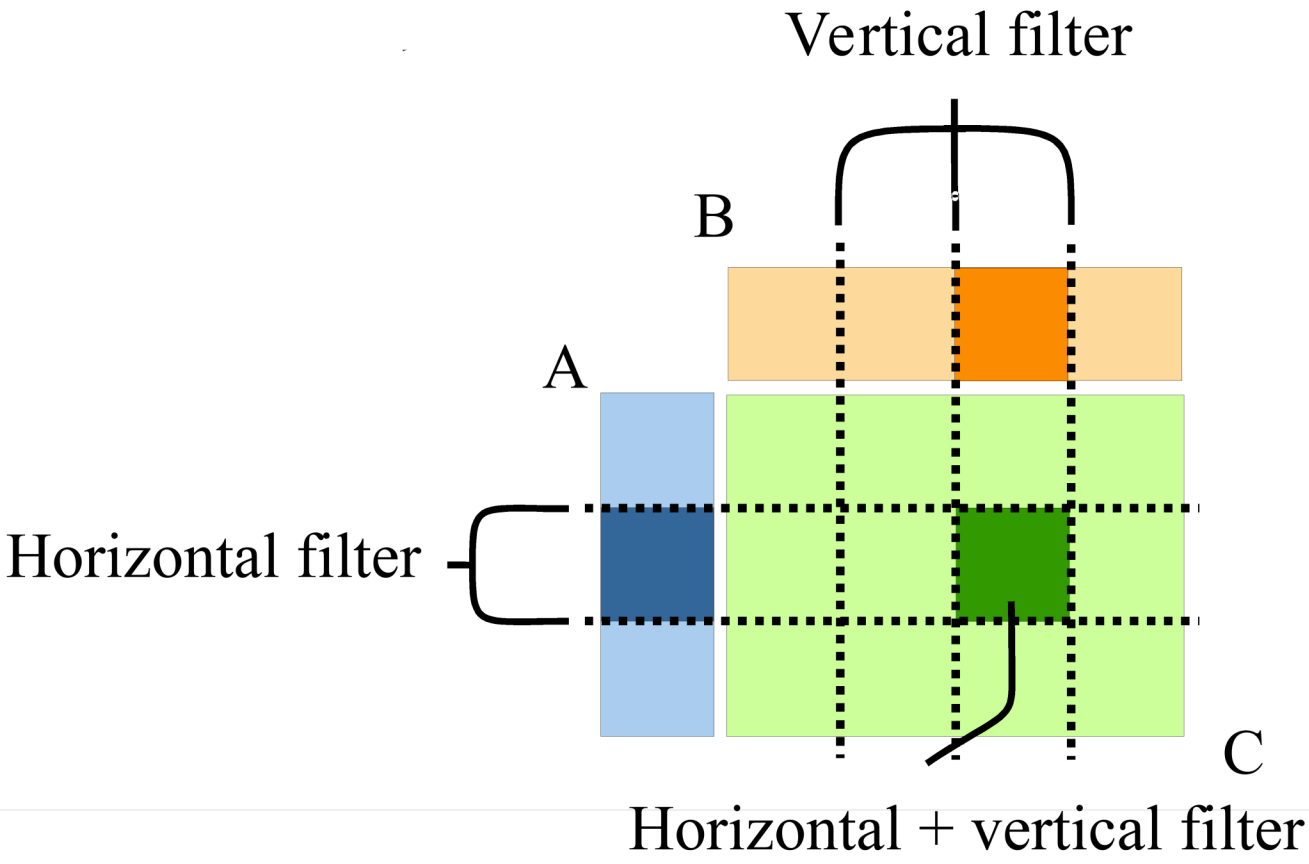
Defining a codelet (2)

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

```
__global__ void vector_mult_cuda(float *val, unsigned n, float factor)
{
        unsigned i = blockIdx.x * blockDim.x + threadIdx.x ;
        if (i < n) 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;

        unsigned per_block = 64 ;
        unsigned nblocks = (n + per_block-1) / per_block;
        vector_mult_cuda
           <<<nblocks,per_block, 0, starpu_cuda_get_local_stream()>>>
           (val, n, *factor);
}
```

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 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);

Interaction with StarPU execution

- Can wait for a given task
  - starpu_task_wait(task);
- Can access to the result within computation

```
starpu_data_acquire(vector_handle, STARPU_R);
printf("%d", array[0]);
starpu_data_release(vector_handle);
```

- Or as a callback

```
while (!converged) {
    starpu_task_insert(&cl, …);
    starpu_data_acquire_cb(vector_handle, STARPU_R,
        test_converged, NULL);
}
```

- And many more

Data support

- Various types
  - Predefined: Vectors, matrices, BCSR, CSC
  - Can be completely user-defined: e.g. compressed matrix, h-matrix
- Dynamic partitioning
  - Split matrix, vector, or completely user-defined
  - Can be synchronous: starpu_data_partition()
  - Or asynchronous:

```
starpu_data_partition_plan(handle, &sub_handles);

starpu_task_insert(…, handle, …);
starpu_data_partition_submit(handle, &sub_handles);
starpu_task_insert(…, sub_handles[i], ...);
starpu_data_unpartition_submit(handle, &sub_handles);
starpu_task_insert(…, handle, …);
```

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)

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

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

Task-based programming

- Needs code restructuring
  - Split computation into tasks
    - BLAS, typically
    - Supposed to have “stable” performance
- Constraining
  - No global variables
    - Mandatory for GPUs
- Actually… functional programming

So a good move, in the end ☺

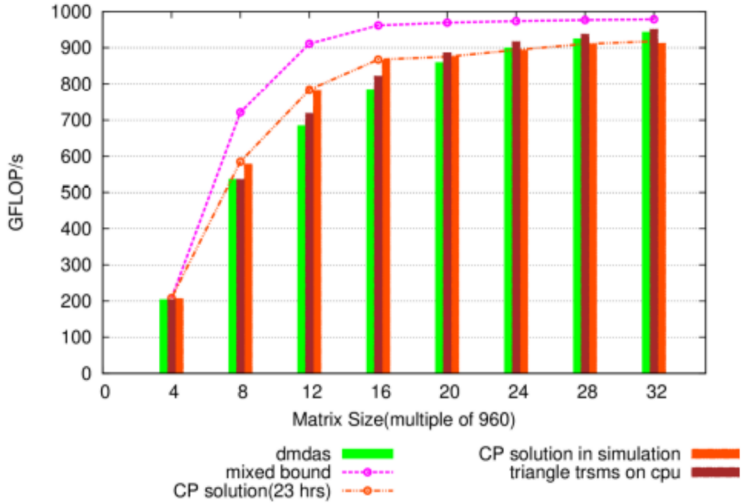
- Have to accept constraints and losing control

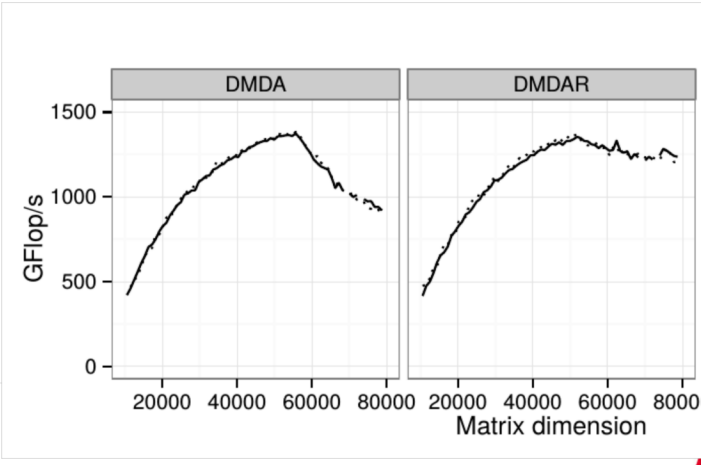
Just like we did when moving from assembly to high-level languages

Task-based support

Then all of this comes “for free” :

- Task/data scheduling
  - Pipelining
  - Load balancing
  - GPU memory limitation management
  - Data prefetching
- Performance bounds
- Distributed execution through MPI
- High-level performance analysis
- Out-of-core : optimized swapping to disk
- Debugging sequential execution
- Reproducible performance simulation





