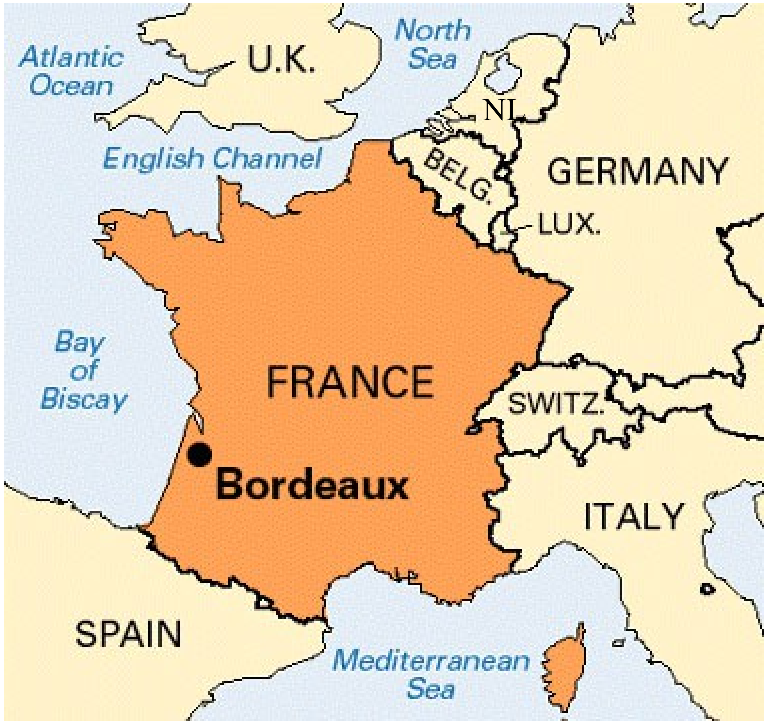
StarPU : Exploiting heterogeneous architectures through dynamic task scheduling

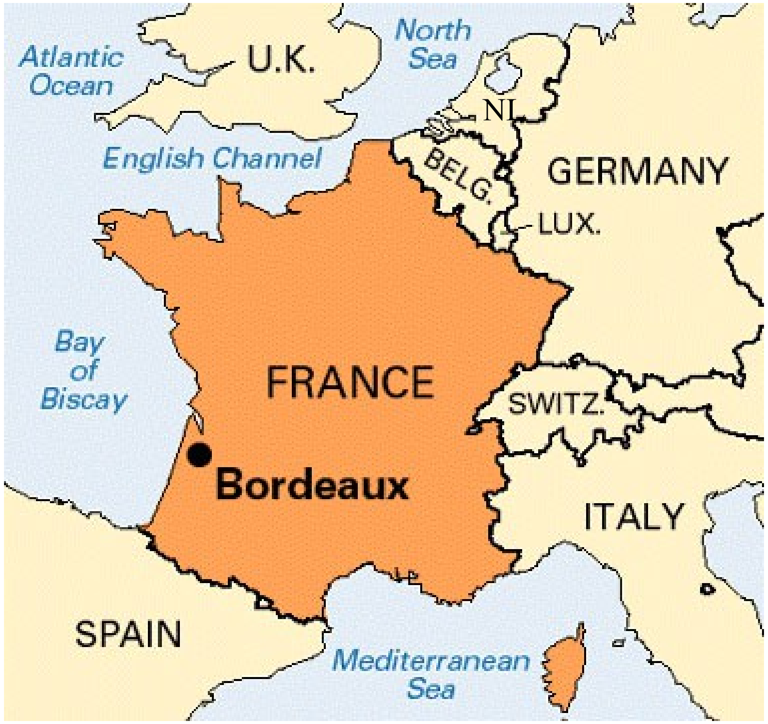
Cédric Augonnet
Samuel Thibault

INRIA Bordeaux, LaBRI, University of Bordeaux

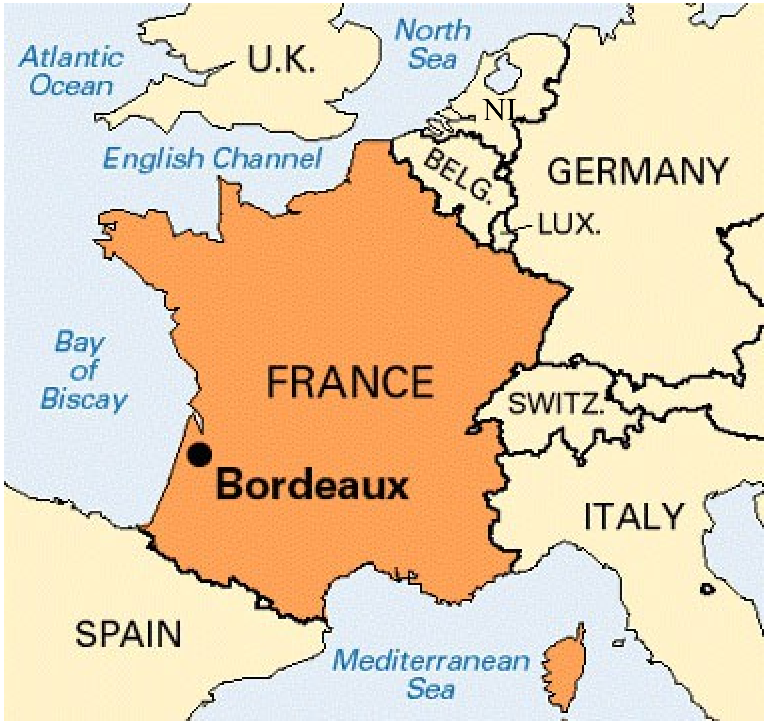
The RUNTIME Team



The RUNTIME Team



The RUNTIME Team



Doing Parallelism for centuries !

The RUNTIME Team

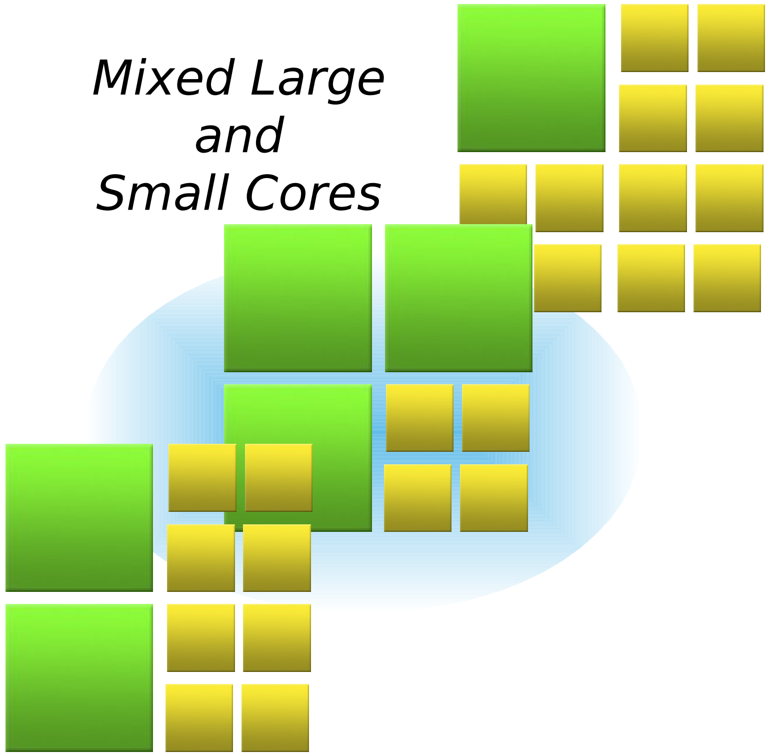
Research directions

- 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
- Our aim
  - Design efficient runtime systems
  - Approaching the raw performance of hardware
  - While preserving application portability: **performance portability**
- See http://runtime.bordeaux.inria.fr/ for more information

Introduction

Toward heterogeneous multi-core architectures

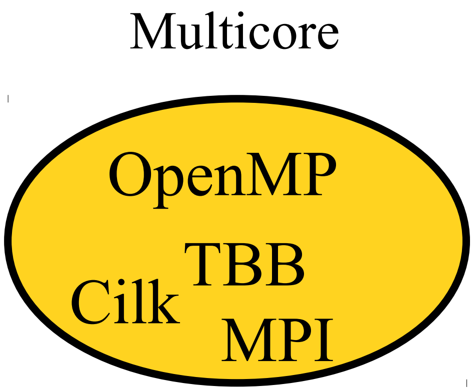
- Multicore is here
  - Hierarchical architectures
  - Manycore
- Architecture specialization
  - Now
    - Accelerators (GPGPUs, FPGAs)
    - Coprocessors (Cell)
  - In the near Future
    - Many simple cores
    - A few full-featured cores
  - Intel MIC, SCC

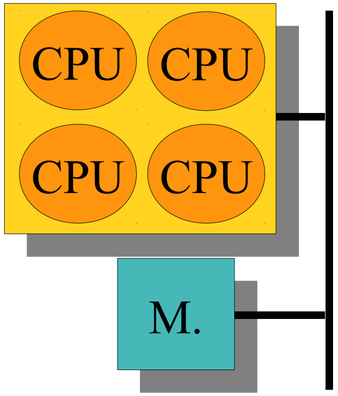


Introduction

How to program these architectures?

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

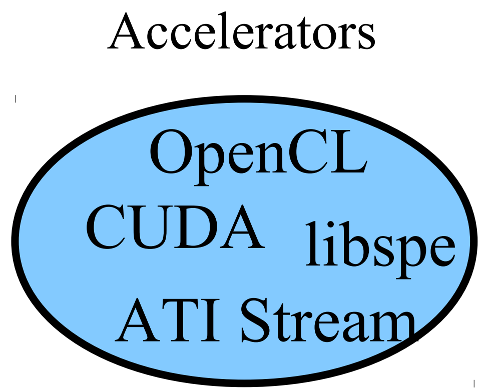


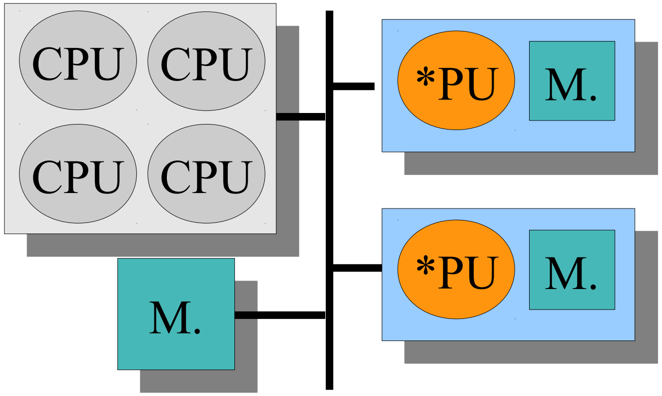


Introduction

How to program these architectures?

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

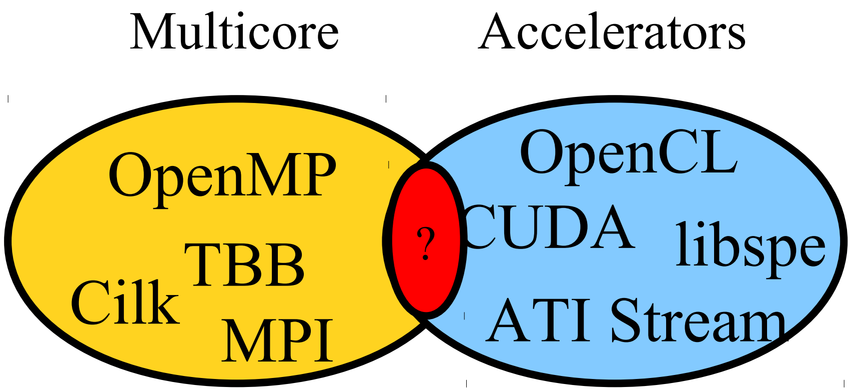


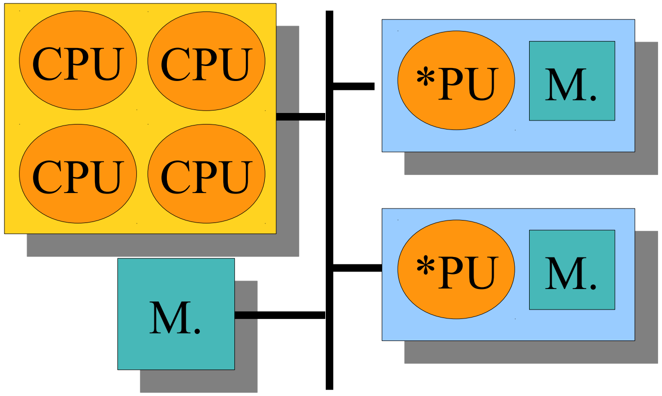


Introduction

How to program these architectures?

