StarPU a runtime system for Scheduling Tasks, *or* How to get portable performance on accelerator-based platforms without the agonizing pain

Cédric Augonnet Samuel Thibault Raymond Namyst

INRIA Bordeaux, LaBRI, University of Bordeaux

NVIDIA GPU Technology Conference – San Jose (USA) – September 2010

Once upon a time in computer architecture ...

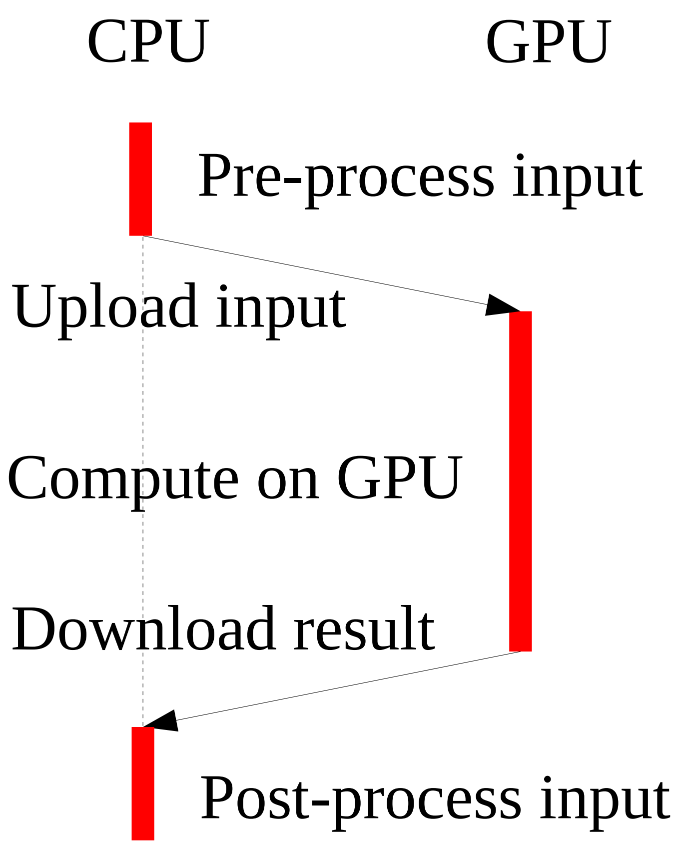
Introduction
Programming Accelerator-based machines

- Prehistory (<2007)
  - SMP machines (1960s !)
  - NUMA architectures (1970s)
  - Vector machines (1980s-90s)
  - Multicore chips (2000s)

- GPGPU for masses ? (from 2007)
  - CPUs are now deprecated ?
  - Rewrite all codes for accelerators
  - Pure offloading model

Introduction
Programming Accelerator-based machines

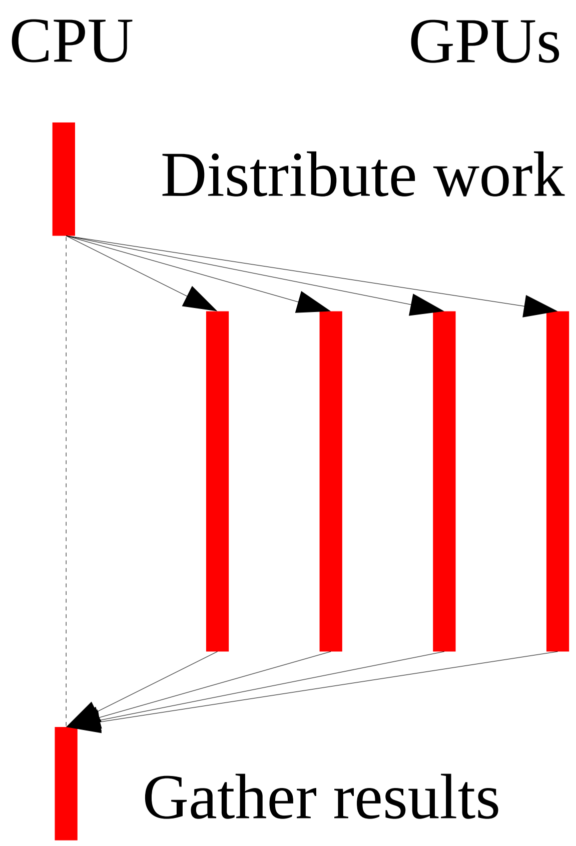
- Pure offloading model
    - CUDA 1.x (2007 – 2008)
        - Synchronous cards
    - Ignore CPUs
    - Concentrate on efficient kernels
        - Complex memory accesses
        - CUDA heros (eg. V.Volkov)
    - Port standard libraries to CUDA
        - CUBLAS
        - CUFFT
        - GPUCV...