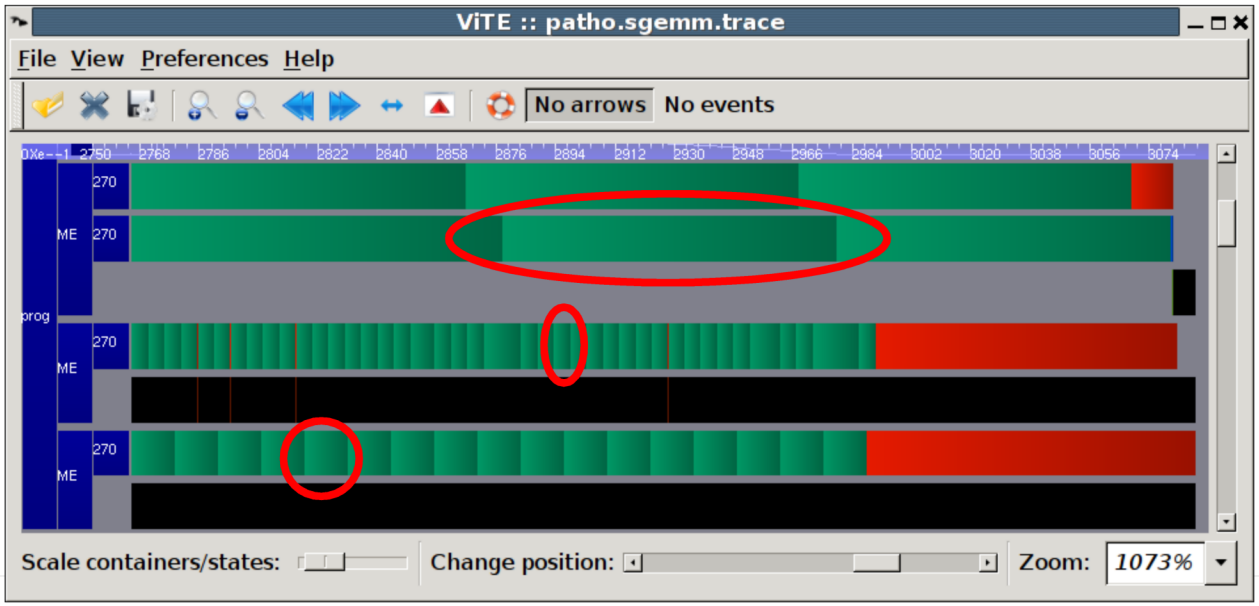
Task Scheduling

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

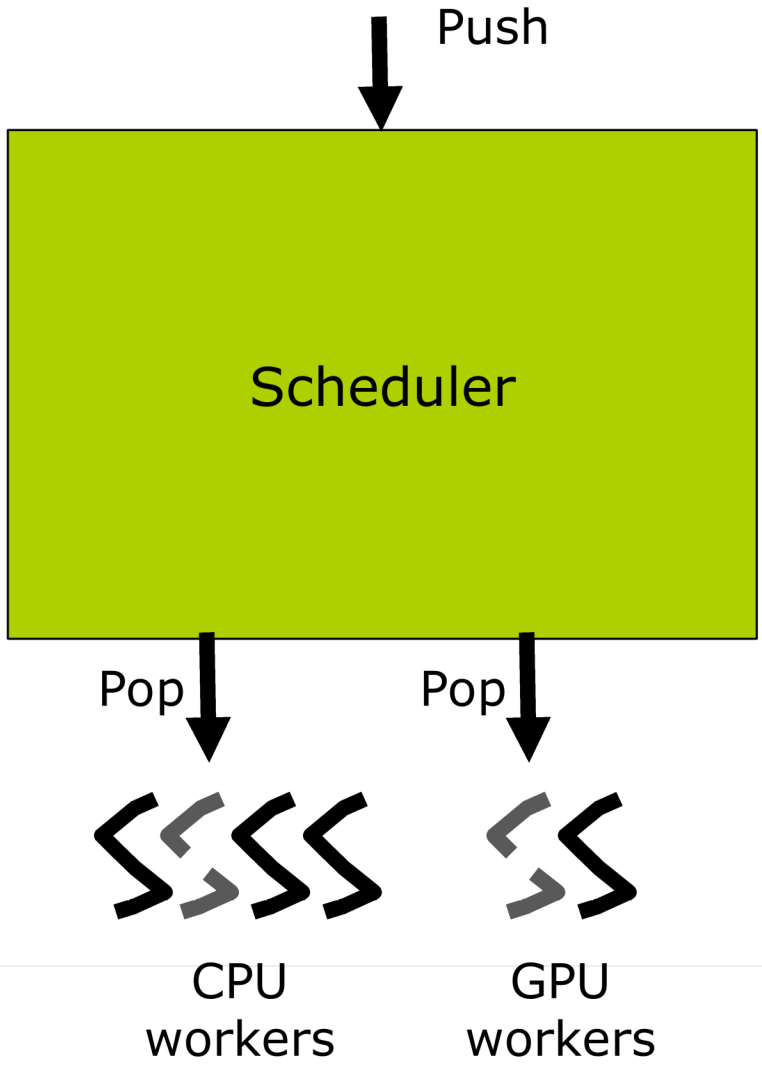
Task scheduling

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



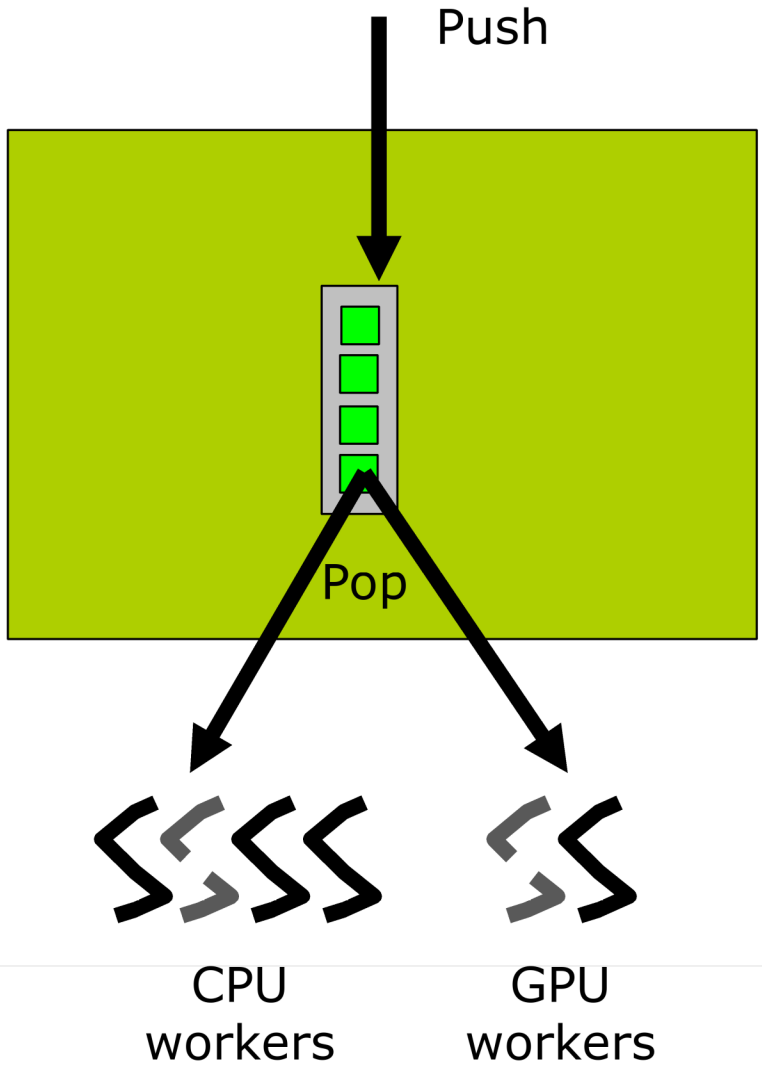
Task scheduling

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



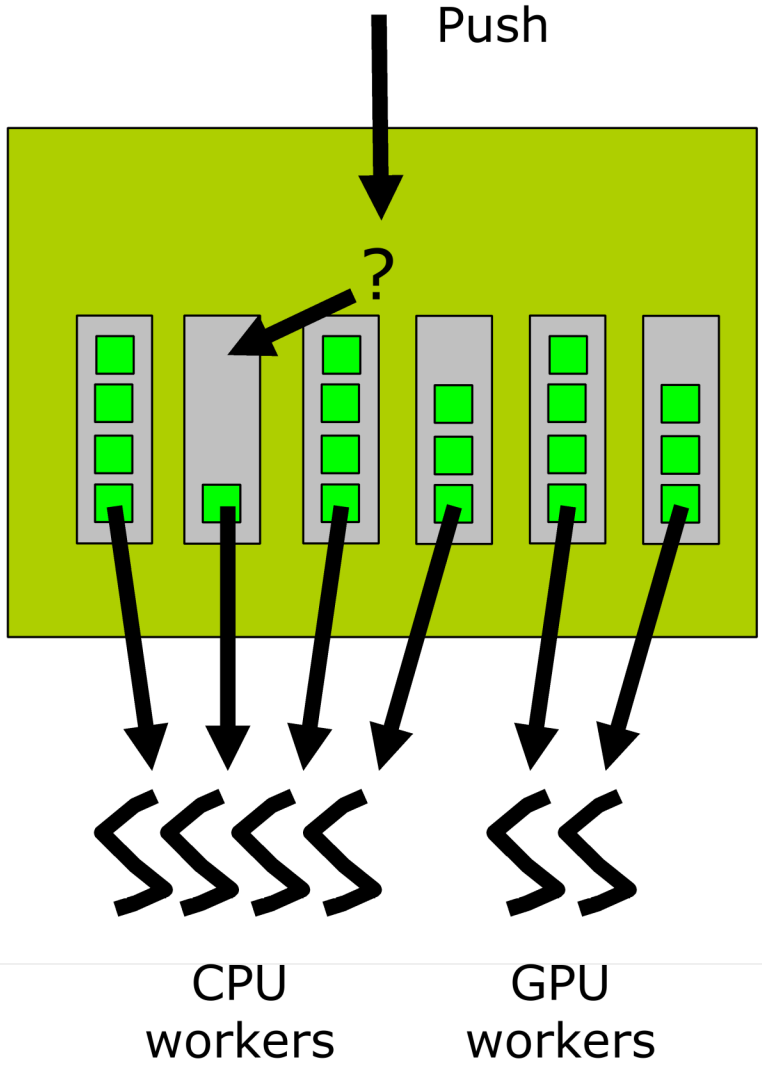
Task scheduling

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



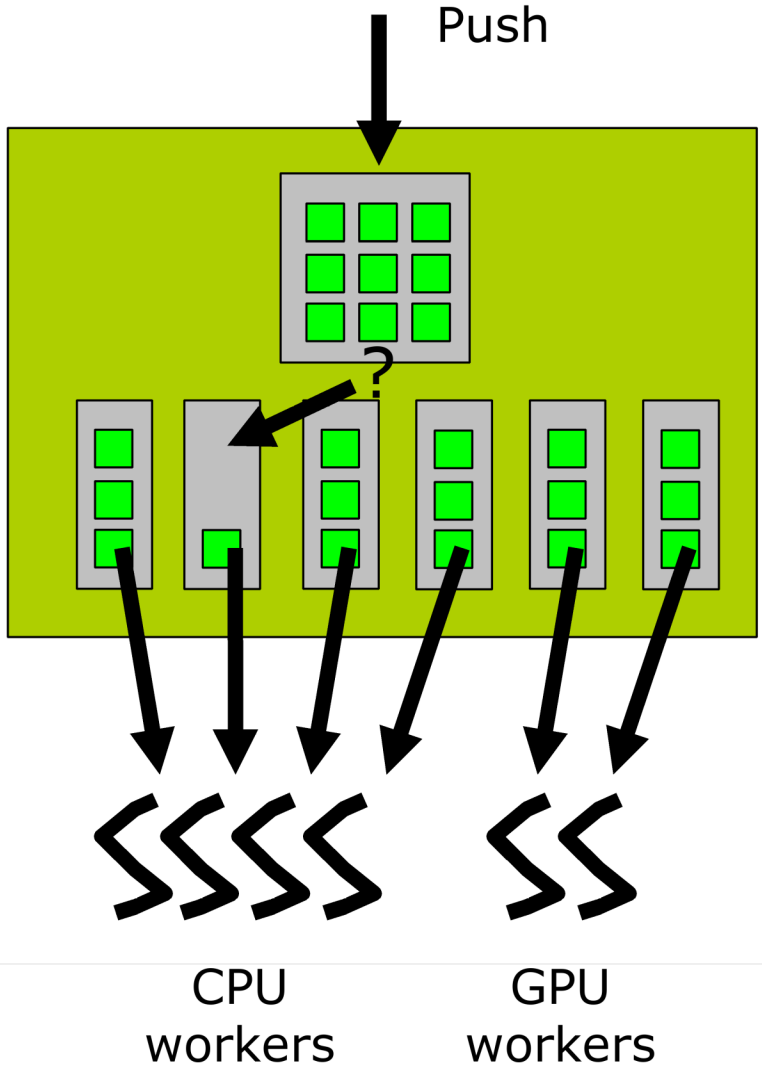
Task scheduling

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



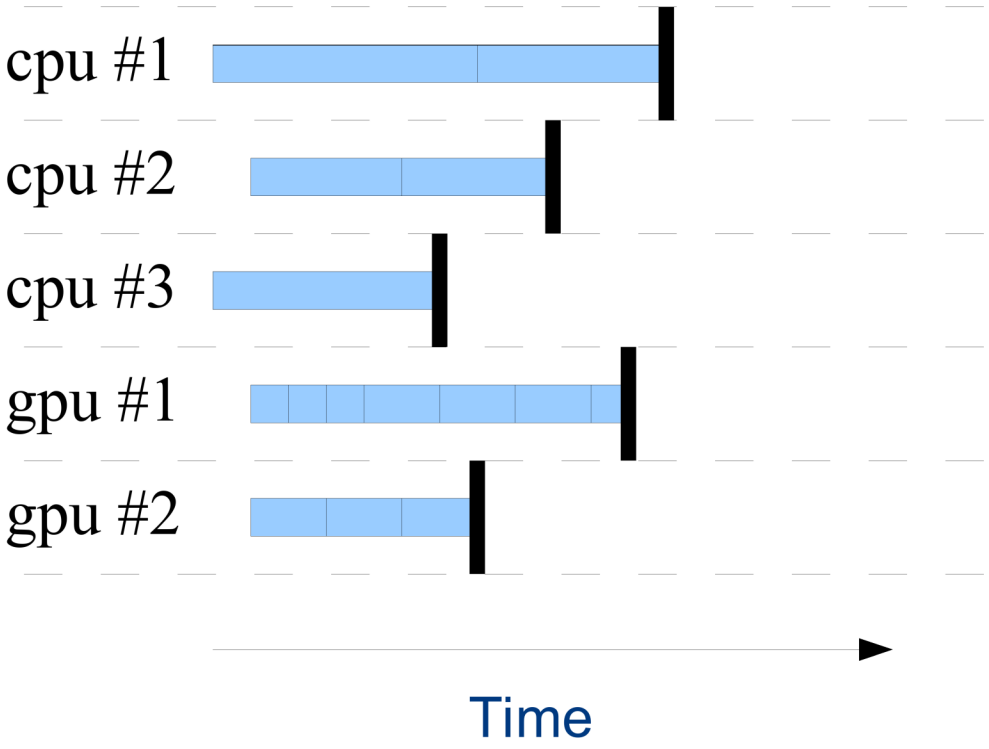
History-based performance model

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