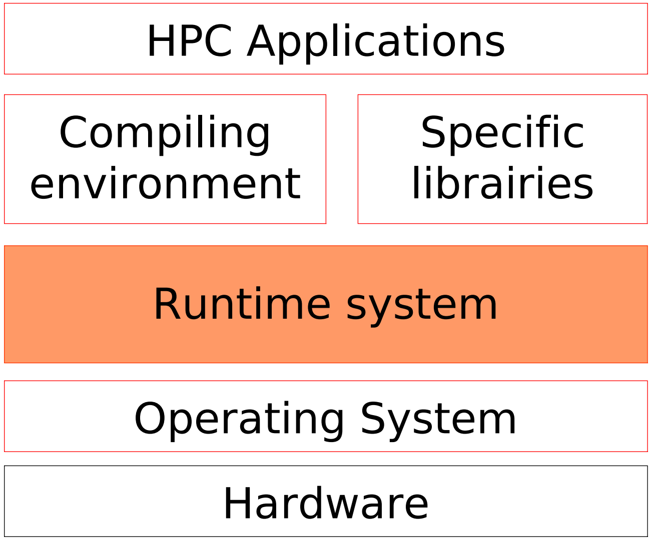
- Multicore programming
  - pthreads, OpenMP, TBB, ...
- Accelerator programming
  - Consensus on OpenCL?
  - (Often) Pure offloading model
- Hybrid models?
  - **Take advantage of all resources ☺**
  - **Complex interactions ☹**





Introduction
Challenging issues at all stages

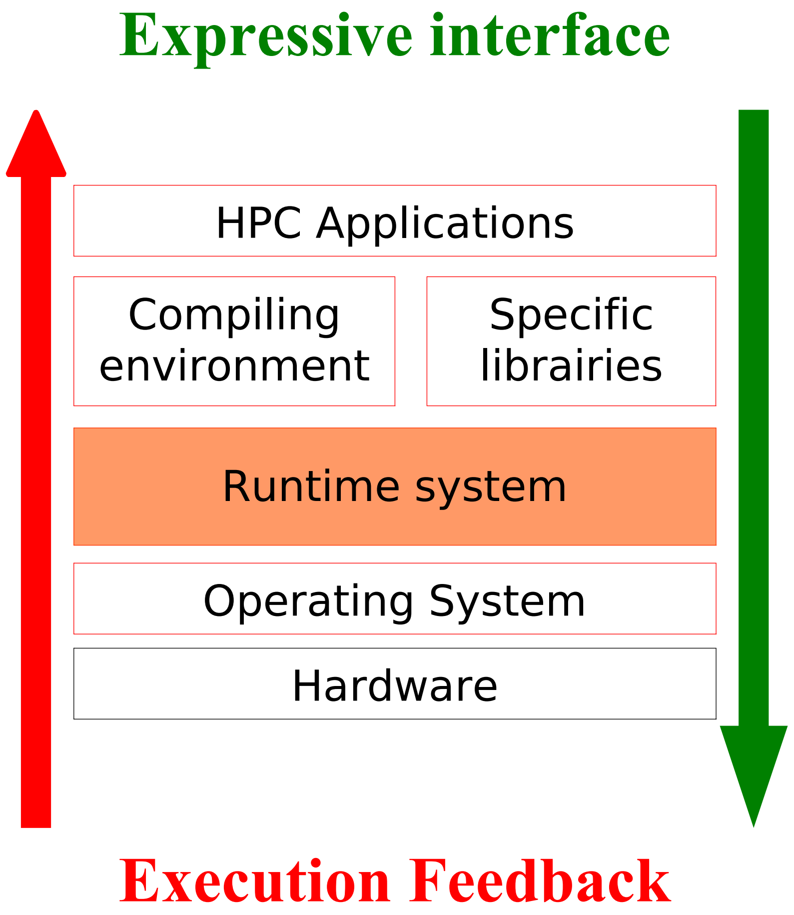
- Applications
  - Programming paradigm
  - BLAS kernels, FFT, …
- Compilers
  - Languages
  - Code generation/optimization
- Runtime systems
  - Resources management
  - Task scheduling
- Architecture
  - Memory interconnect



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



Overview of StarPU

Overview of StarPU

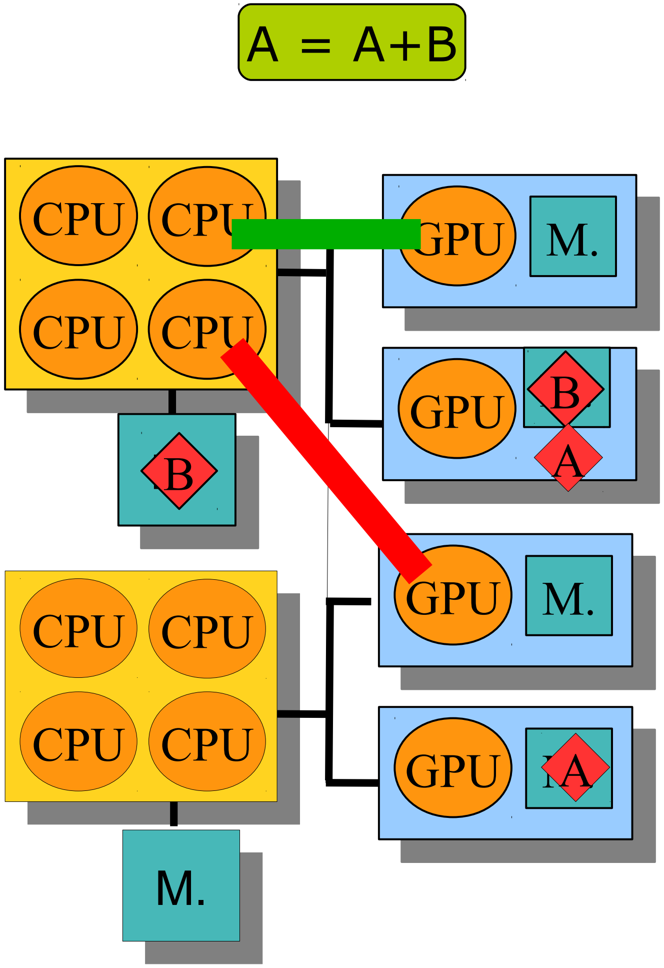
Rationale

Task scheduling

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

Memory transfer

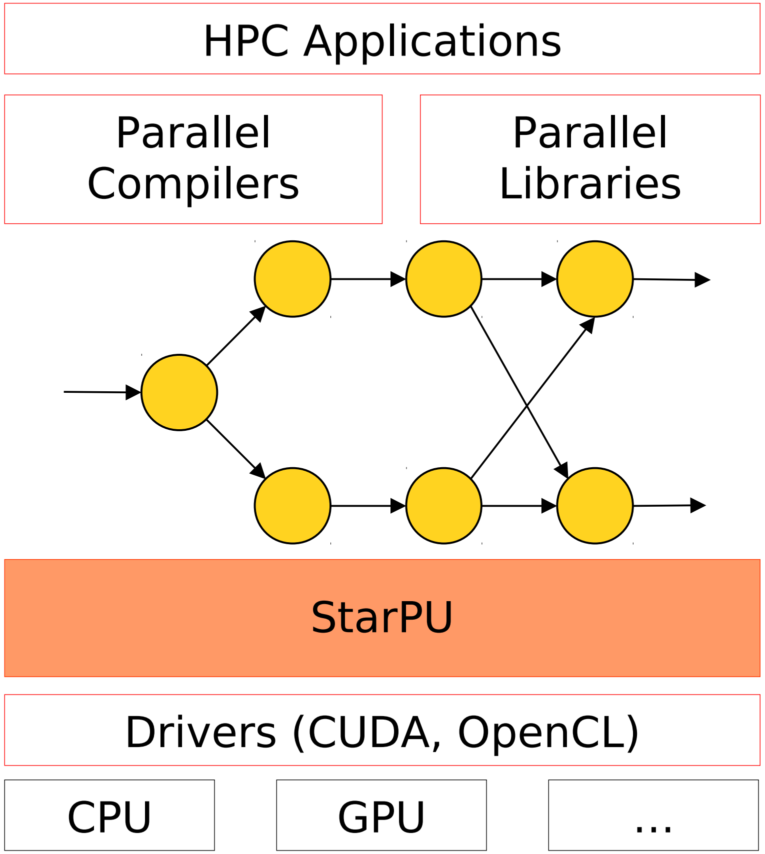
- Eliminate redundant transfers
- Software VSM (Virtual Shared Memory)



The StarPU runtime system

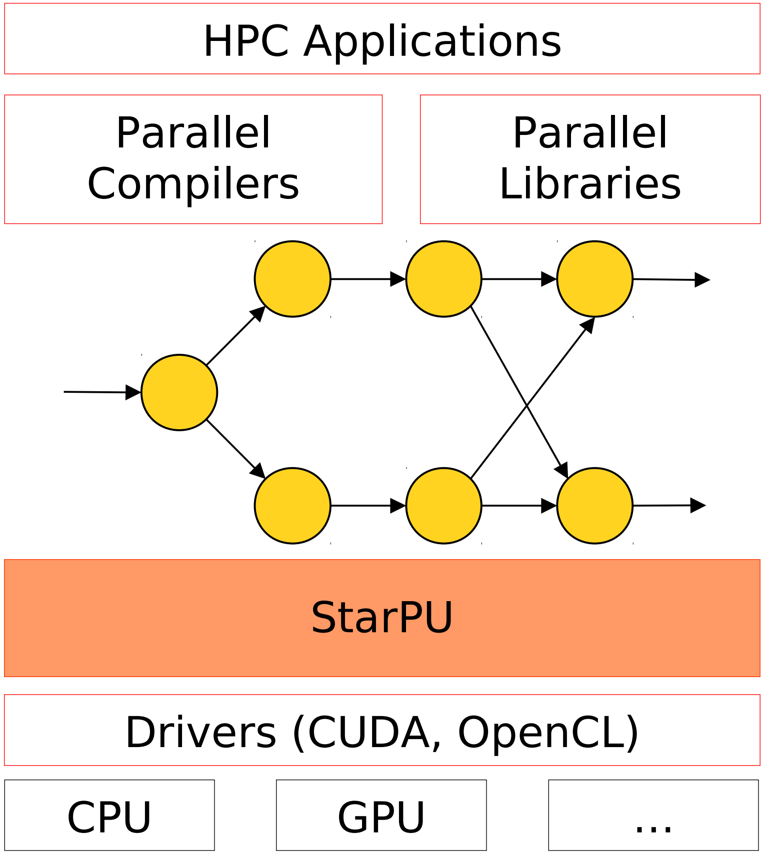
The need for runtime systems

- "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
    - Weak consistency
    - Replication
    - Single writer
    - High level API
        - Partitioning filters
- Input & ouput of tasks = reference to VSM data

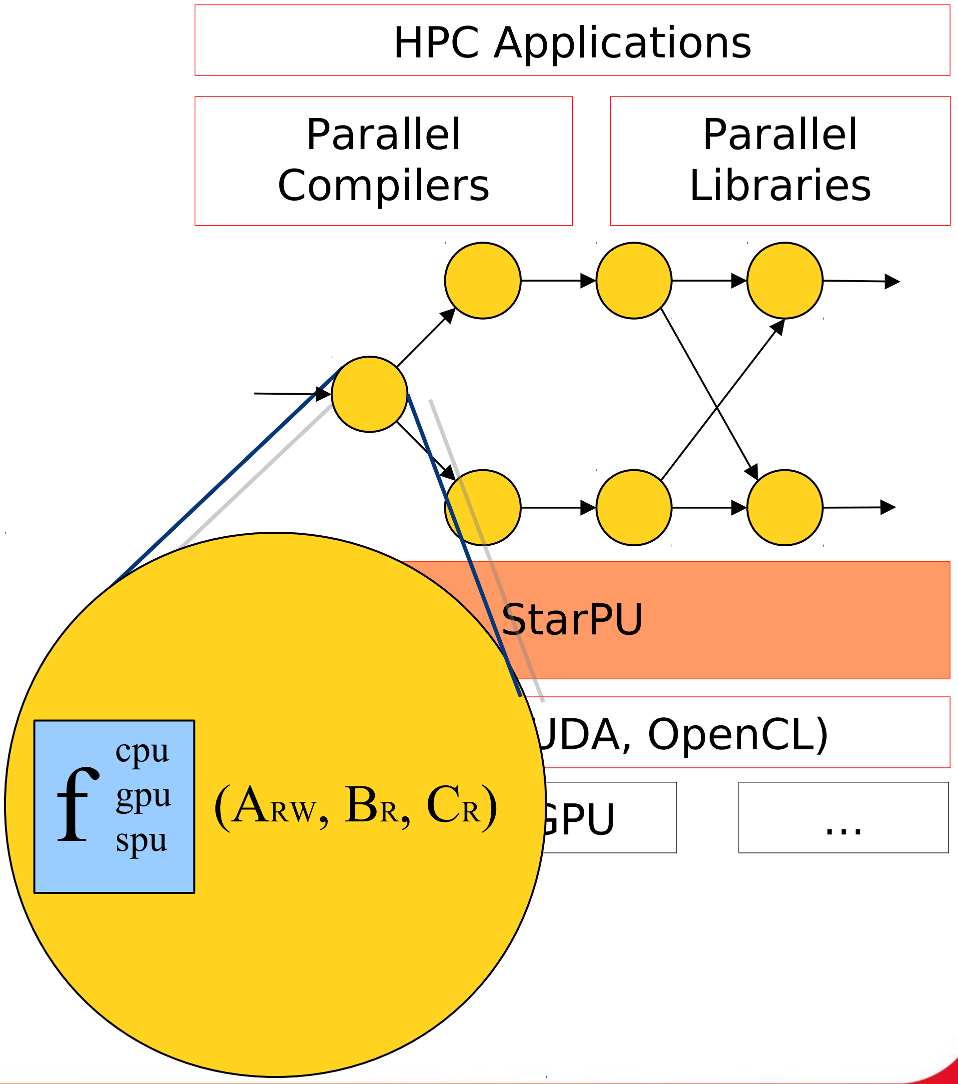


The StarPU runtime system

Task scheduling

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

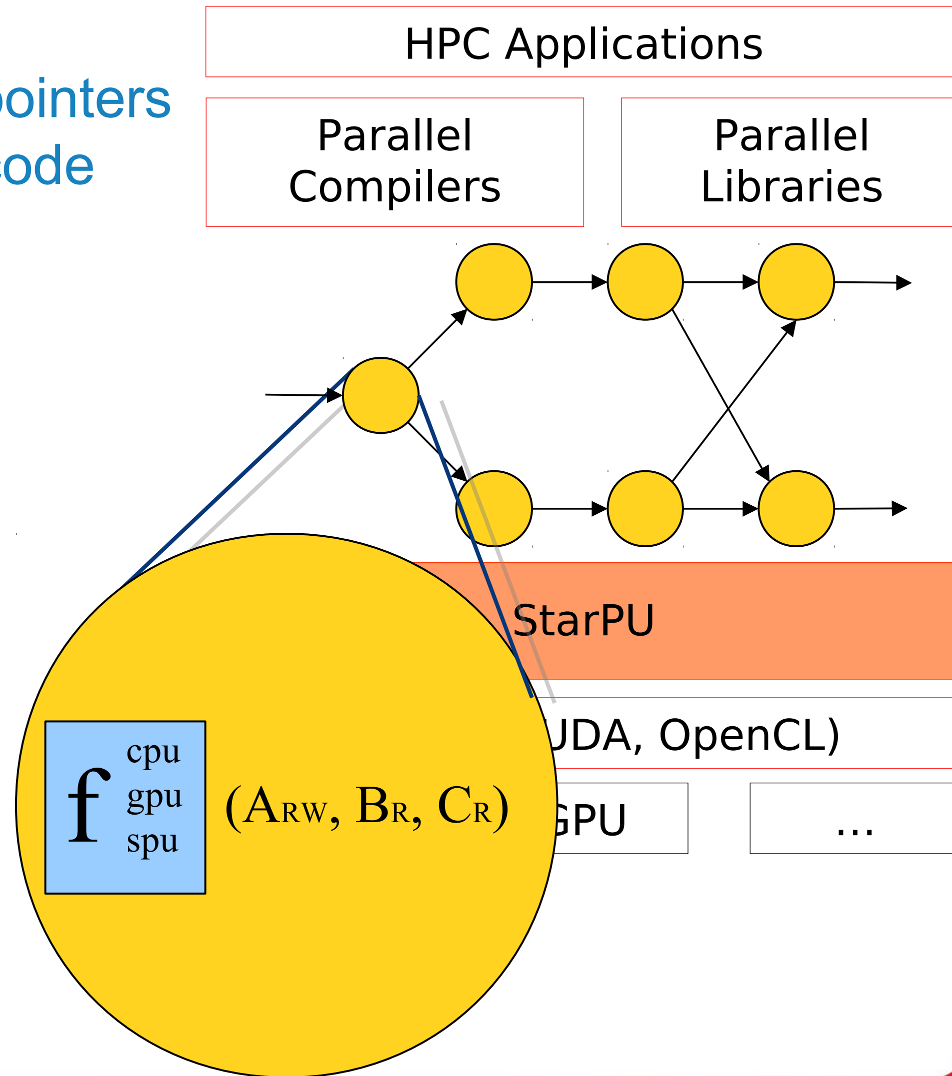
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



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...