Introduction

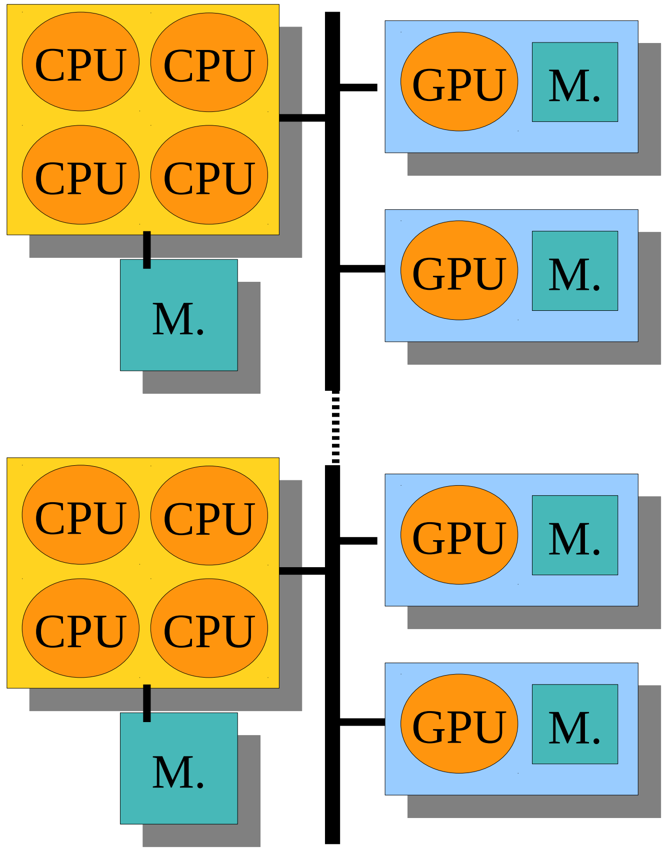
Programming Accelerator-based machines

- Multi-GPU era
  - CUDA 2.x (2008 – 2009)
    - Asynchronous transfers
    - S1070 servers
  - Still Ignore CPUs (in general)
  - Suitable for regular applications
    - Massively parallel problems
    - Use previously written kernels
  - New problems
    - Parallel programming for real
    - PCI bus = bottleneck
    - Pre/Post-processing is costly