Also STARPU_REGRESSION_BASED, STARPU_NL_REGRESSION_BASED, or explicit

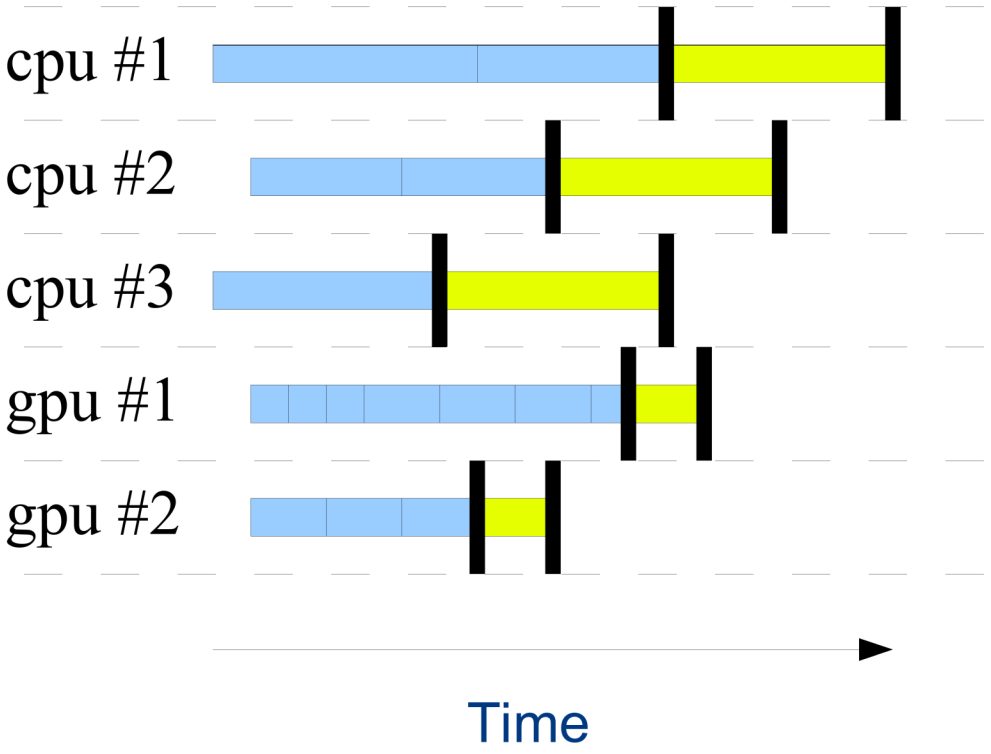
Prediction-based scheduling
Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time



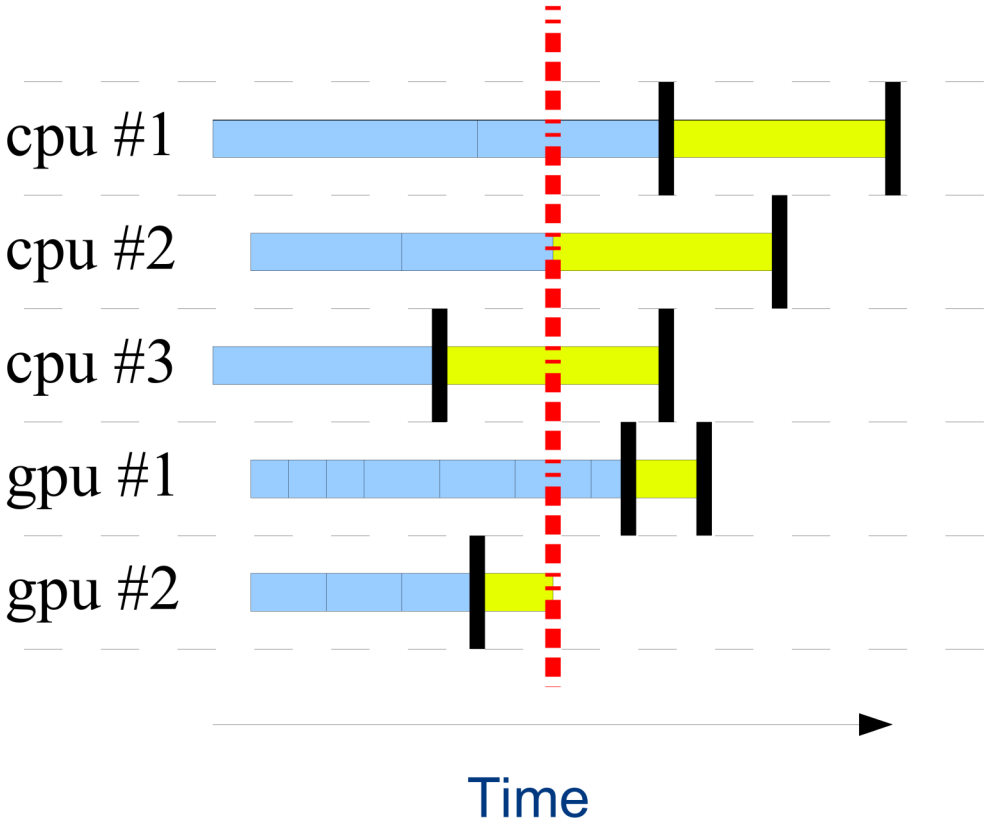
Prediction-based scheduling
Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time



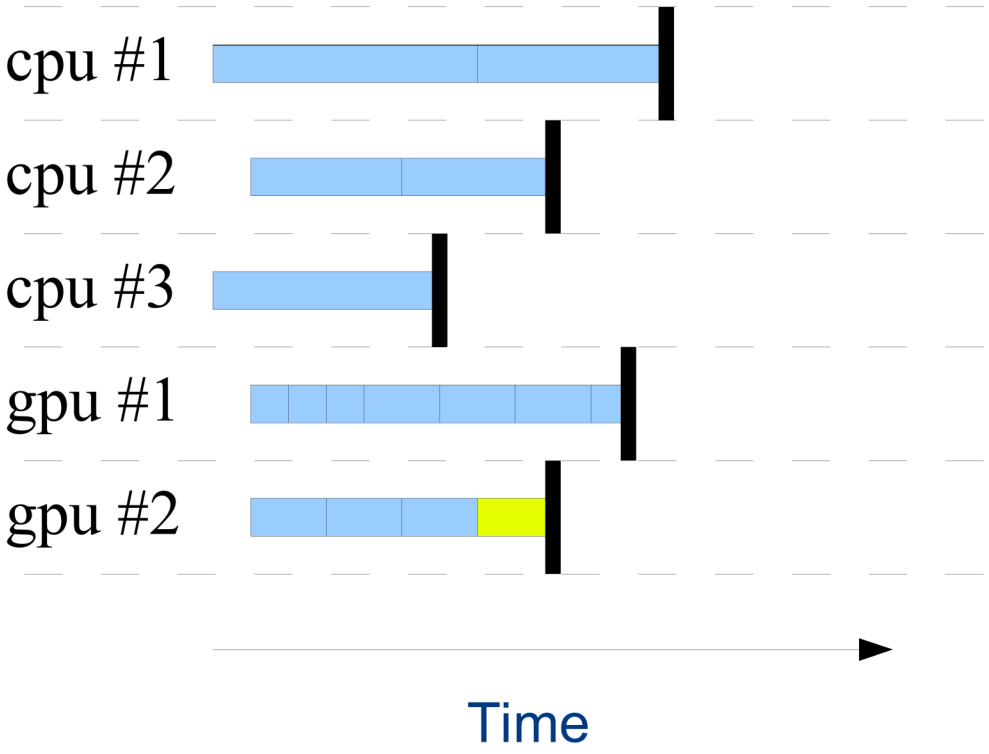
Prediction-based scheduling
Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time



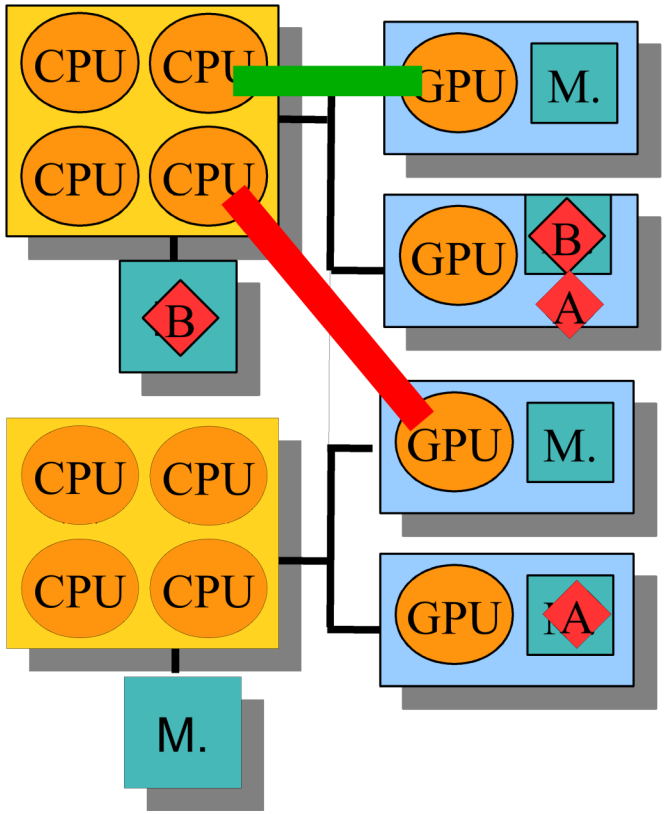
Prediction-based scheduling
Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time



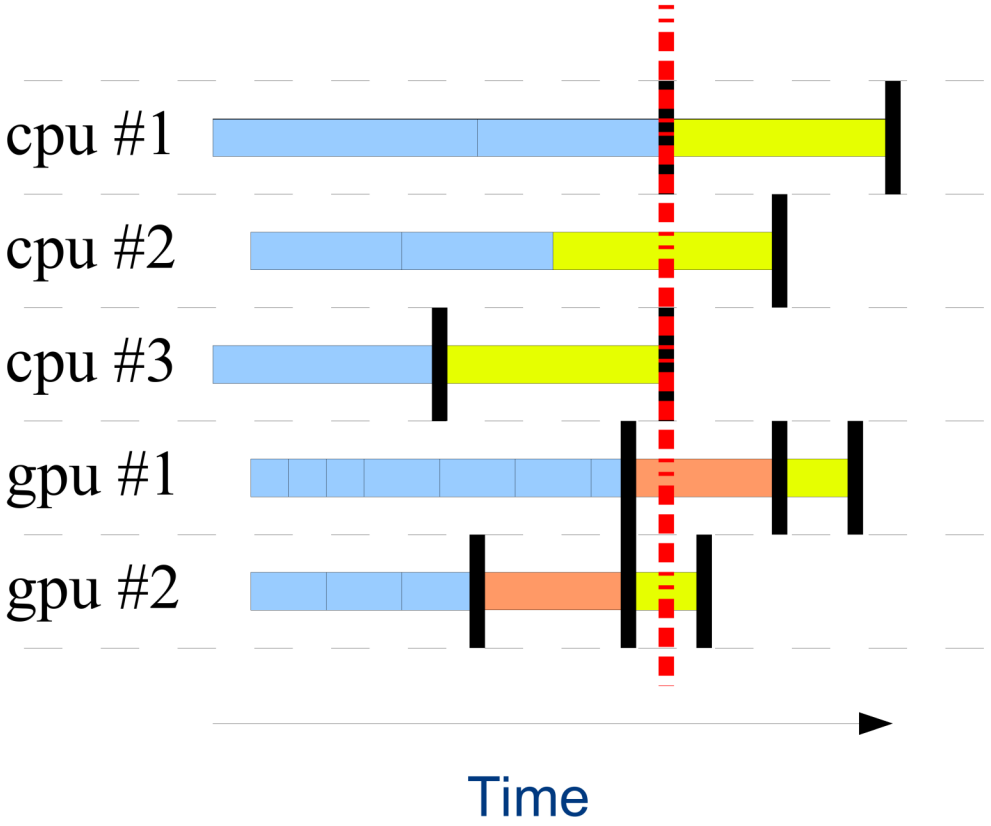
Predicting data transfer overhead
Motivations