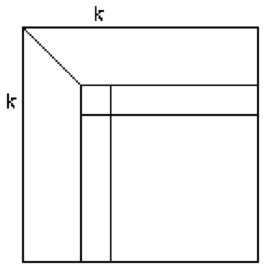


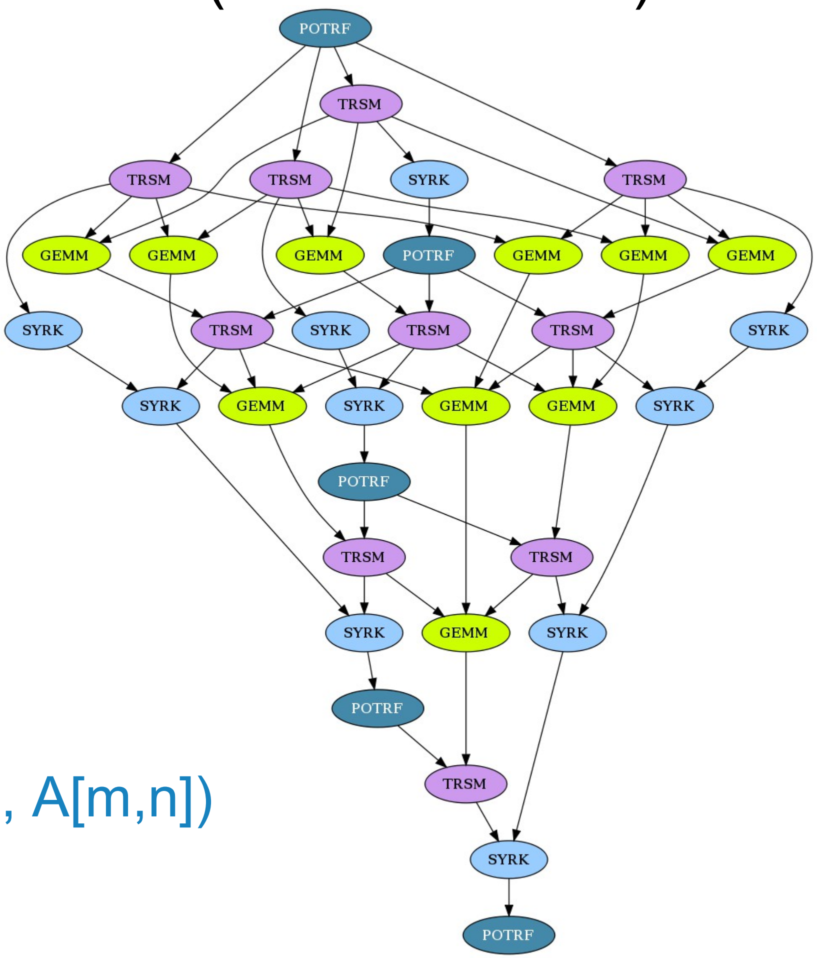
Task management

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

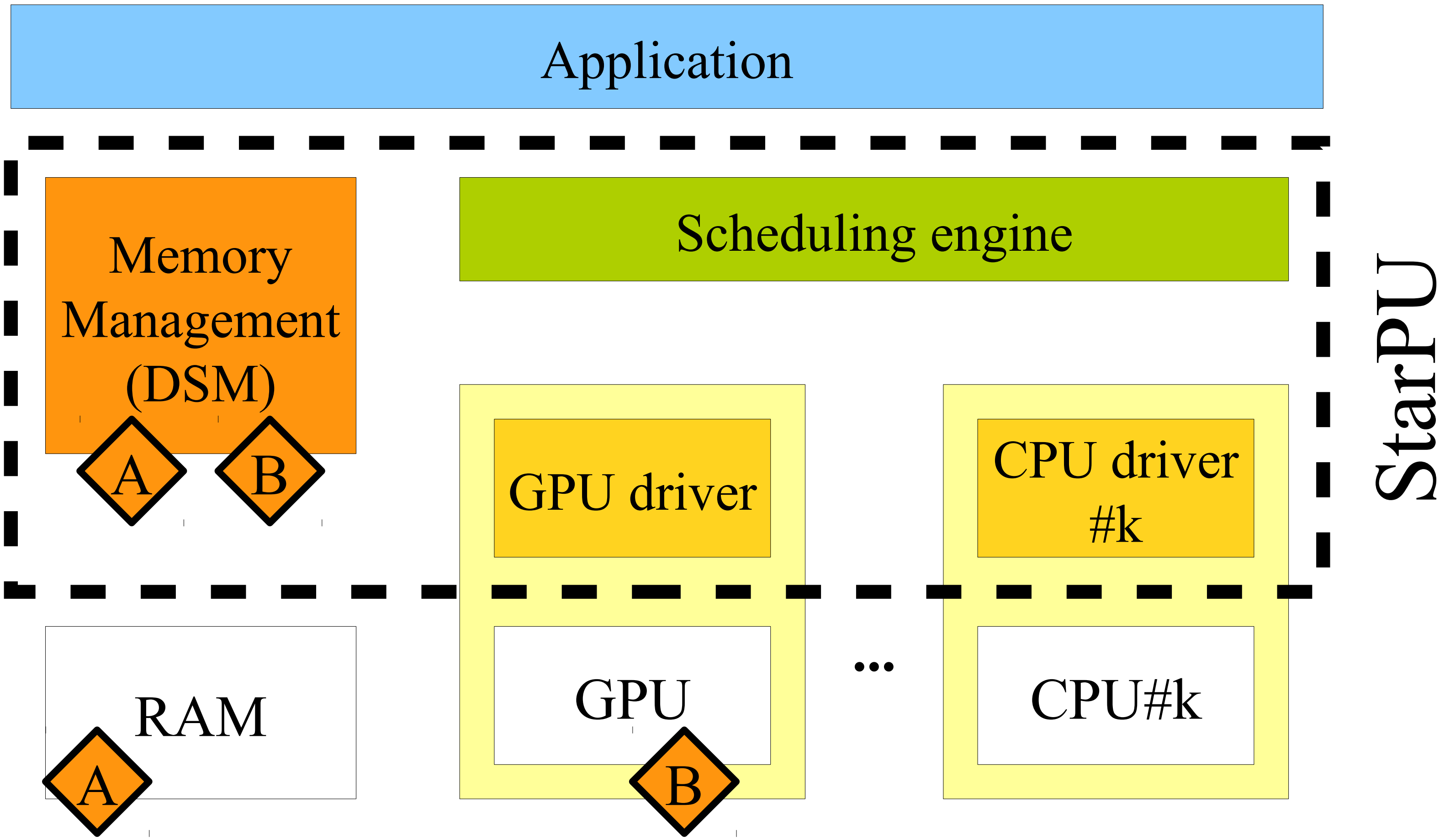
```
For (k = 0 .. tiles – 1)
{
    POTRF(A[k,k])
    for (m = k+1 .. tiles – 1)
        TRSM(A[k,k], A[m,k])
    for (m = k+1 .. tiles – 1)
        SYRK(A[m,k], A[m,m])

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





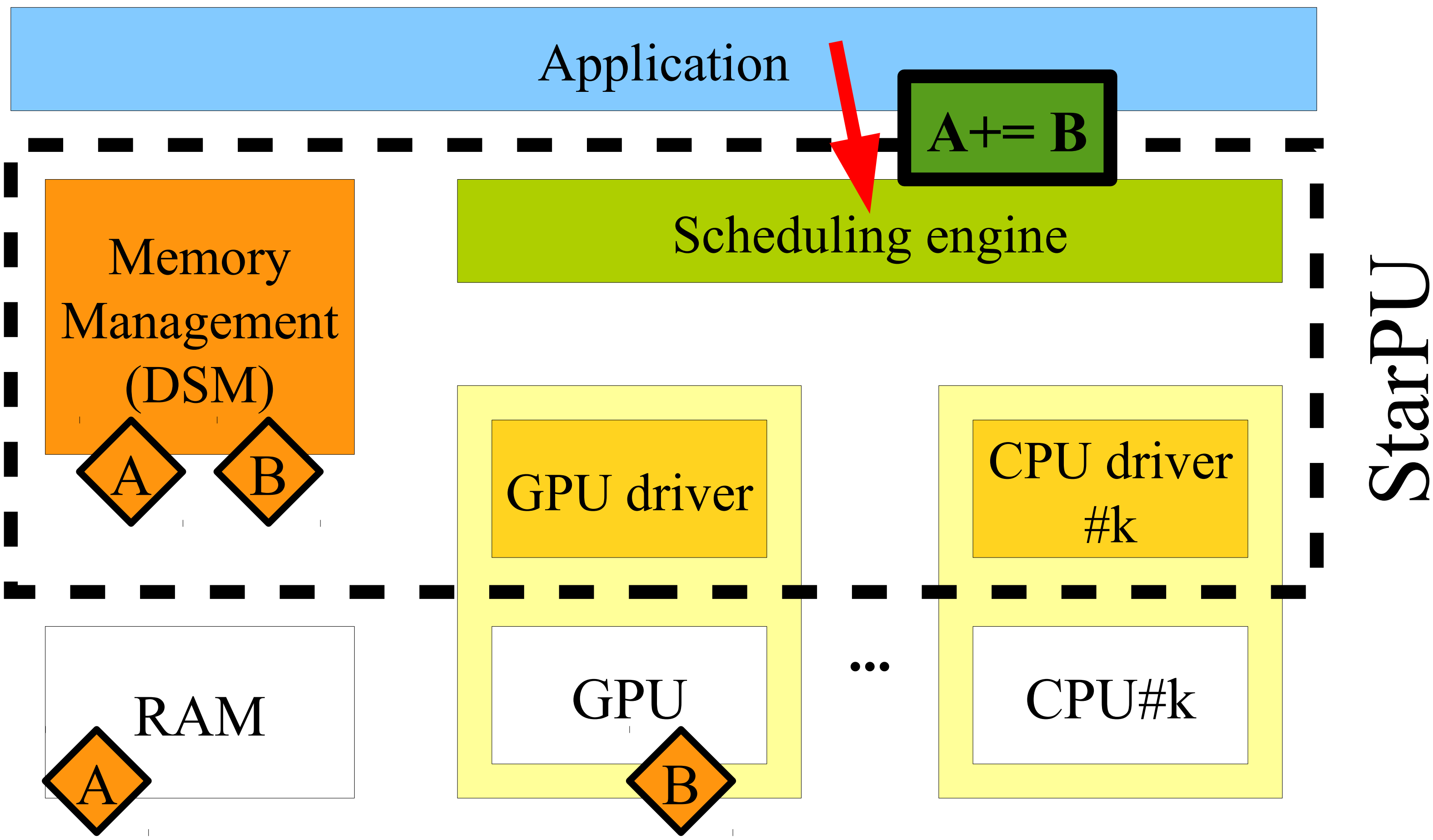
The StarPU runtime system

Execution model



The StarPU runtime system

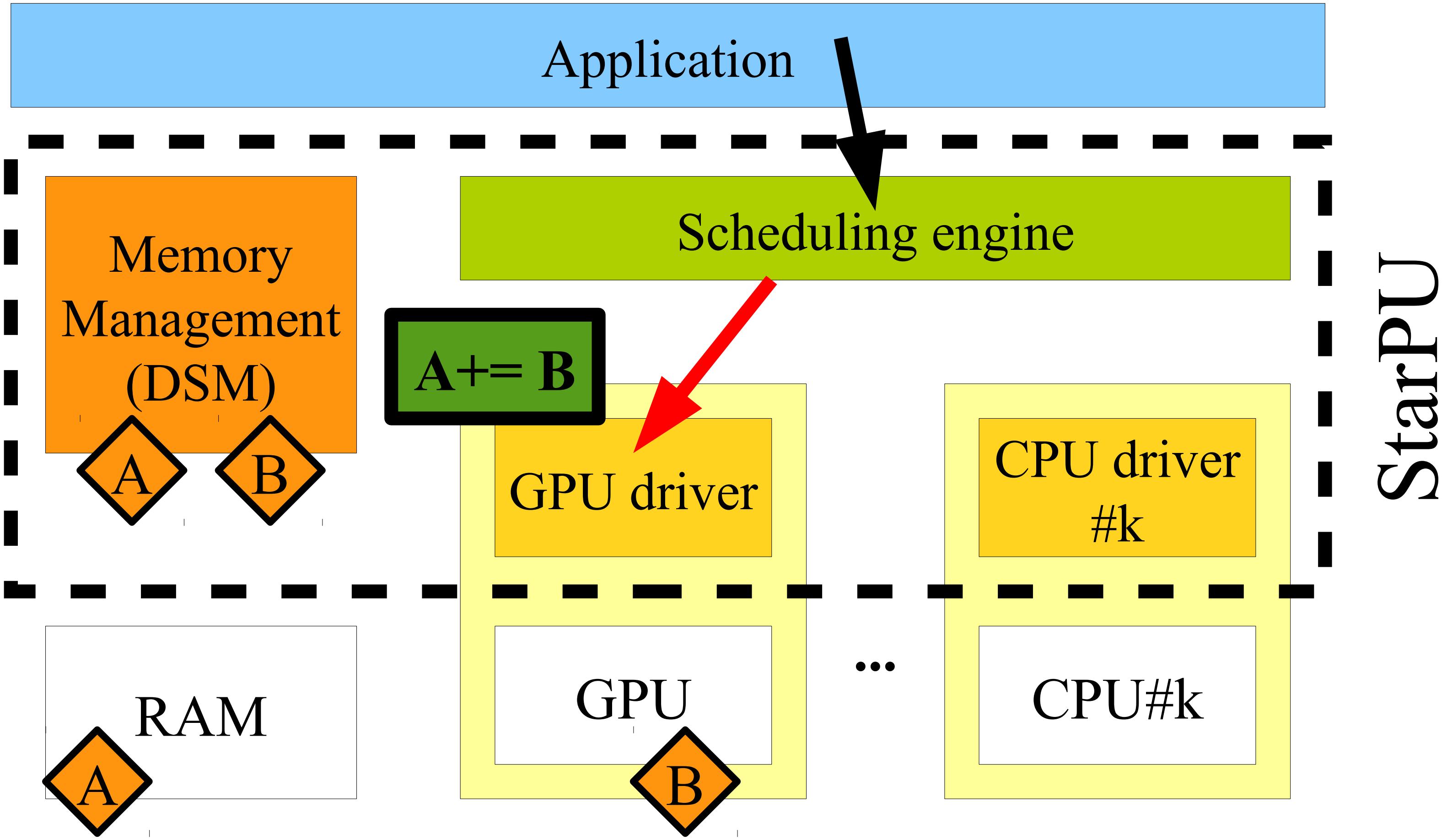
Execution model



Submit task « A += B »

The StarPU runtime system

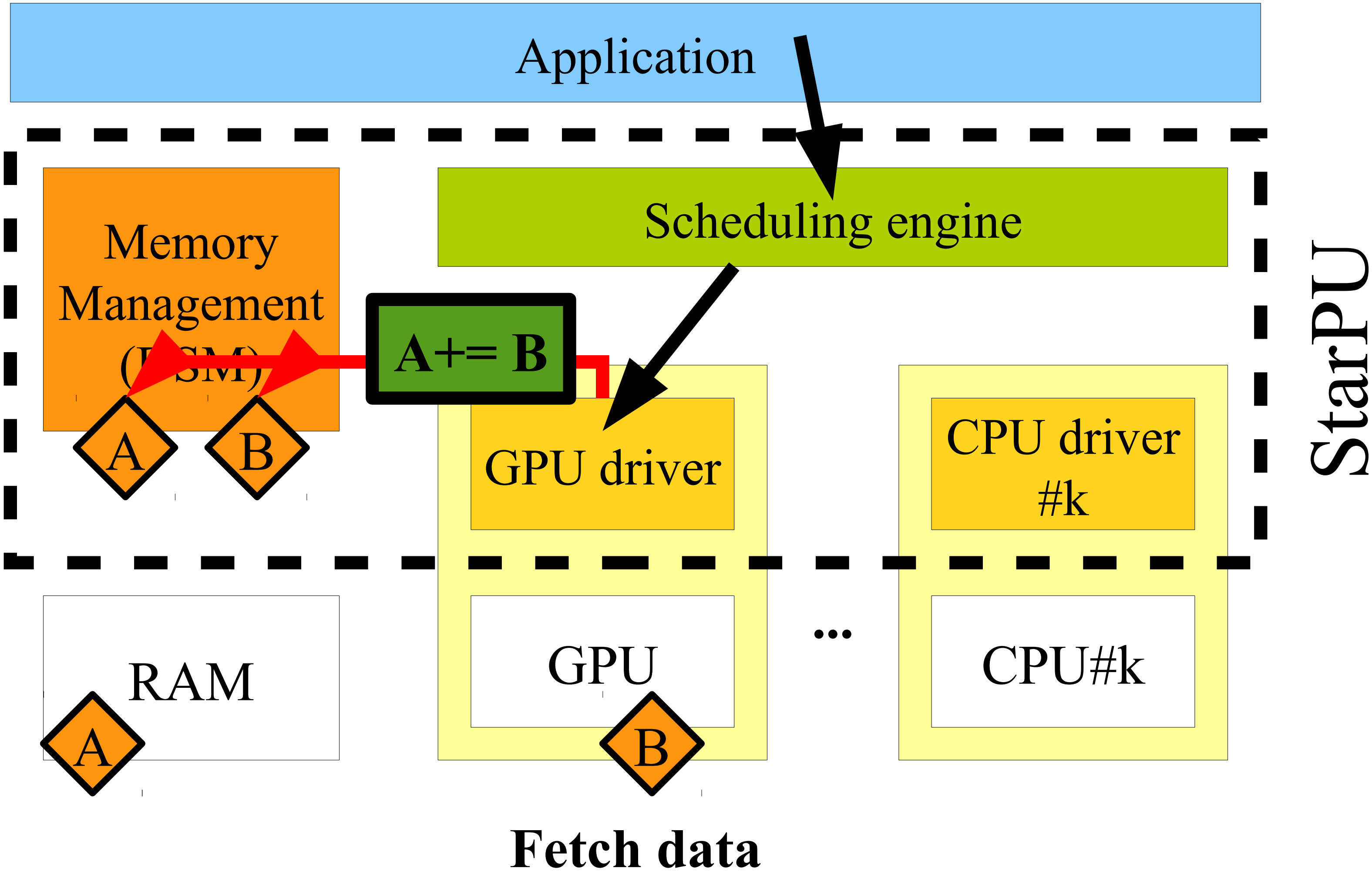
Execution model



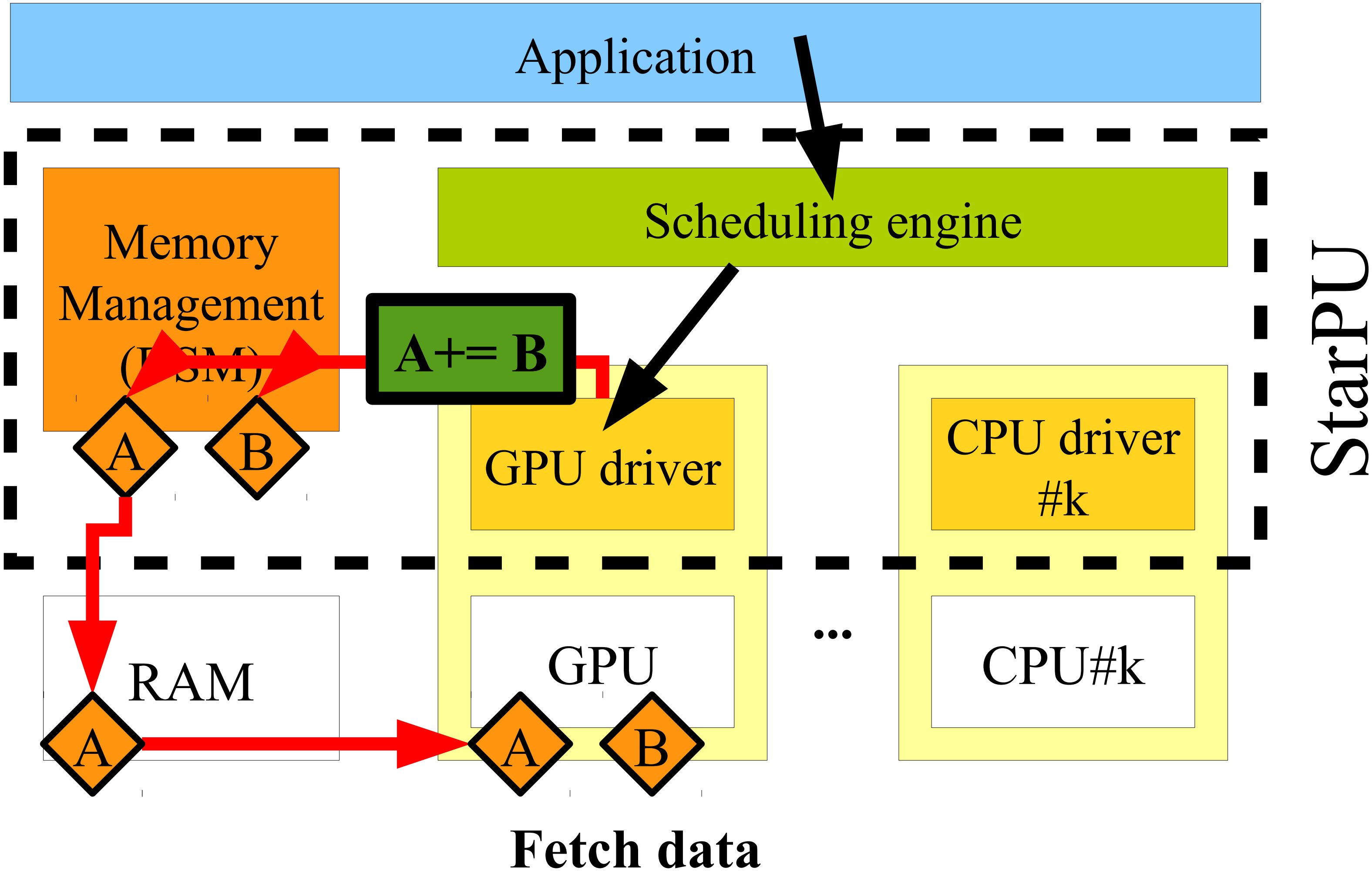
Schedule task

The StarPU runtime system

Execution model

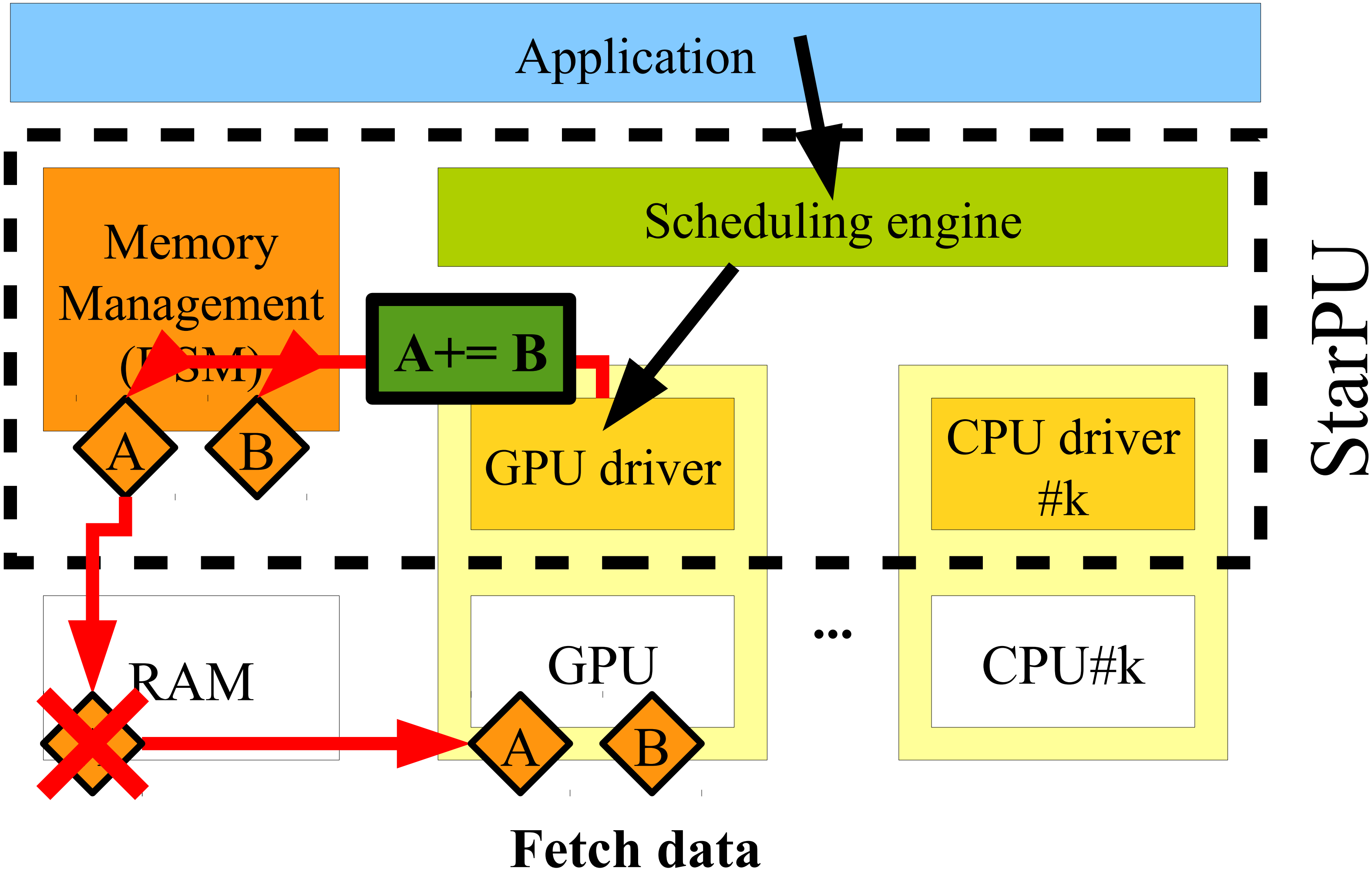


The StarPU runtime system