Introduction
Programming Accelerator-based machines

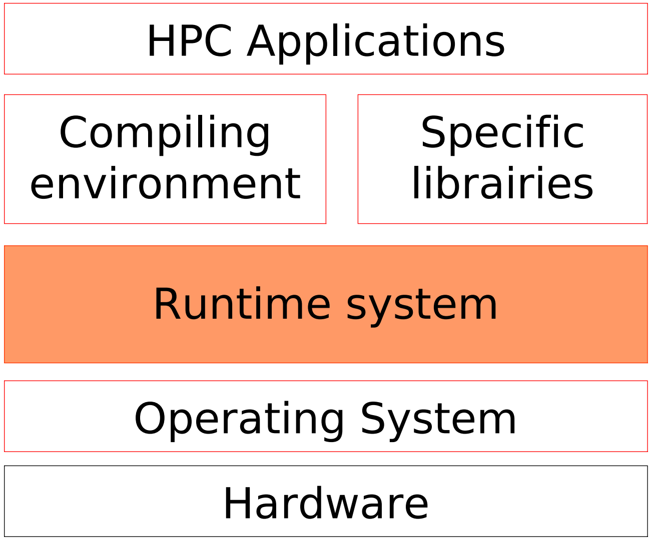
- GPU computing era
  - CUDA 3.x (2009 – ?)
    - End of GPGPU
    - Hybrid machines
  - Tightly coupled CPUs and GPUs
    - Take advantage of all ressources
    - MUCH more complicated
  - Load balancing
    - Who does what ?
    - Heterogeneous capabilities
  - Data management
    - Numerous data transfers
    - Fully asynchronous model