- 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



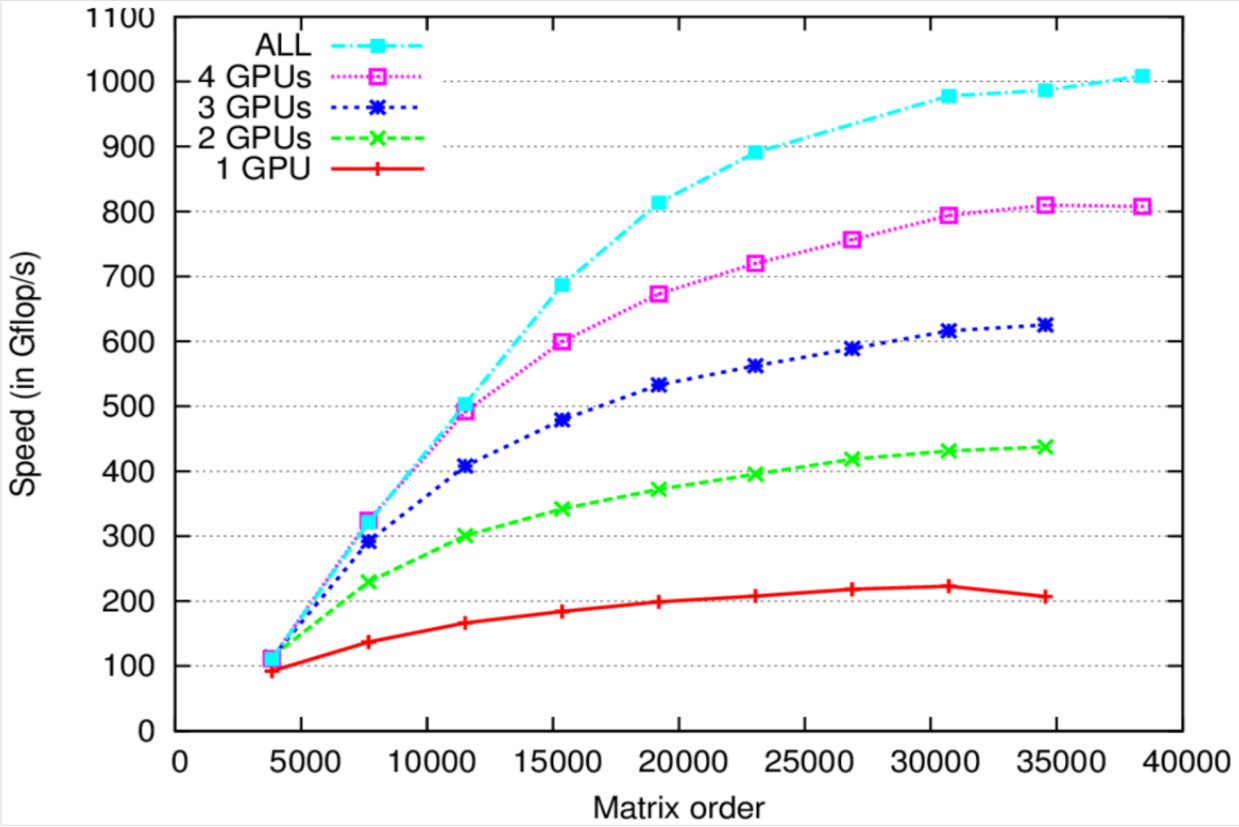
Prediction-based scheduling
Load balancing

- Data transfer time
  - Sampling based on off-line calibration
- Can be used to
  - Better estimate overall exec time
  - Minimize data movements
- Further
  - Power overhead
- [ICPADS'10]



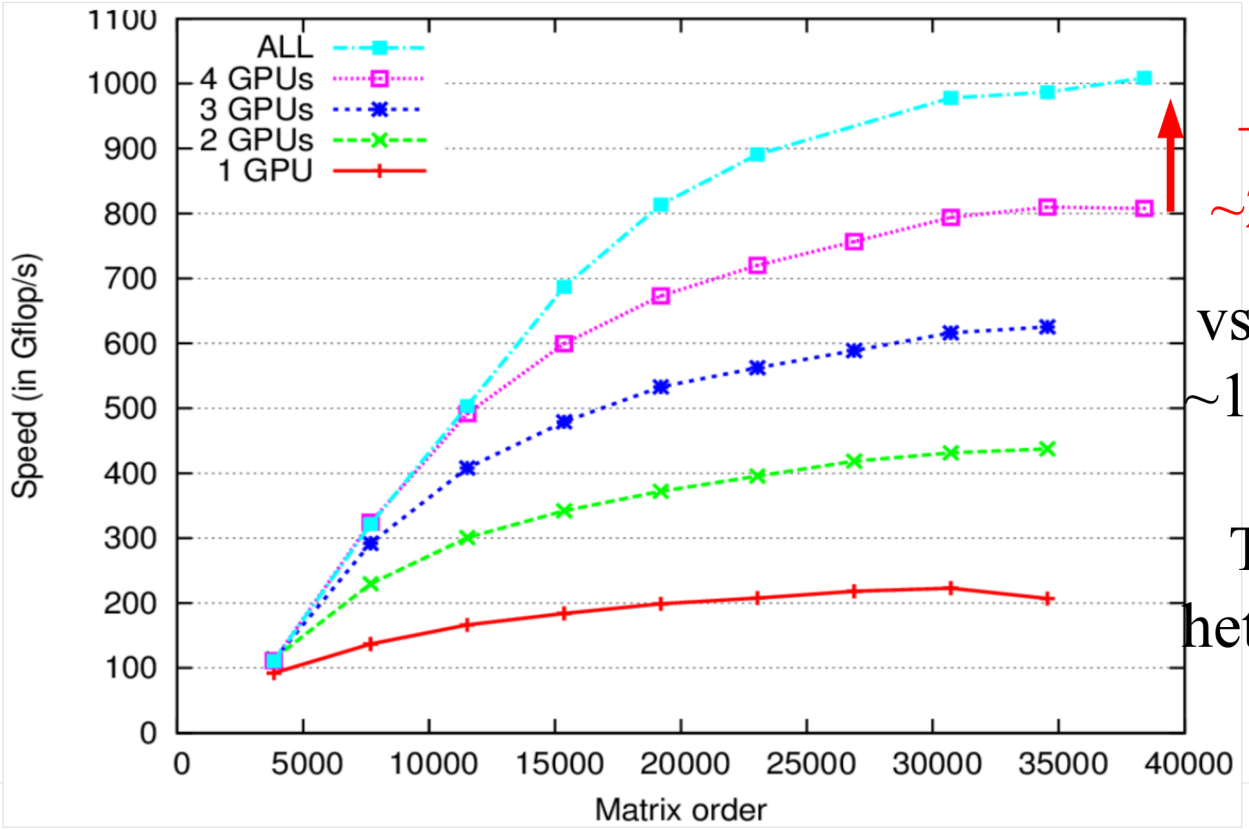
Mixing PLASMA and MAGMA with StarPU

- 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)



+12 CPUs
~200GFlops

vs measured
~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

(see StarPU handbook for details)

Bus performance

```
$ ./tools/starpu_machine_display
5 CPU cores
     CPU 0
     …
3 CUDA Devices
     CUDA 0 (Tesla C2050 3.0 GiB 02:00.0)
     …
from      to RAM     to CUDA 0   to CUDA 1   to CUDA 2
RAM       0.0        5236.89     5236.71     5240.12
CUDA 0    4547.68    0.0         3031.37     3093.99
CUDA 1    4547.62    3030.38     0.0         3093.90
CUDA 2    4537.36    3823.06     3823.17     0.0
```