Execution model



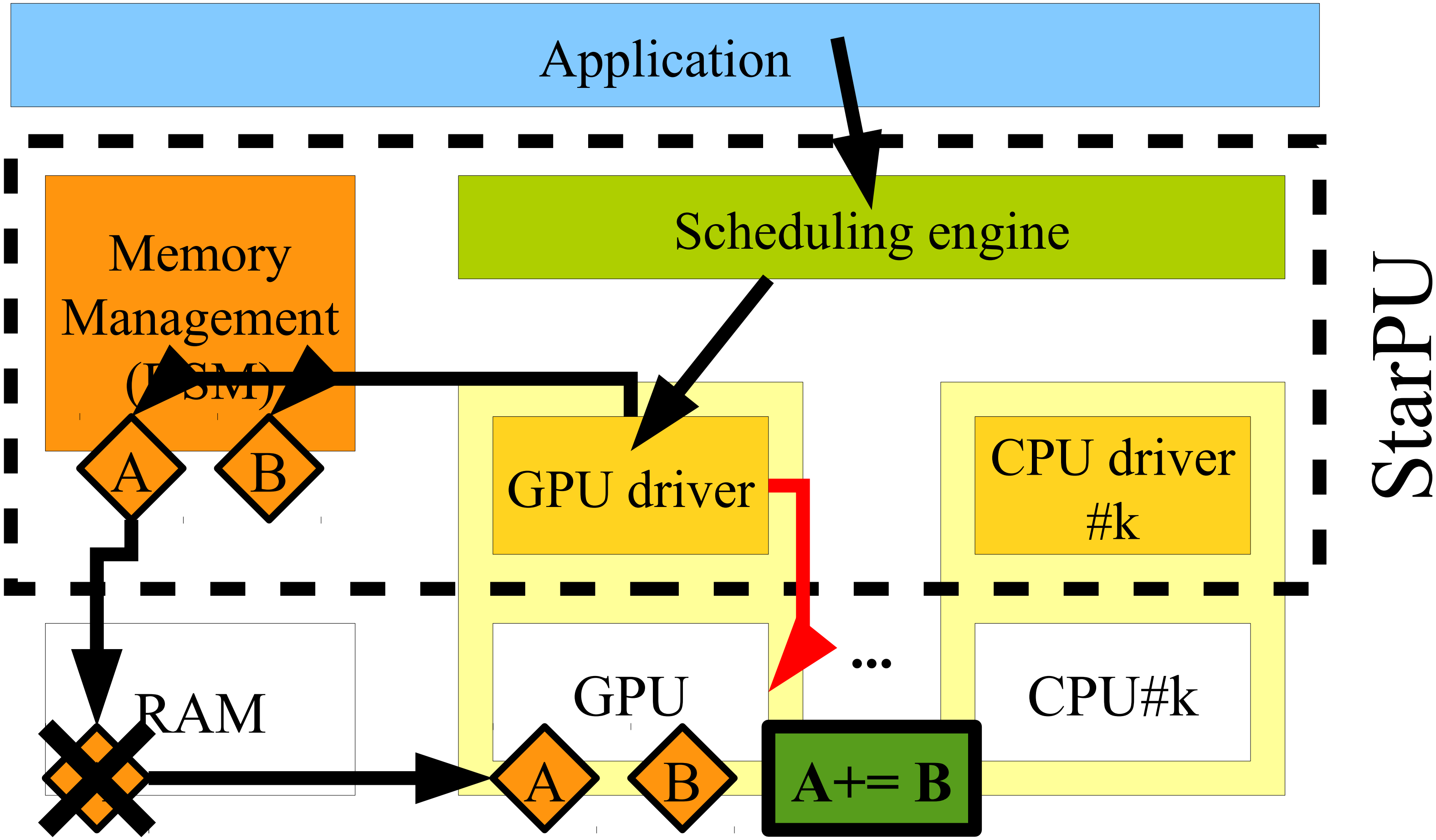
The StarPU runtime system

Execution model



The StarPU runtime system

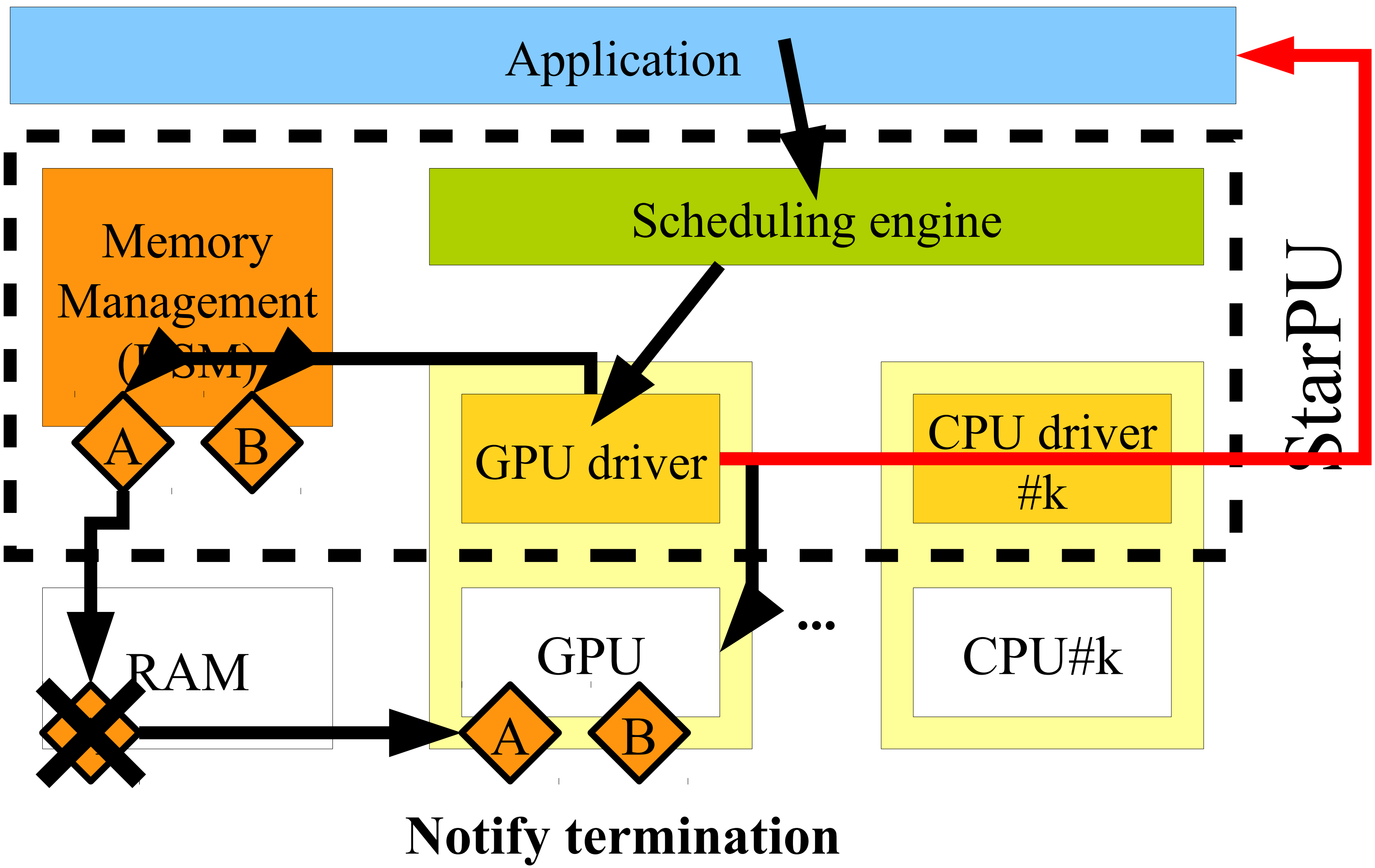
Execution model



Offload computation

The StarPU runtime system

Execution model



The StarPU runtime system

Development context

- History
  - Started about 4 years ago
    - PhD Thesis of Cédric Augonnet
  - StarPU main core ~ 40k lines of code
  - Written in C

- 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)
  - Intel MIC, SCC (in private branch)
  - Cell processors (experimental)
- Supported Operating Systems
  - Linux
  - Mac OS
  - Windows

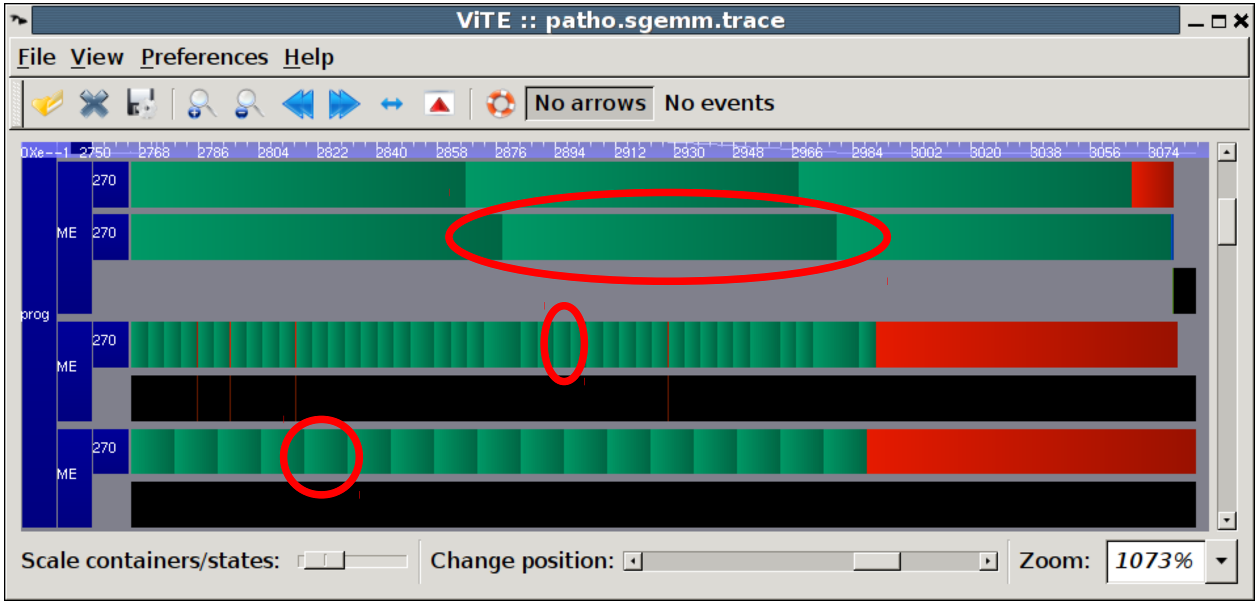
Task Scheduling

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

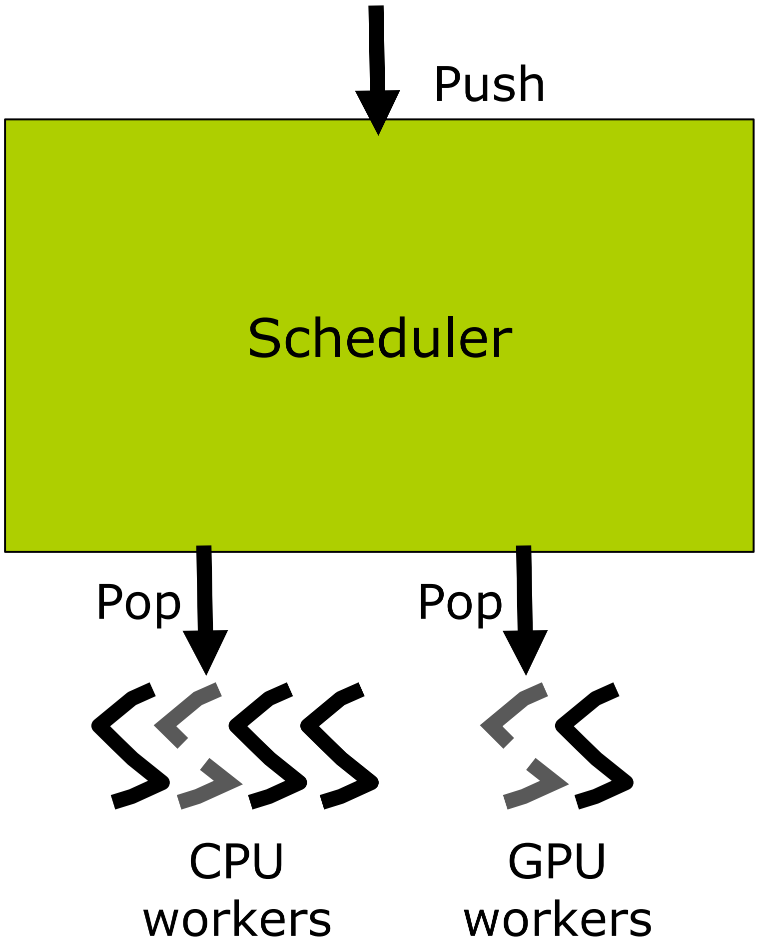
Task scheduling

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