Introduction

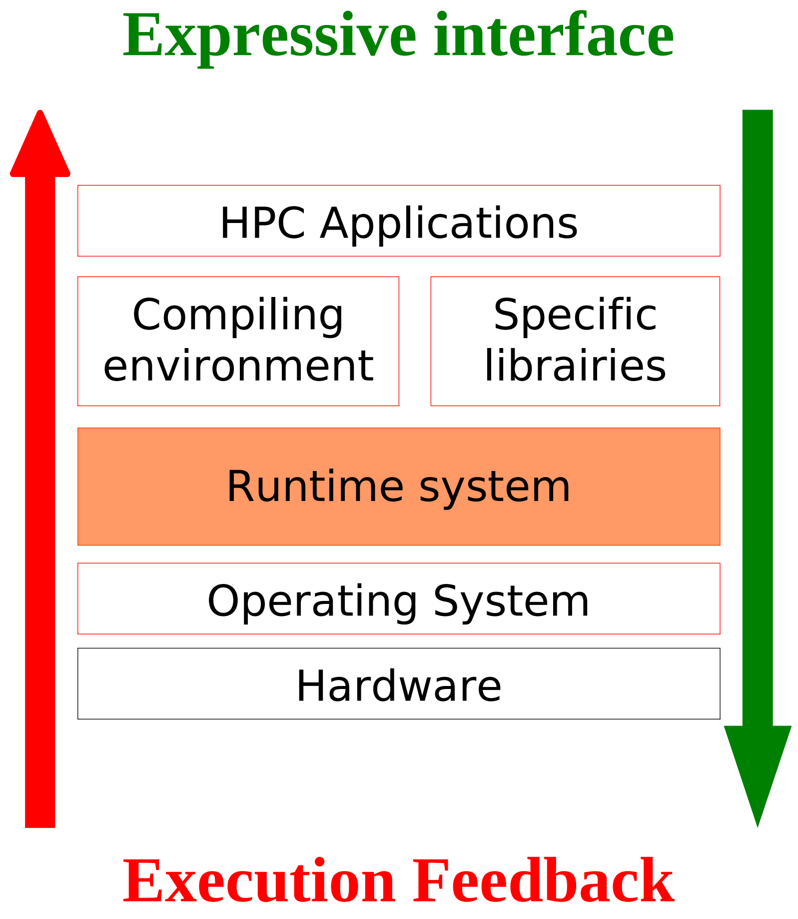
Challenging issues at all stages

- 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

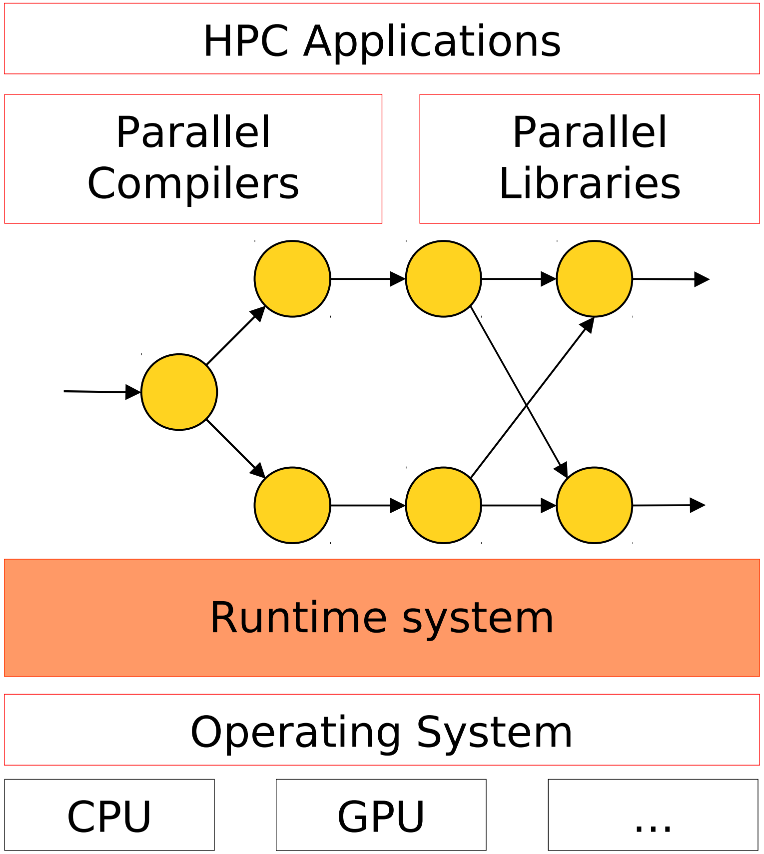
- The StarPU runtime system
- Task Scheduling
  - Load balancing
  - Improving data locality
- Evaluation on dense linear algebra algorithms
  - Synthetic “LU” decomposition
  - Mixing PLASMA and MAGMA (Cholesky & QR)
- Scheduling parallel tasks
- Adding support for MPI in StarPU

The StarPU runtime system

The StarPU runtime system

The need for runtime systems

- “do dynamically what can’t be done statically anymore”
- Library that provides
  - Task scheduling
  - Memory management
- Compilers and libraries generate (graphs of) parallel tasks
  - Additional information is welcome!