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
```

Disk usage

```
$ STARPU_BUS_STATS=1 ./tests/disk/disk_copy
0 -> 1: 337 MB/s
1 -> 0: 337 MB/s
0 -> 1: 1593 µs
1 -> 0: 1593 µs
NUMA 0 -> Disk 0  0.0625 GB  88.6847 MB/s  (transfers: 2 - avg 32MB)
Total transfers: 0.0625 GB
```

Energy consumption

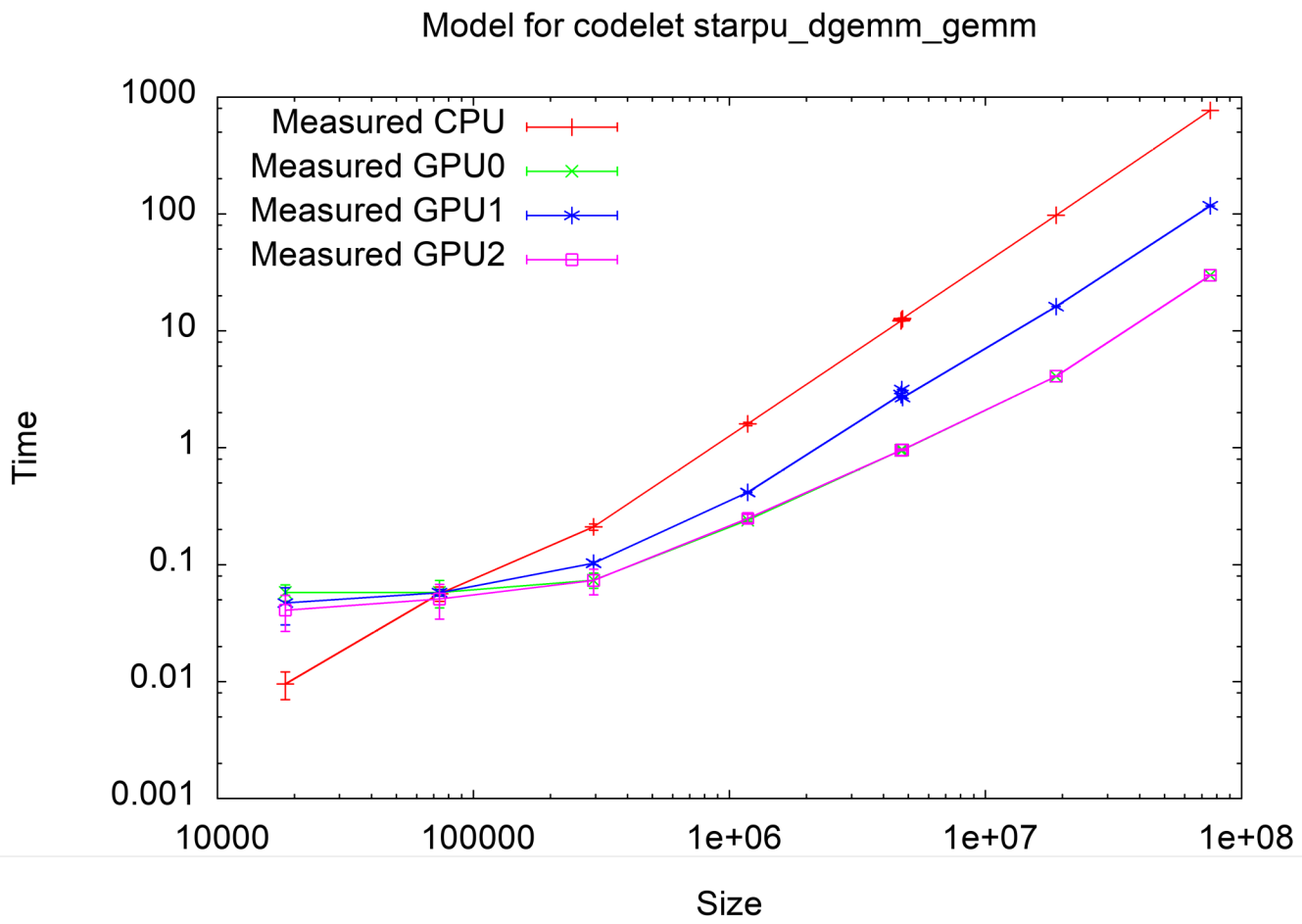
```
$ STARPU_WORKER_STATS=1 STARPU_PROFILING=1 ./examples/stencil/stencil
OpenCL 0 (Quadro FX 5800)
773 task(s)
total: 409.60 ms executing: 340.51 ms sleeping: 0.00
5040.000000 J consumed
OpenCL 1 (Quadro FX 5800)
767 task(s)
total: 409.62 ms executing: 346.28 ms sleeping: 0.00
10280.000000 J consumed
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
880805ba49152                1.233333e+02    1.063576e+01    1612
8bd4e11d2359296              1.331984e+04    6.971079e+02    635
performance model for cuda_0
# hash          size              mean              dev              n
880805ba49152                2.743658e+01    2.178427e+00    496
8bd4e11d2359296              6.207991e+02    6.941988e+00    307
```

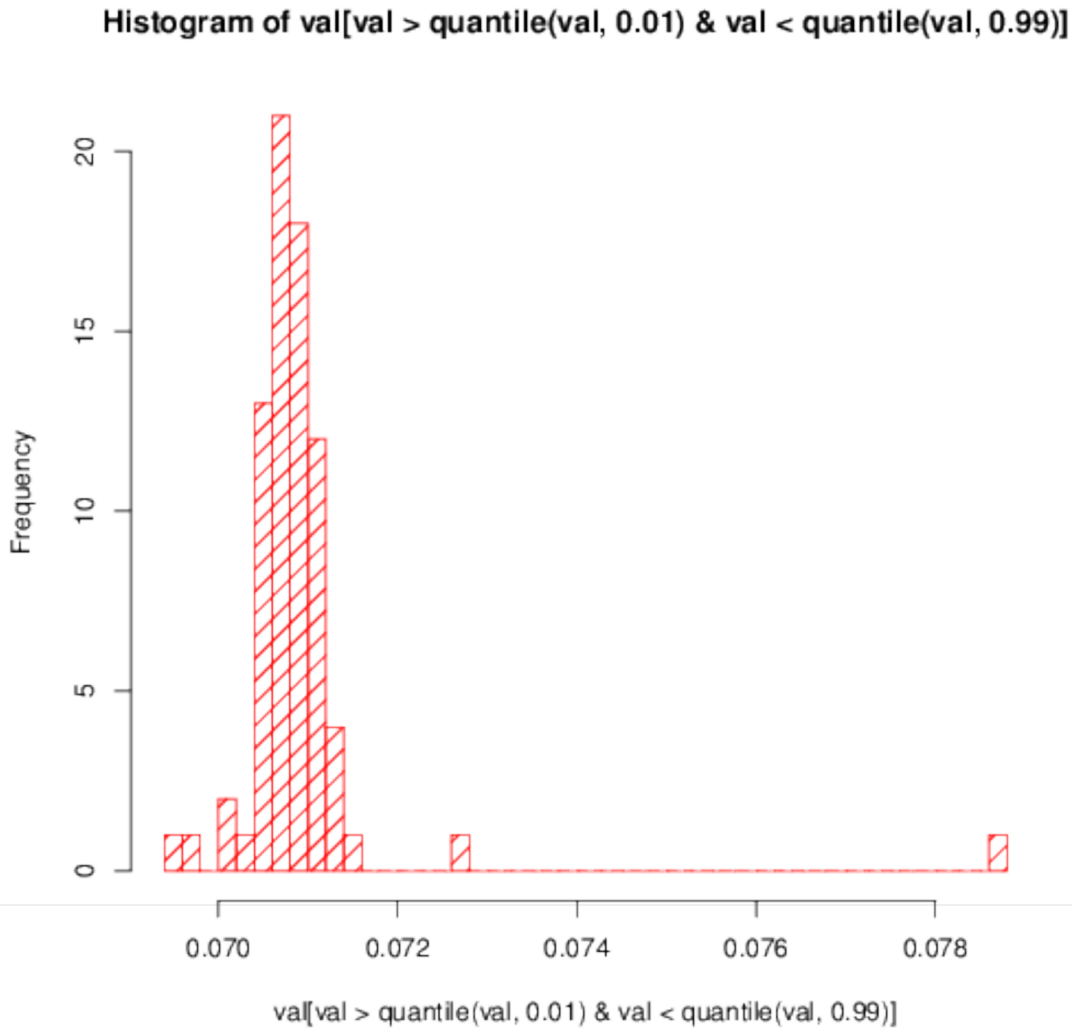
Performance models plot

```
$ starpu_perfmodel_plot -s starpu_dgemm_gemm

$ ./starpu_dgemm_gemm.gp
```



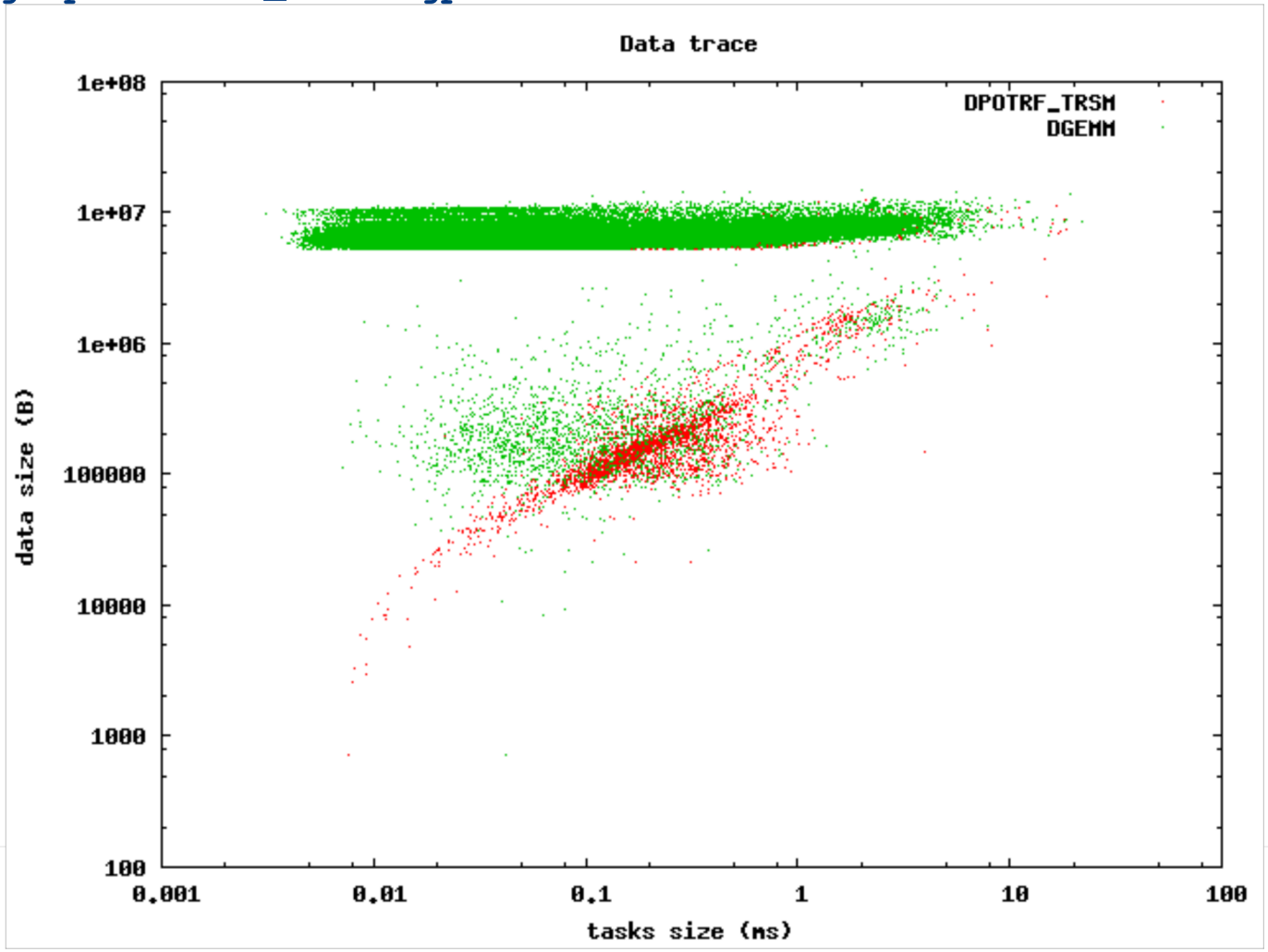
Kernel performance plot

```
$ starpu_fxt_tool -i /tmp/prof_file_user_sthibaul0

$ starpu_codelet_histo_profile distrib.data
```



Kernel performance plot

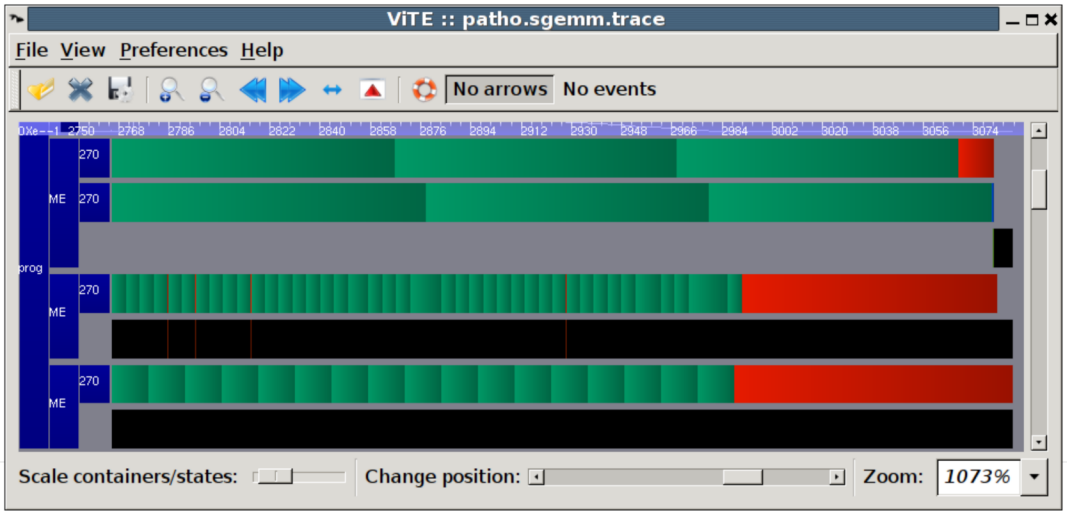
```
$ starpu_fxt_data_trace /tmp/prof_file_sthibaul_0