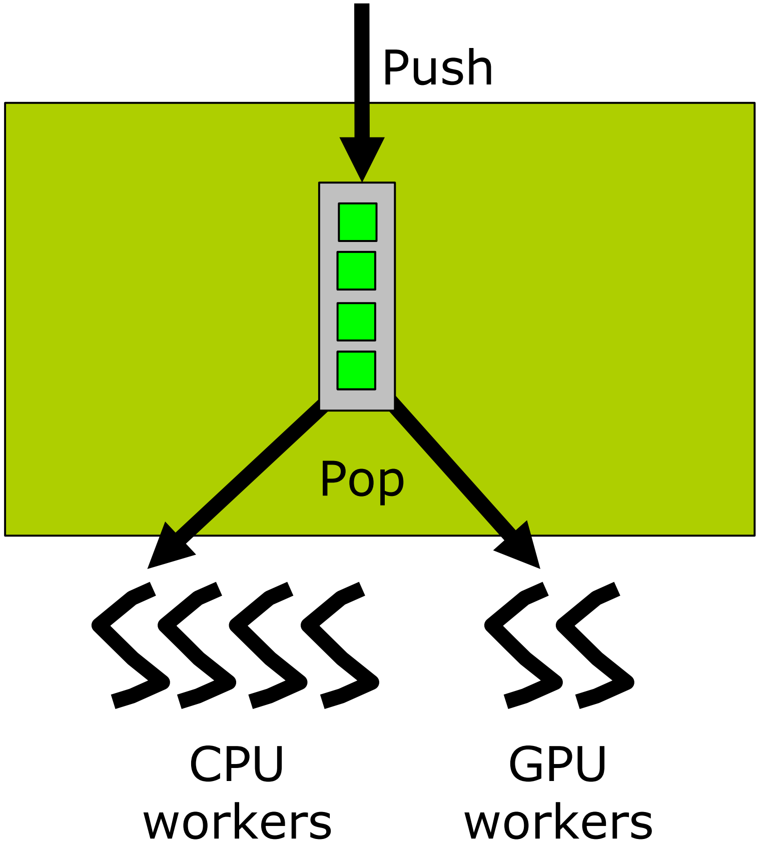
Task scheduling

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



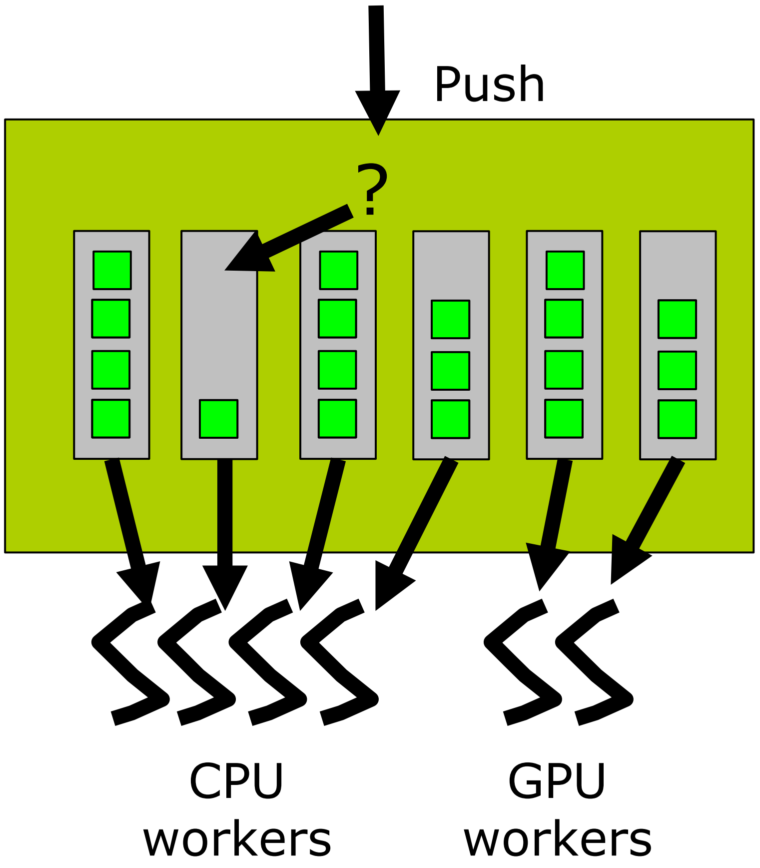
Task scheduling

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

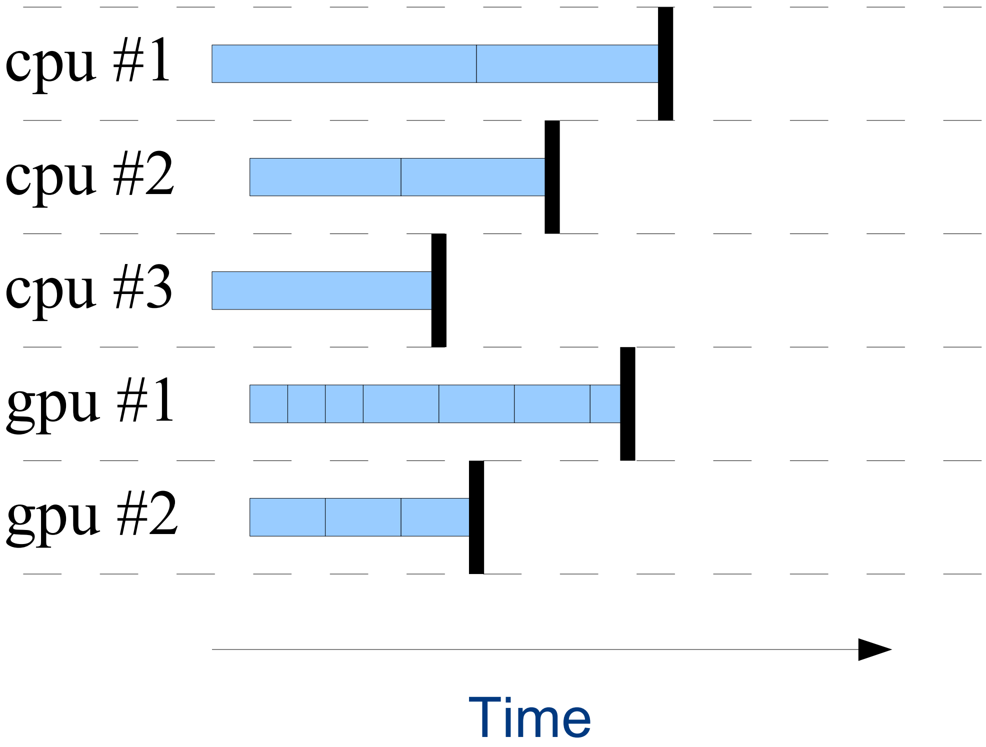
Idle processing units poll for work (“pop”)

Various scheduling policies, can even be user-defined



Prediction-based scheduling
Load balancing

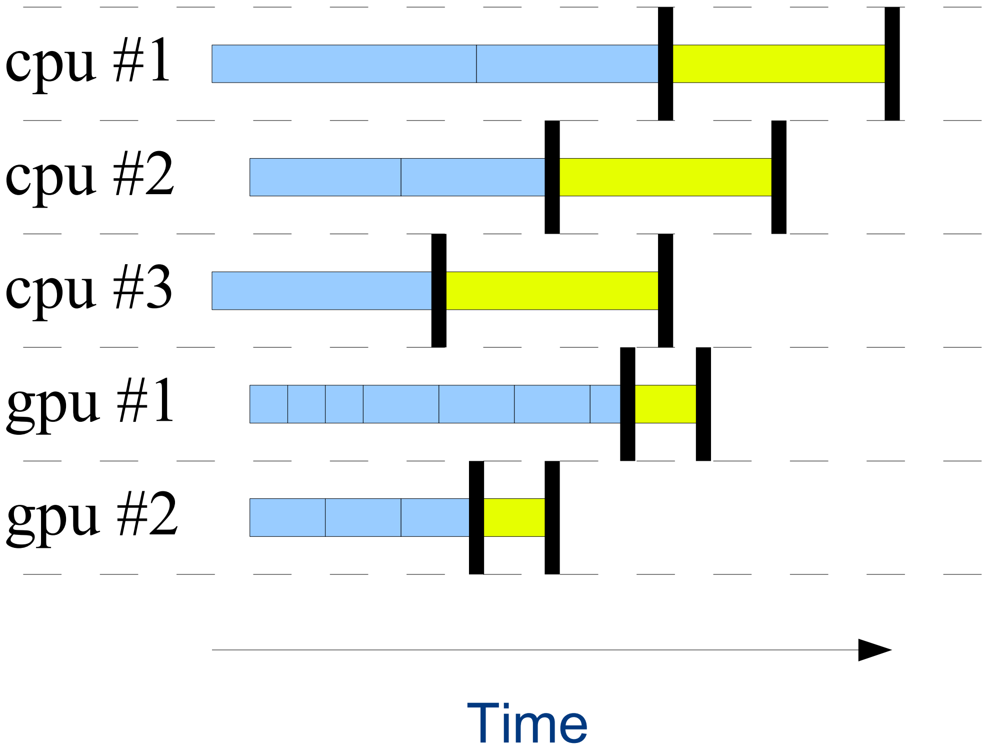
- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time



Prediction-based scheduling

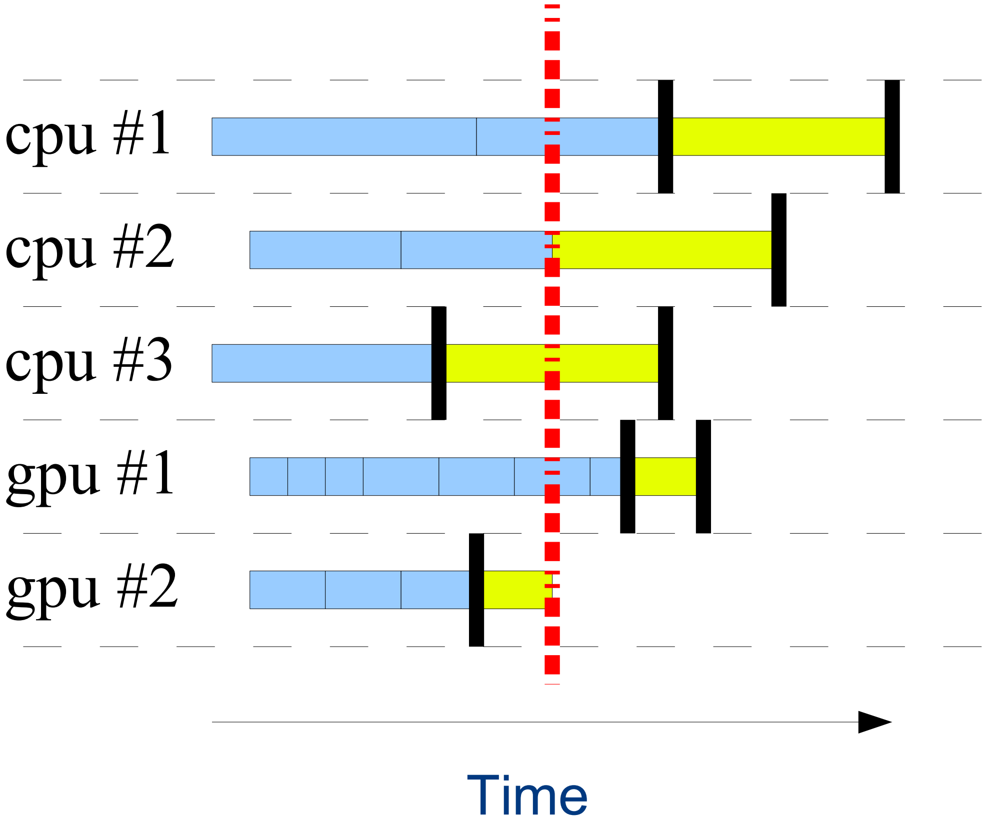
Load balancing

- 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

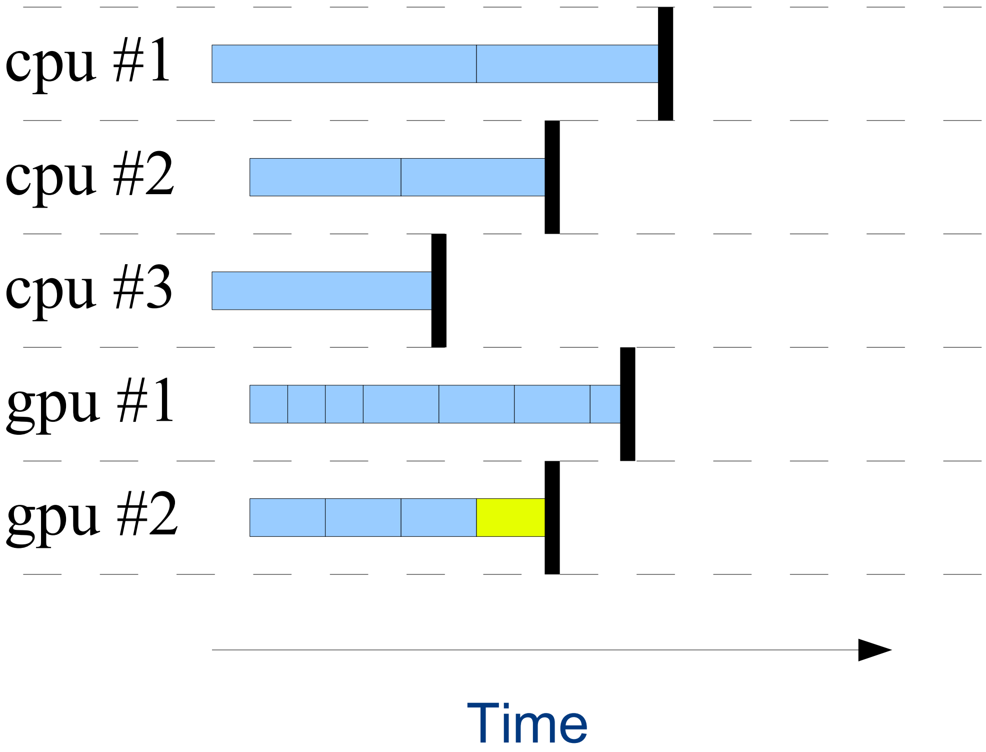
- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model



- 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

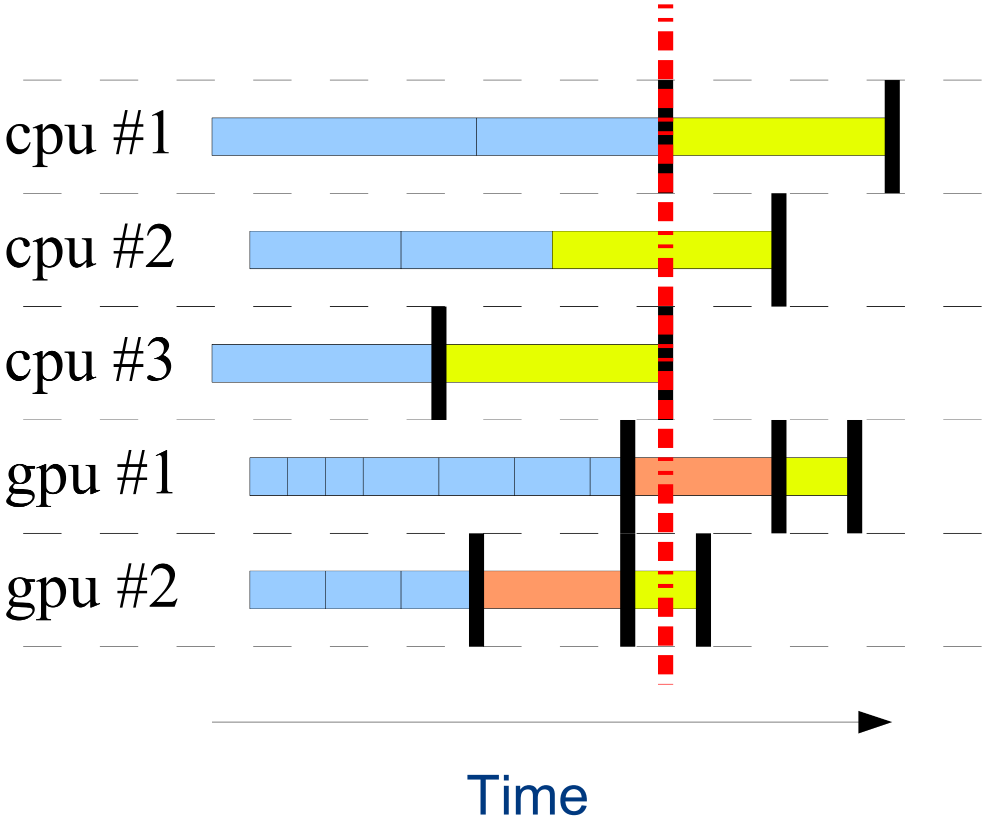


- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time

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



Mixing PLASMA and MAGMA with StarPU

Mixing PLASMA and MAGMA with StarPU

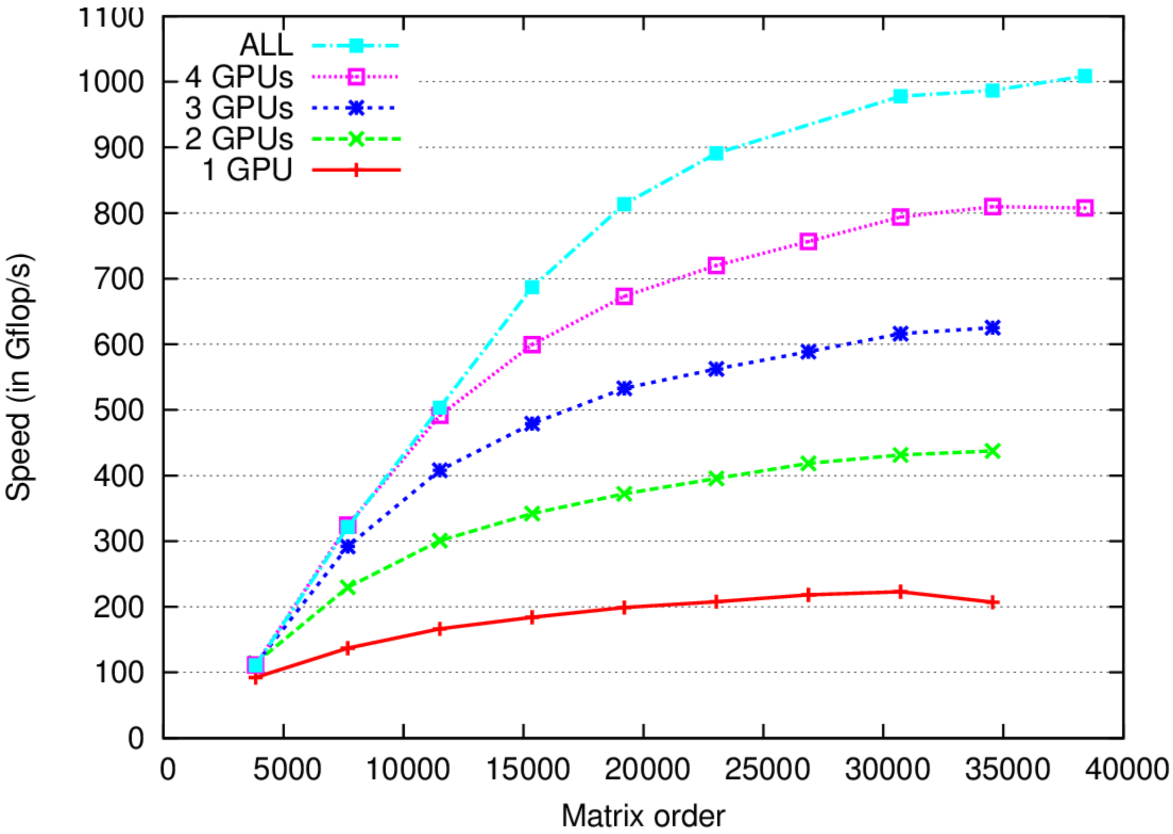
- State of the art algorithms
  - PLASMA (Multicore CPUs)
    - Dynamically scheduled with Quark
  - MAGMA (Multiple GPUs)
    - Hand-coded data transfers
    - Static task mapping

- Design of combination
  - Use PLASMA algorithm with « magnum tiles »
  - PLASMA kernels on CPUs, MAGMA kernels on GPUs
  - Replace the QUARK scheduler with StarPU

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

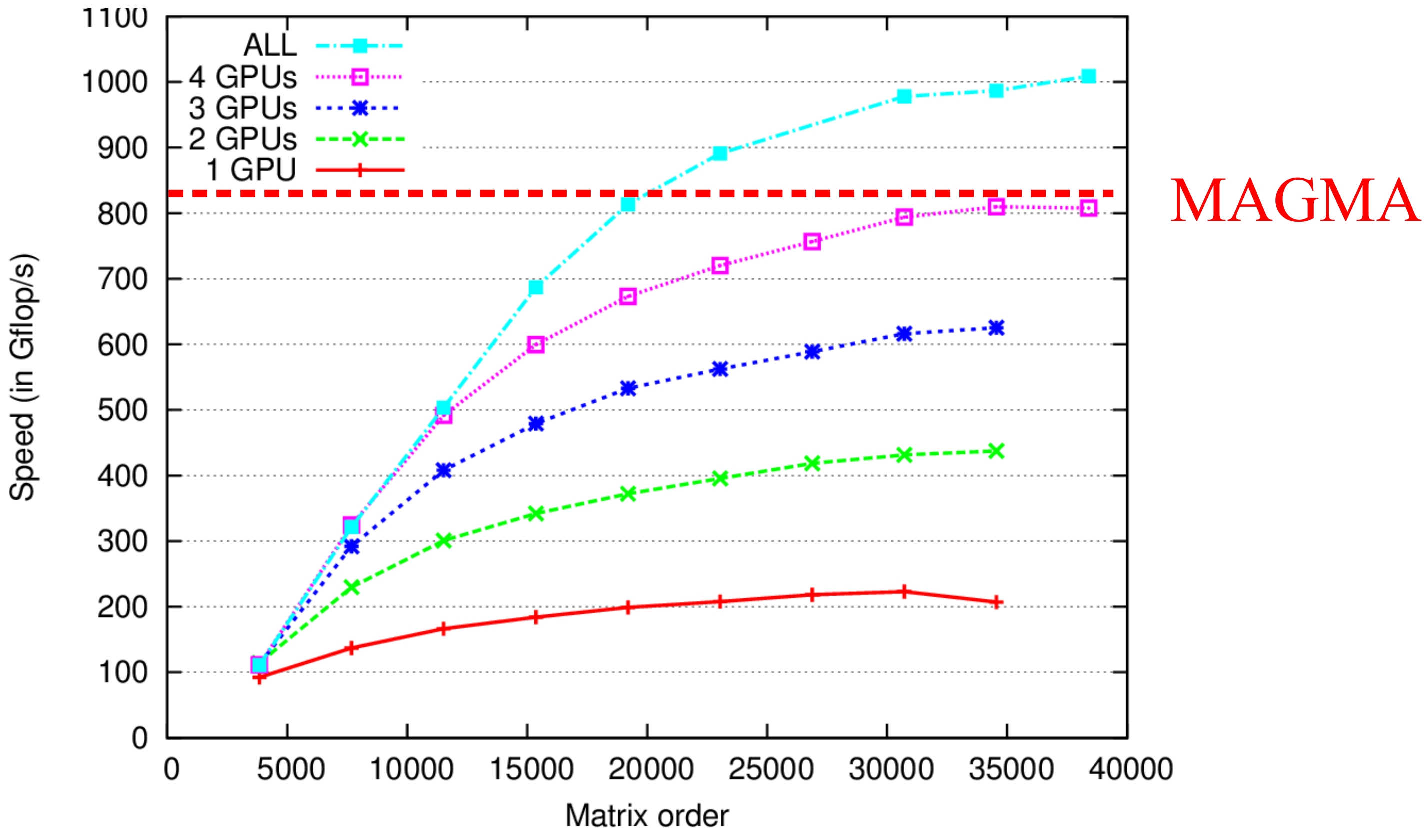
Mixing PLASMA and MAGMA with StarPU

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



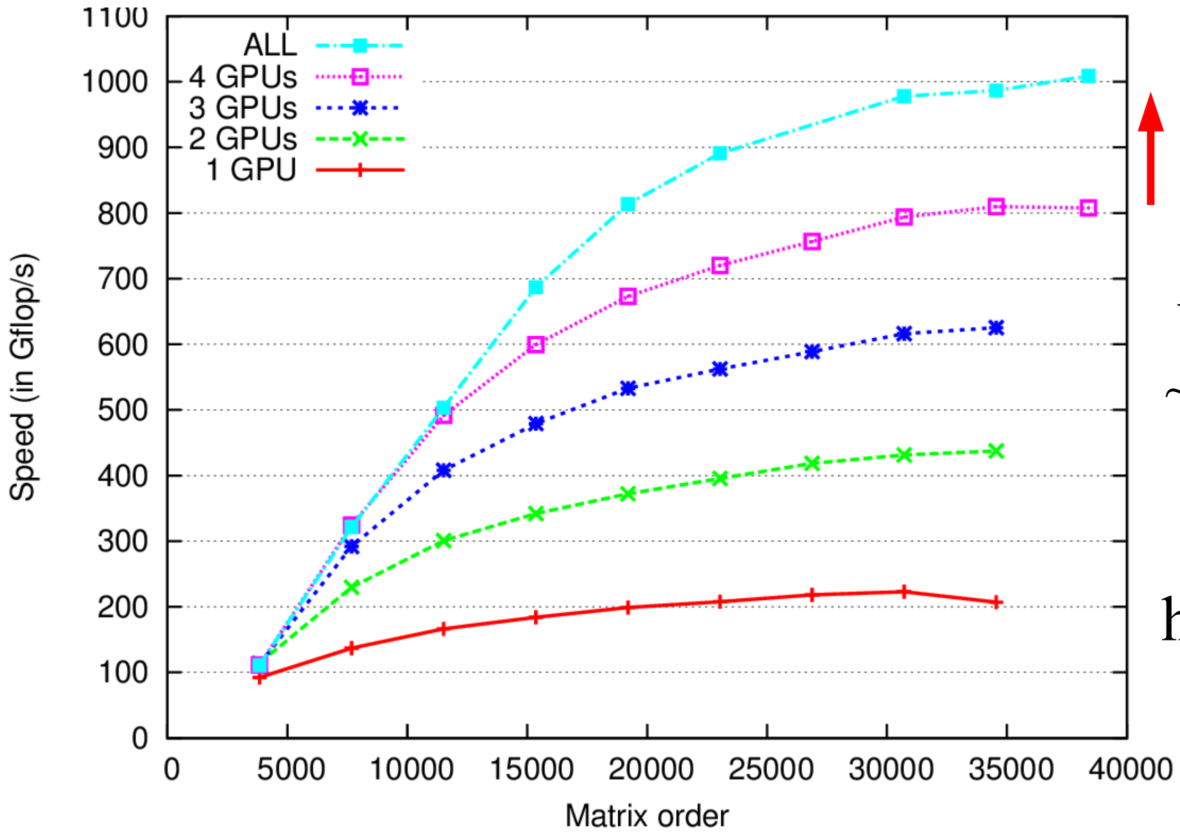
Mixing PLASMA and MAGMA with StarPU

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