$ gnuplot data_trace.gp
```



Offline performance analysis

Visualize execution traces

- Generate a Pajé trace
  - https://savannah.nongnu.org/projects/fkt
  - ./configure --with-fxt
  - fxt_tool -i /tmp/prof_file_user_yourlogin
    → paje.trace
- Vite trace visualization tool
  - Freely available from http://vite.gforge.inria.fr/ (open source !)
  - vite paje.trace



2 Xeon cores

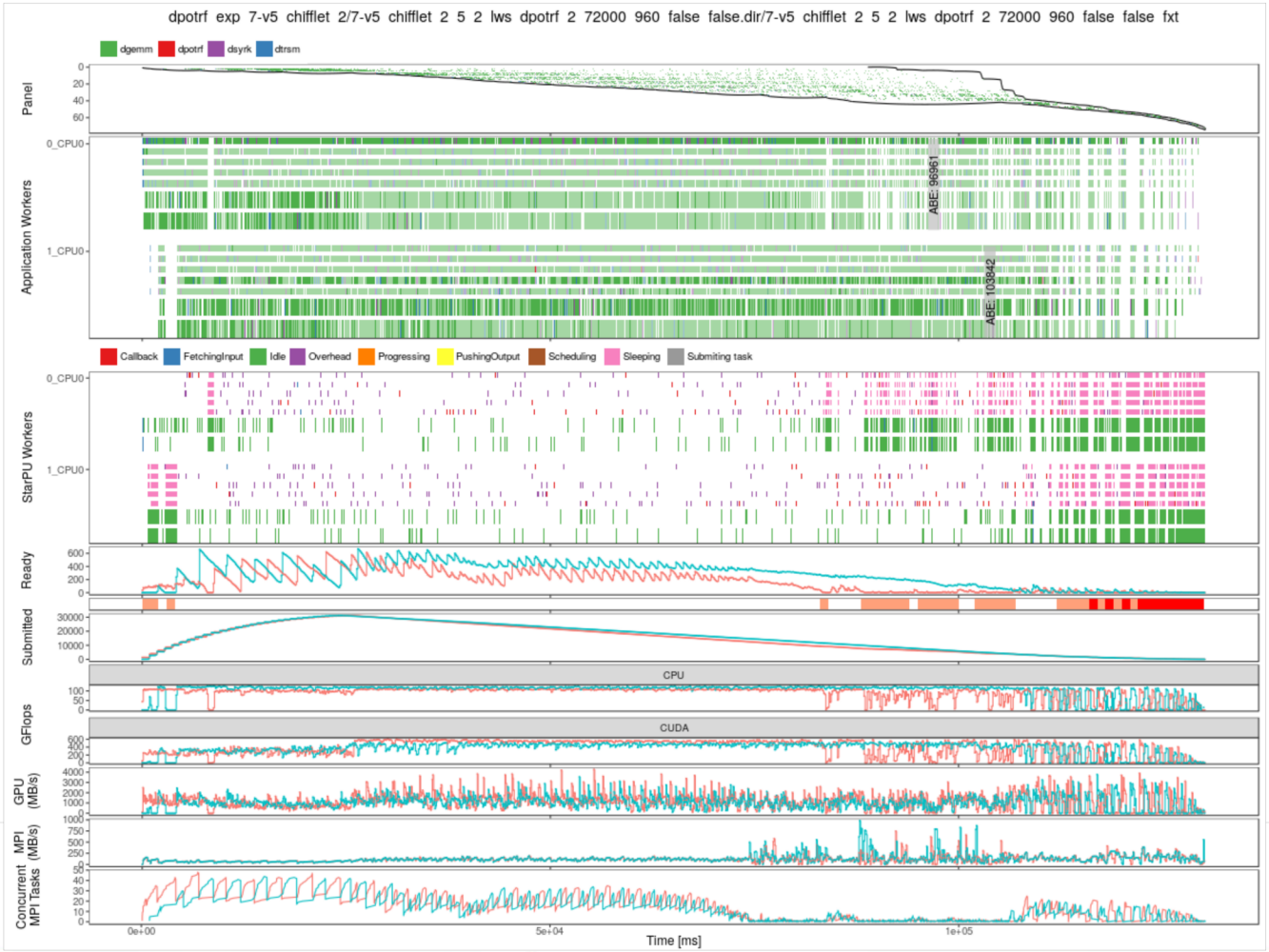
Quadro FX5800

Quadro FX4600

Offline performance analysis

Visualize execution traces

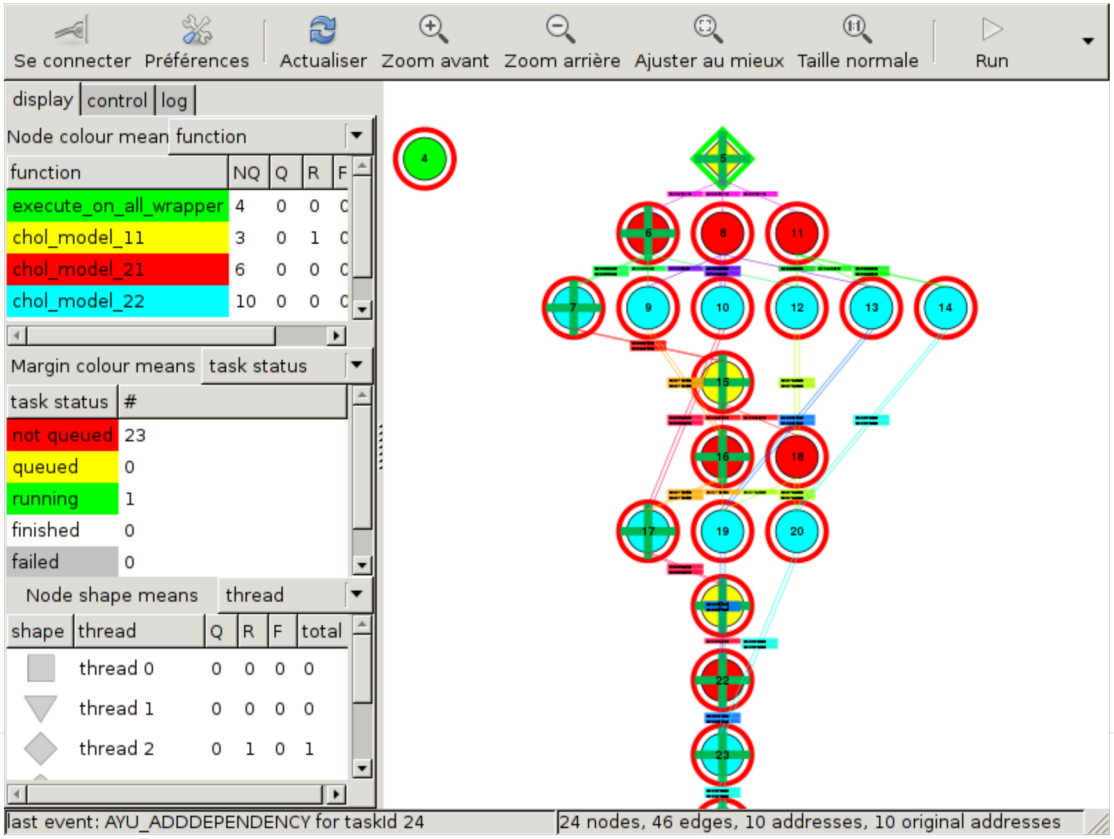
- Cluster traces too (On-going work)



Temanejo: task debugger

A debugger at the task level

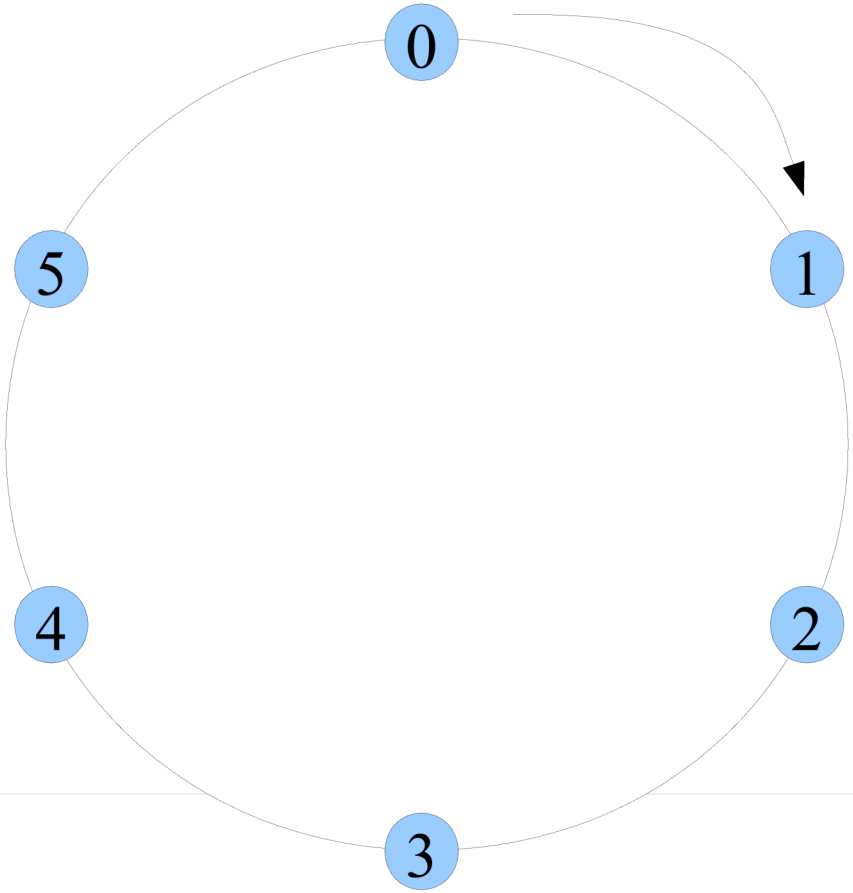
- Visualize task graph
- Add Breakpoints
- Execute task-by-task
- ...



Cluster support

MPI ring example

- Token passed and incremented from node to node



MPI ring 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 ring example

```
for (loop = 0 ; loop < NLOOPS; loop++) {
        if ( !(loop == 0 && rank == 0)) {
                starpu_data_acquire(data_handle, STARPU_W) ;
                MPI_Recv(&data, prev_rank, …) ;
                starpu_data_release(data_handle) ;
        }
        starpu_task_insert(&increment_codelet, STARPU_RW, data_handle, 0);
        starpu_task_wait_for_all();
        if ( !(loop == NLOOPS-1 && rank == size-1)) {
                starpu_data_acquire(data_handle, STARPU_R) ;
                MPI_Send(&data, next_rank, …) ;
                starpu_data_release(data_handle) ;
        }
}
```

StarPU-MPI ring example

```
for (loop = 0 ; loop < NLOOPS; loop++) {
        if ( !(loop == 0 && rank == 0))

                starpu_mpi_irecv_submit(data_handle, prev_rank, …) ;


        starpu_task_insert(&increment_codelet, STARPU_RW, data_handle, 0);

        if ( !(loop == NLOOPS-1 && rank == size-1))

                starpu_mpi_isend_submit(data_handle, next_rank, …) ;


}
starpu_task_wait_for_all() ;
```

How to scale over MPI?

(StarPU handles intra-MPInode scheduling fine)

- Splitting graph by hand
  - Complex, not flexible
- Master-Slave does not scale
  - ➔ Each node should determine its duty by itself
- Algebraic representation of e.g. Parsec
  - Difficult to write
  - Not flexible enough for any kind of application
- Recursive task graph unrolling
  - Complex

➔ Rather just unroll the whole task graph on each node

StarPU-MPI ring example

```
for (loop = 0 ; loop < N * NLOOPS; loop++) {

        starpu_mpi_task_insert(&increment_codelet, STARPU_RW, data_handle,
                               STARPU_ON_NODE, loop % N, 0);

}
starpu_task_wait_for_all() ;
```

Automatic generation of Send/Recv MPI VSM

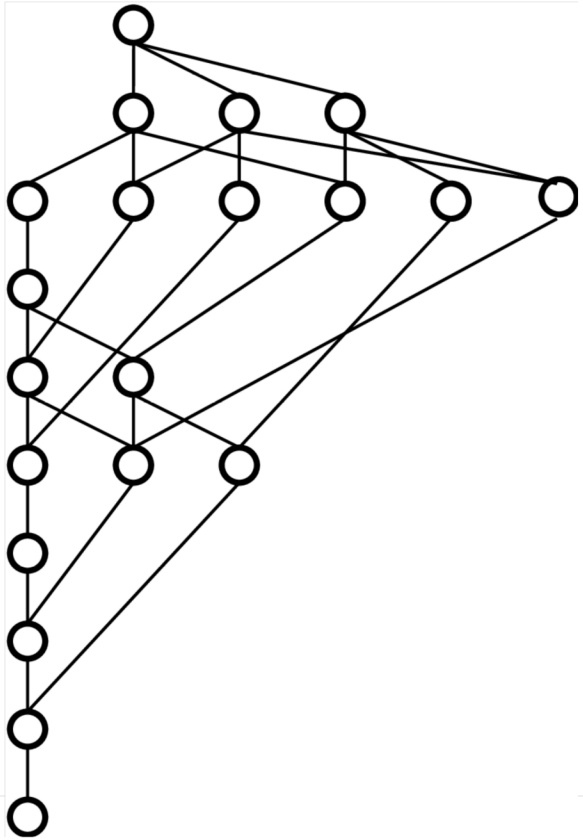
- Application decides data distribution over MPI nodes
- But data coherency extended to the MPI level
  - Automatic starpu_mpi_send/recv calls for each task
- Similar to a DSM, but granularity is whole data and whole task

- All nodes process the whole algorithm
  - Actual task execution according to data being written to

Sequential-looking code !

MPI VSM

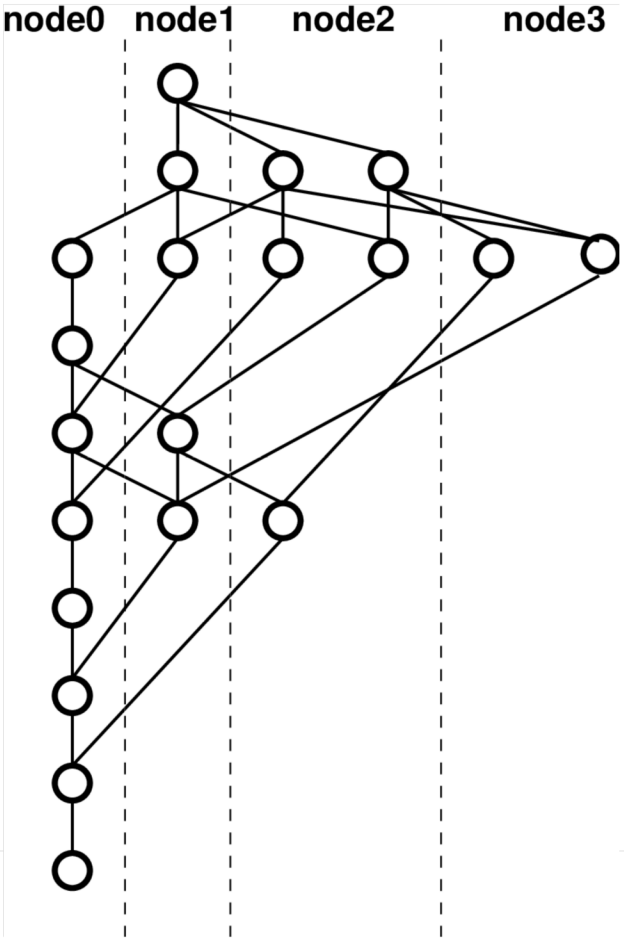
```
For (k = 0 .. tiles – 1) {
        POTRF(A[k,k])
        for (m = k+1 .. tiles – 1)
                TRSM(A[k,k], A[m,k])
        for (m = k+1 .. tiles – 1) {
                SYRK(A[m,k], A[m,m])
                for (n = m+1 .. tiles – 1)
                        GEMM(A[m,k], A[n,k], A[n,m])
        }
}
```



MPI VSM

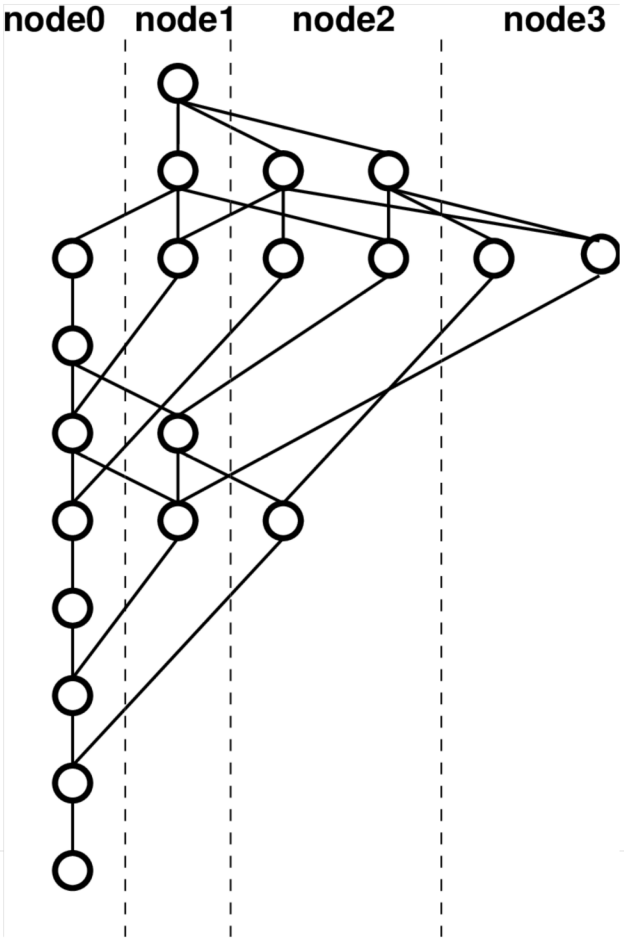
- Data mapping (e.g. 2D block-cyclic)

```
int get_rank(int m, int n) { return ((m%p)*q + n%q); }

For (m = 0 .. tiles – 1)
        For (n = m .. tiles – 1)
                set_rank(A[m,n], get_rank(m,n));