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

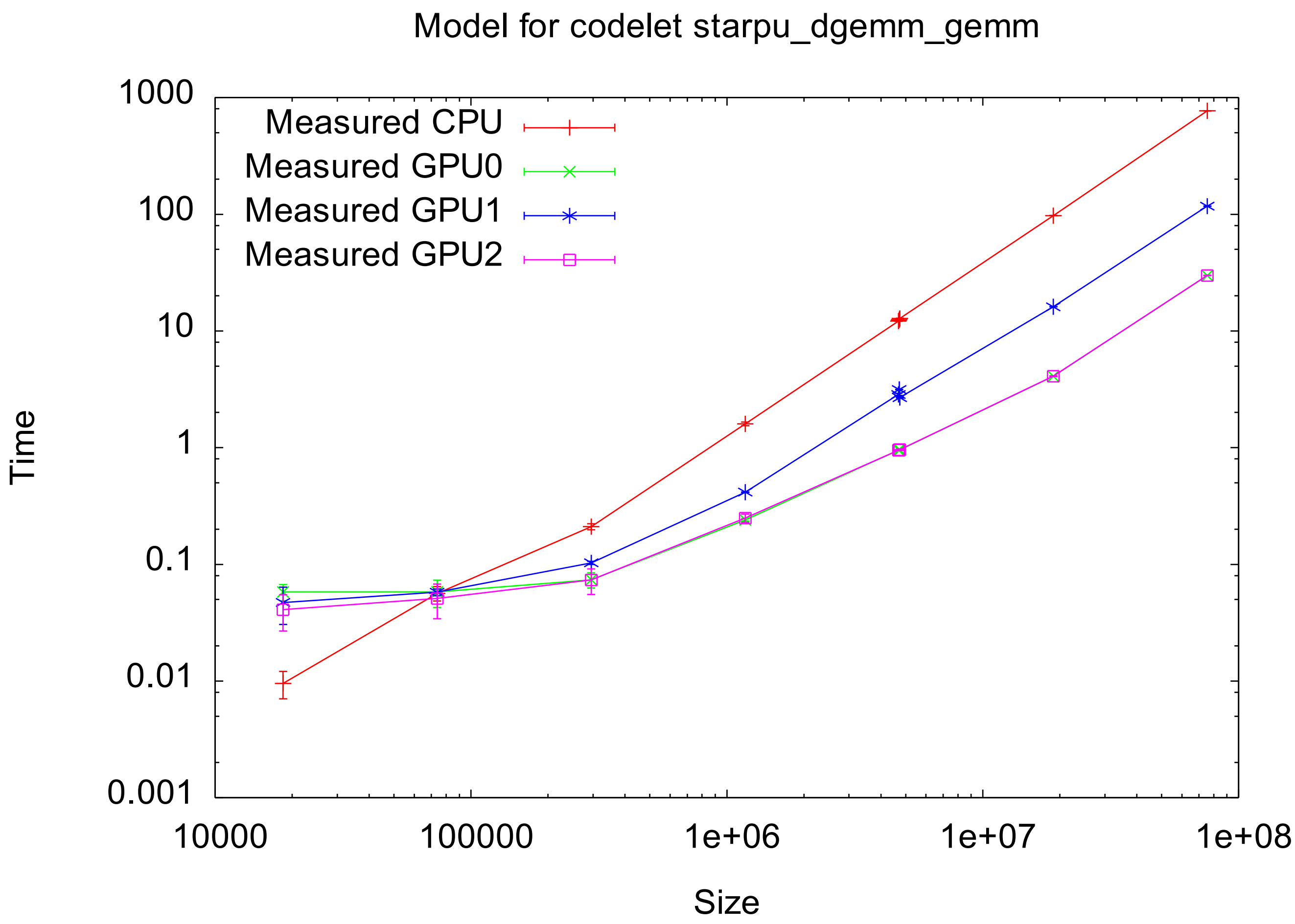
Performance analysis tools

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

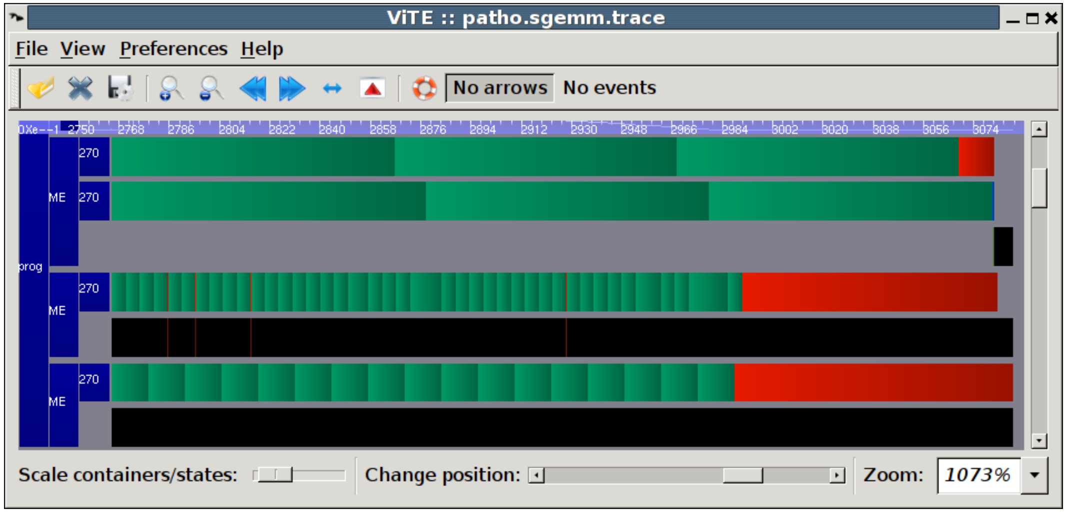
```
$ starpu_perfmodel_plot -s starpu_dgemm_gemm
$ gnuplot starpu_dgemm_gemm.gp
```



Offline performance analysis

Visualize execution traces

- Generate a Pajé trace
  - 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

Extensions

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
  - Handles all needed CPU/GPU transfers
  - Handles task/communications dependencies
  - Overlaps MPI communications, CPU/GPU communications, and CPU/GPU computations

MPI ping-pong example

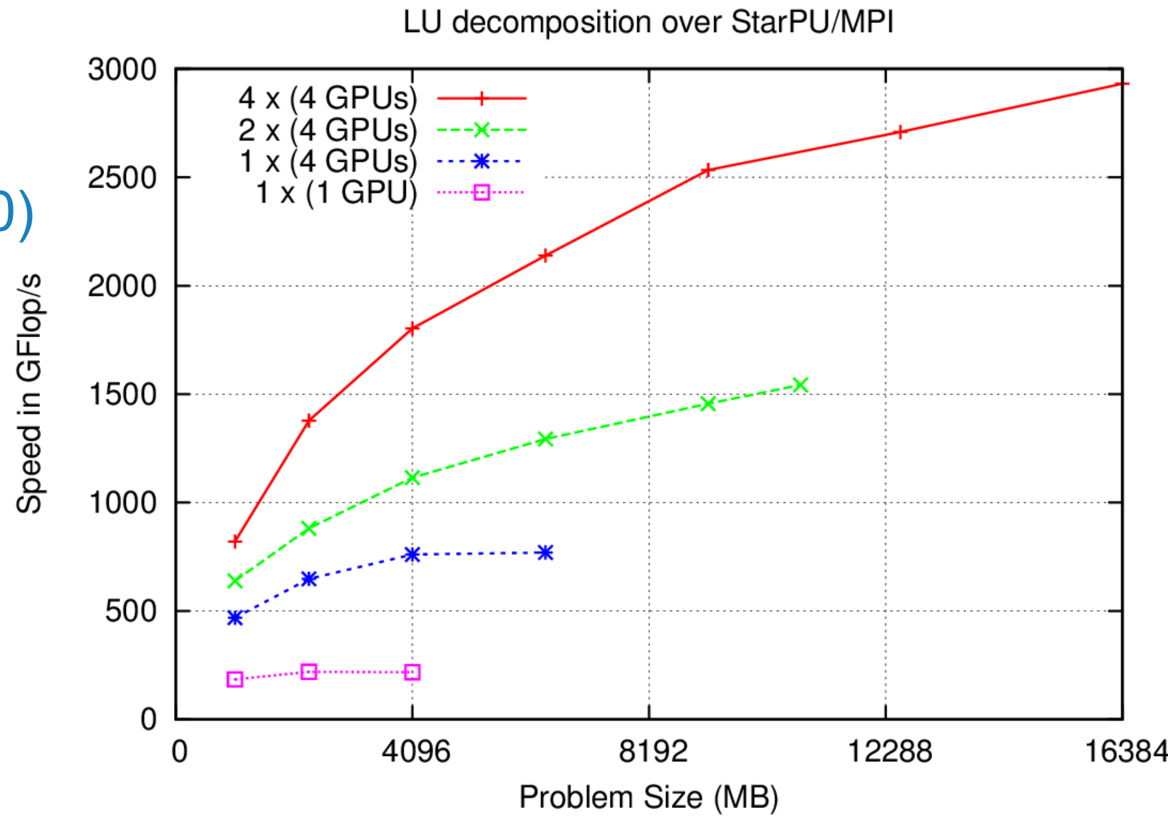
```
for (loop = 0 ; loop < NLOOPS; loop++) {
        if ( !(loop == 0 && rank == 0))
            MPI_Recv(&data, prev_rank, …) ;


        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 !



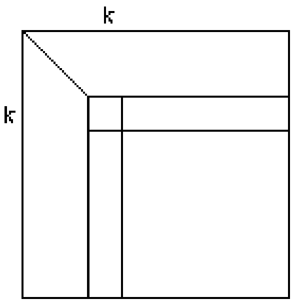
Automatic generation of Send/Recv MPI VSM

- 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

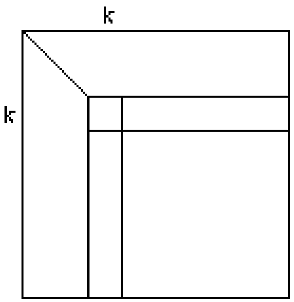
Sequential-looking code !

MPI version of starpu_insert_task MPI VSM – cholesky decomposition




```
for (k = 0 .. tiles-1) {
  starpu_mpi_insert_task(MPI_COMM_WORLD, &potrf,
                         RW, A[k][k], 0);
  for (m = k+1 .. tiles-1)
    starpu_mpi_insert_task(MPI_COMM_WORLD, &trsm,
                           R, A[k][k], RW, A[m][k], 0);
  for (m = k+1 .. tiles-1)
    starpu_mpi_insert_task(MPI_COMM_WORLD, &syrk,
                           R, A[m][k], RW, A[m][m], 0);
  for (m = k+1 .. tiles-1)
    for (n = k+1 .. m-1)
      starpu_mpi_insert_task(MPI_COMM_WORLD, &gemm,
                             R, A[m][k], R, A[n][k], RW, A[m][n], 0);
  }
}
starpu_task_wait_for_all();
```

MPI version of starpu_insert_task
MPI VSM – cholesky decomposition



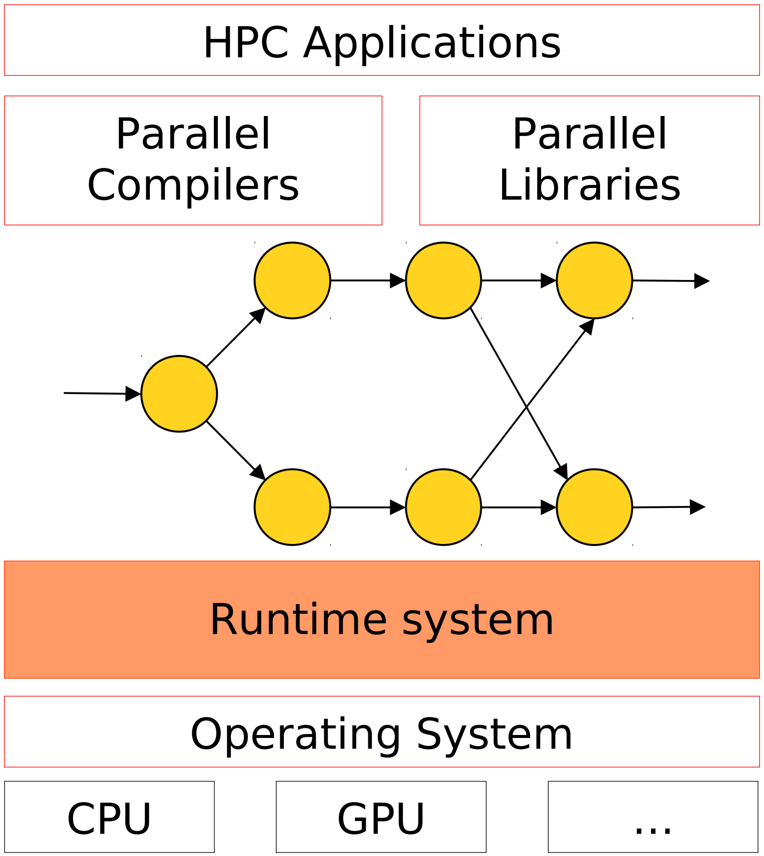

```
for (k = 0 .. tiles-1) {
  POTRF(A[k][k]);

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

  }
}
starpu_task_wait_for_all();
```

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 ?
- MPI load balance
- Out of core
- Application composition
  - Collaborating scheduling context

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 ?
- Application composition
  - Collaborating scheduling context

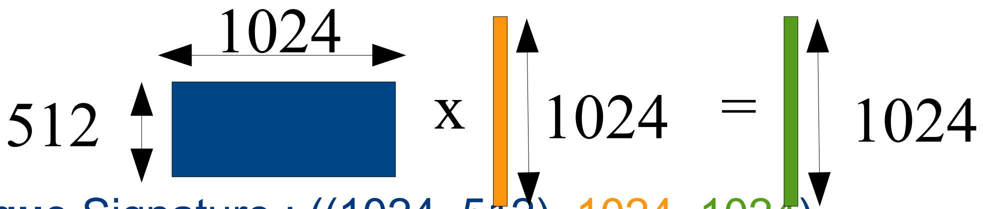
http://starpu.gforge.inria.fr/

Thanks for your attention !

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)

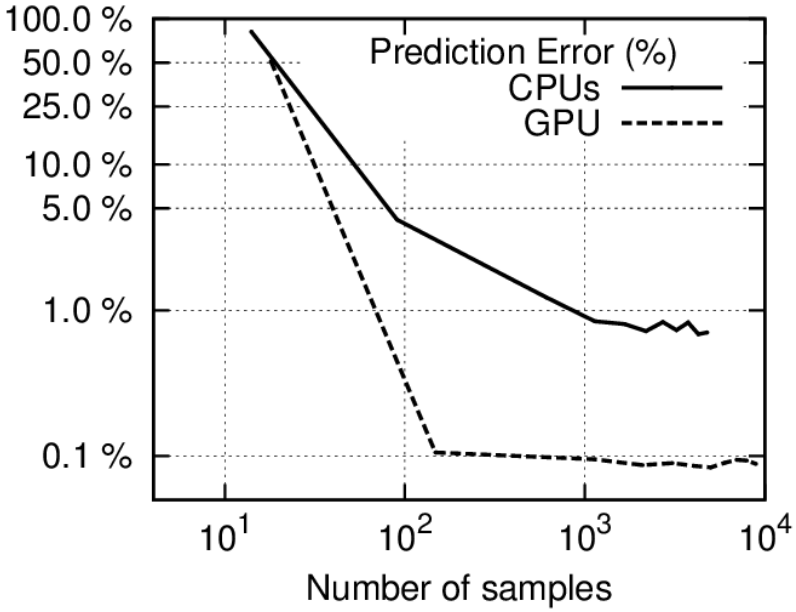
  - Data
    - Signature(Di) = CRC(p1, p2, … , pk)
  - Task ~ Series of data
    - Signature(D1, ..., Dn) = CRC(sign(D1), ..., sign(Dn))

- 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