For (k = 0 .. tiles – 1) {
        POTRF(A[k,k])
        for (m = k+1 .. tiles – 1)
                TRSM(A[k,k], A[m,k])
        for (m = k+1 .. tiles – 1) {
                SYRK(A[m,k], A[m,m])
                for (n = m+1 .. tiles – 1)
                        GEMM(A[m,k], A[n,k], A[n,m])
        }
}
```



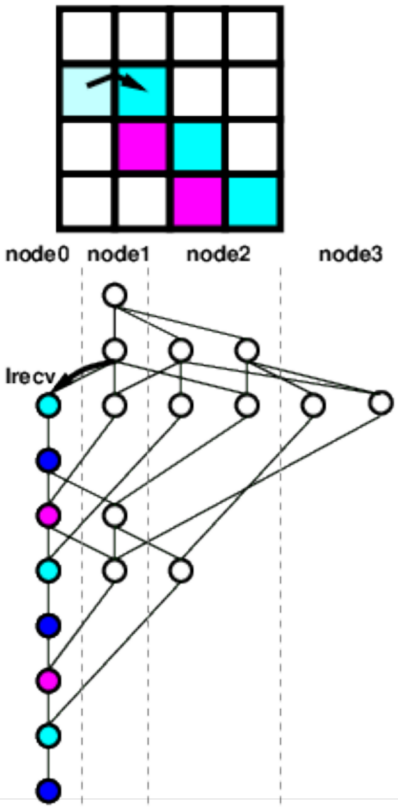
MPI VSM

- Each node unrolls the whole task graph
- Data ↔ node mapping
  - Provided by the application
    - E.g. 2D block-cyclic
  - Can be modified during submission
    starpu_mpi_data_migrate()
- Task ↔ node mapping
  - Tasks move to data they modify
- Separation of concerns: graph vs mapping
- MPI transfers
  - Automatically queued
- Local view of the computation
  - No synchronizations
  - No global scheduling

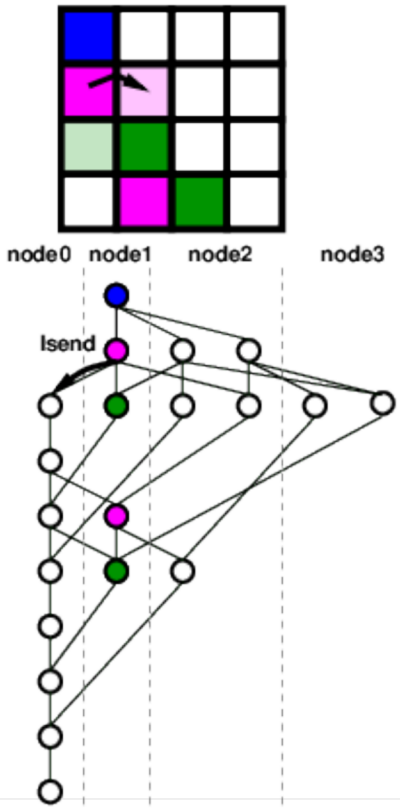


MPI VSM

- Right-Looking Cholesky decomposition (from PLASMA)



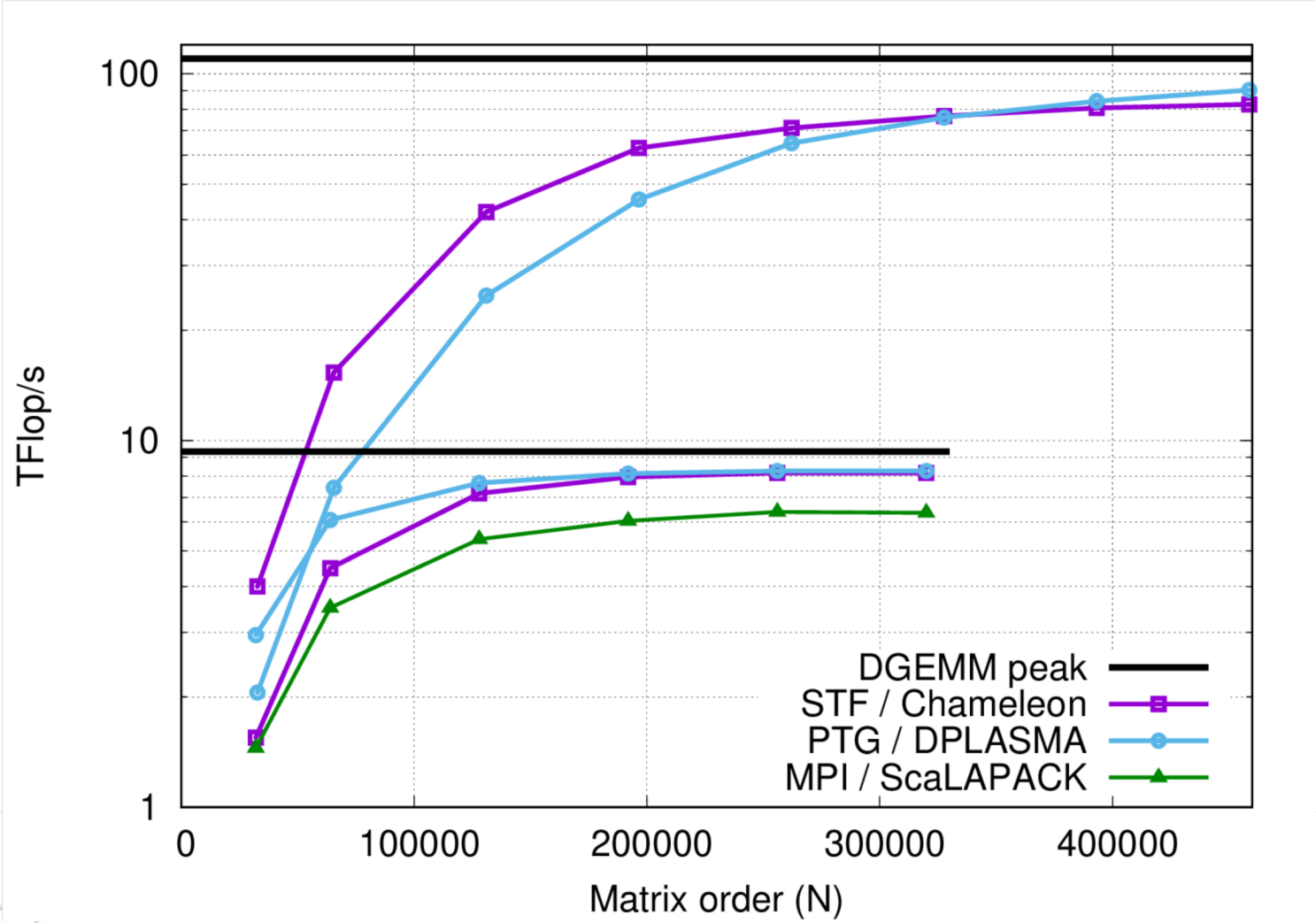
Node 0 execution



Node 1 execution

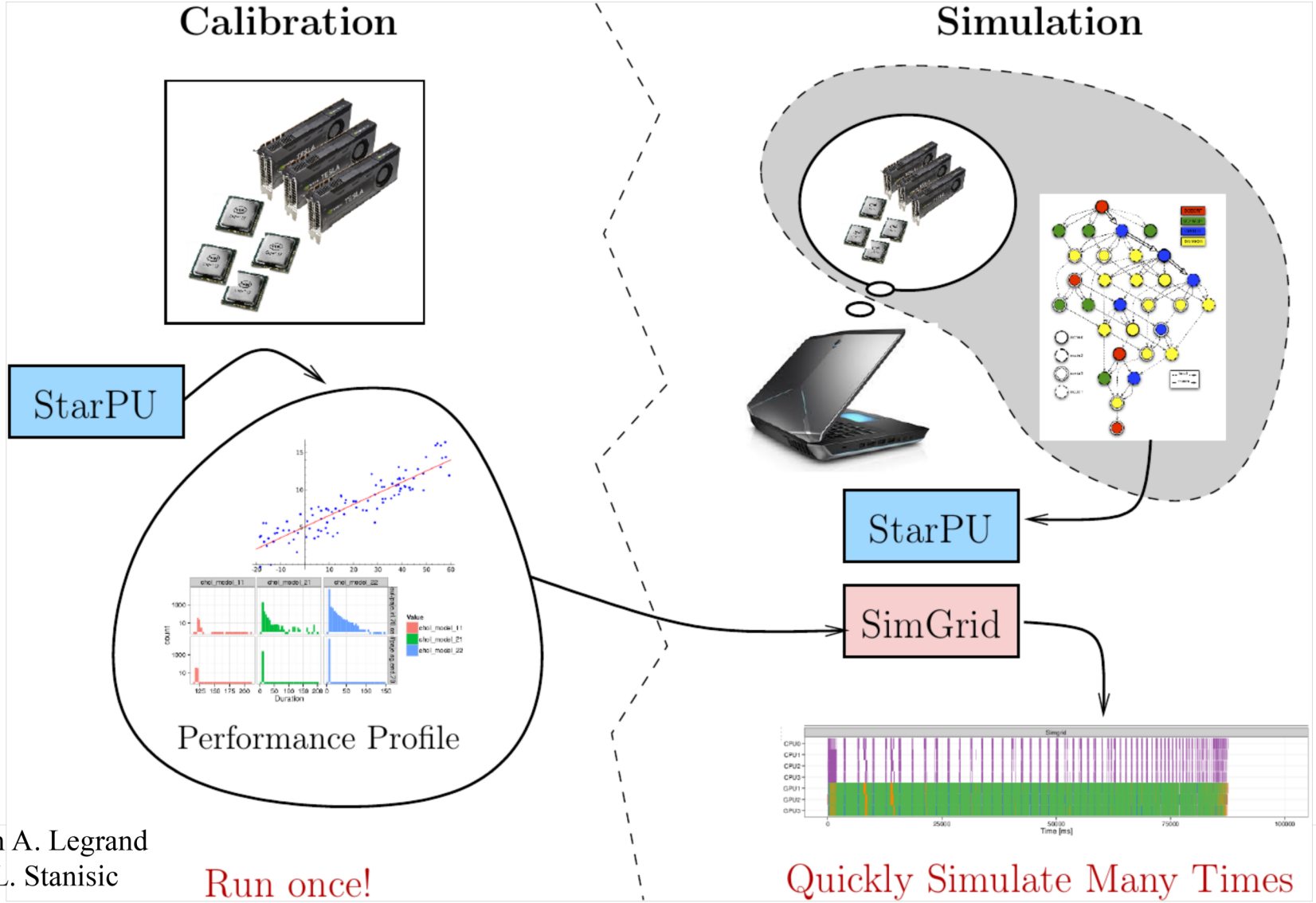
Cholesky cluster performance

@CEA: 144 nodes with 8 CPU cores (E5620) + 2 GPUs (M2090)



Simulation

Simulation with SimGrid



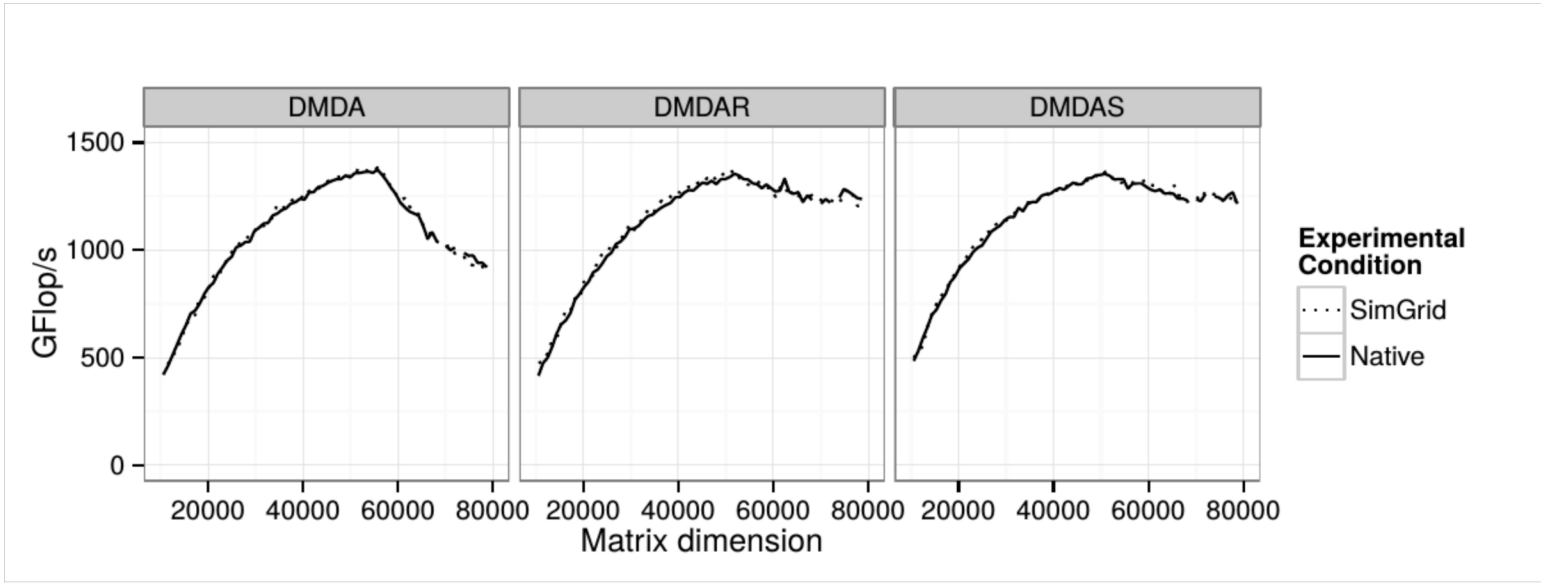
From A. Legrand and L. Stanisic

Simulation with SimGrid

- Run application natively on target system
  - Records performance models
- Rebuild application against simgrid-compiled StarPU
- Run again
  - Uses performance model estimations instead of actually executing tasks

- Way faster execution time
- Reproducible experiments
- No need to run on target system
- Can change system architecture

Simulation with SimGrid

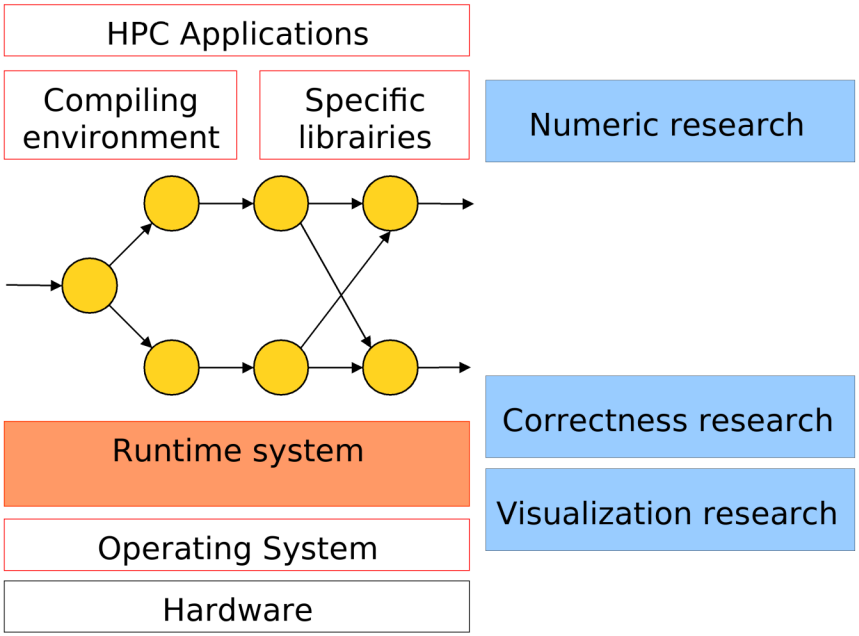


- Way faster execution time
- Reproducible experiments
- No need to run on target system
- Can change system architecture

Conclusion

Task graphs

- Nice programming model
  - Keep sequential program!
- Optimized execution
- Playground for research
  - Runtime
  - Scheduling
  - Numeric algorithms
  - Statistics
  - Correctness
- Used for various real-world computations
  - Cholesky/QR/LU (dense/sparse/compressed), stencil, CG, CFD, FMM…



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