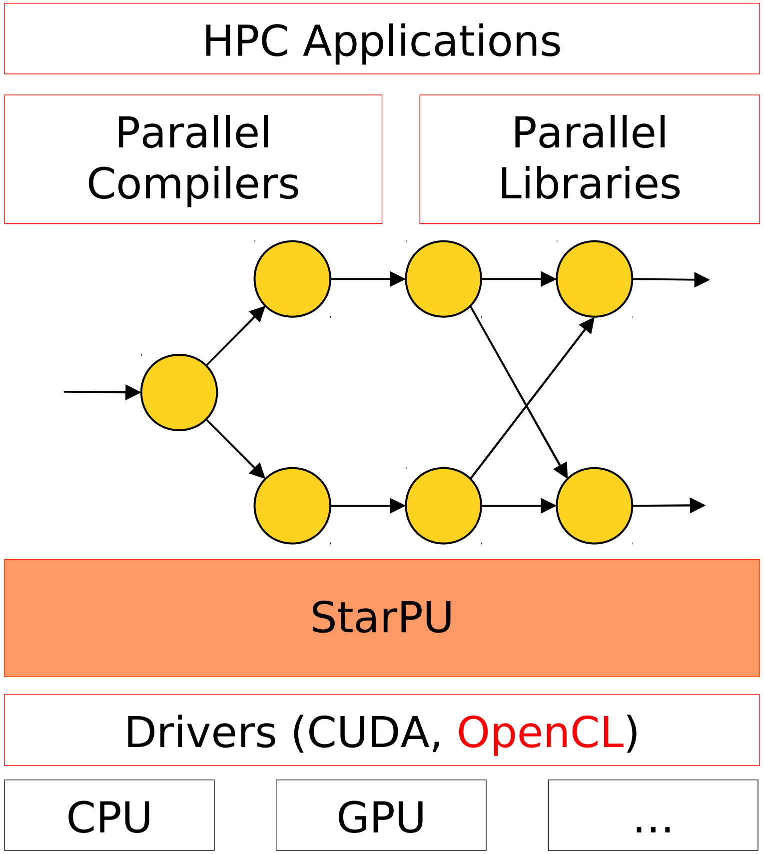


The StarPU runtime system

Data management library

- 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

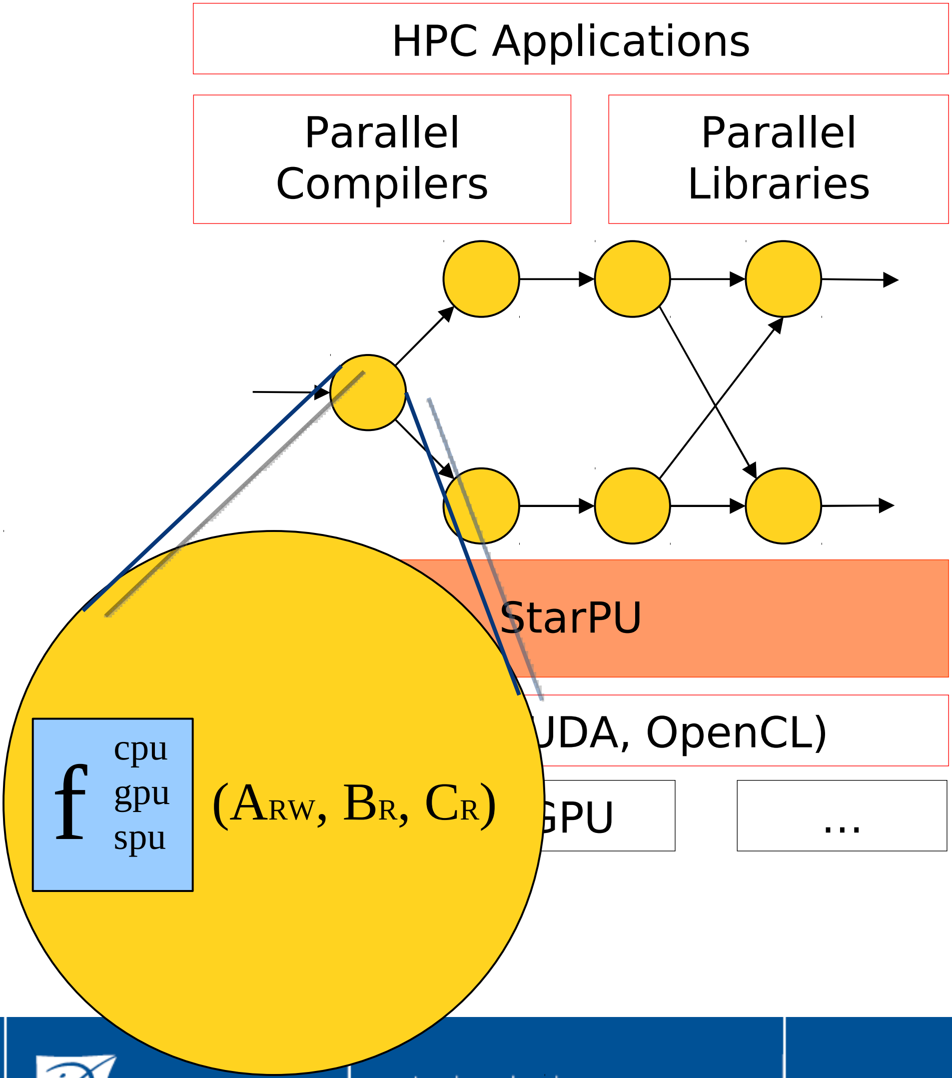


The StarPU runtime system

Task scheduling

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

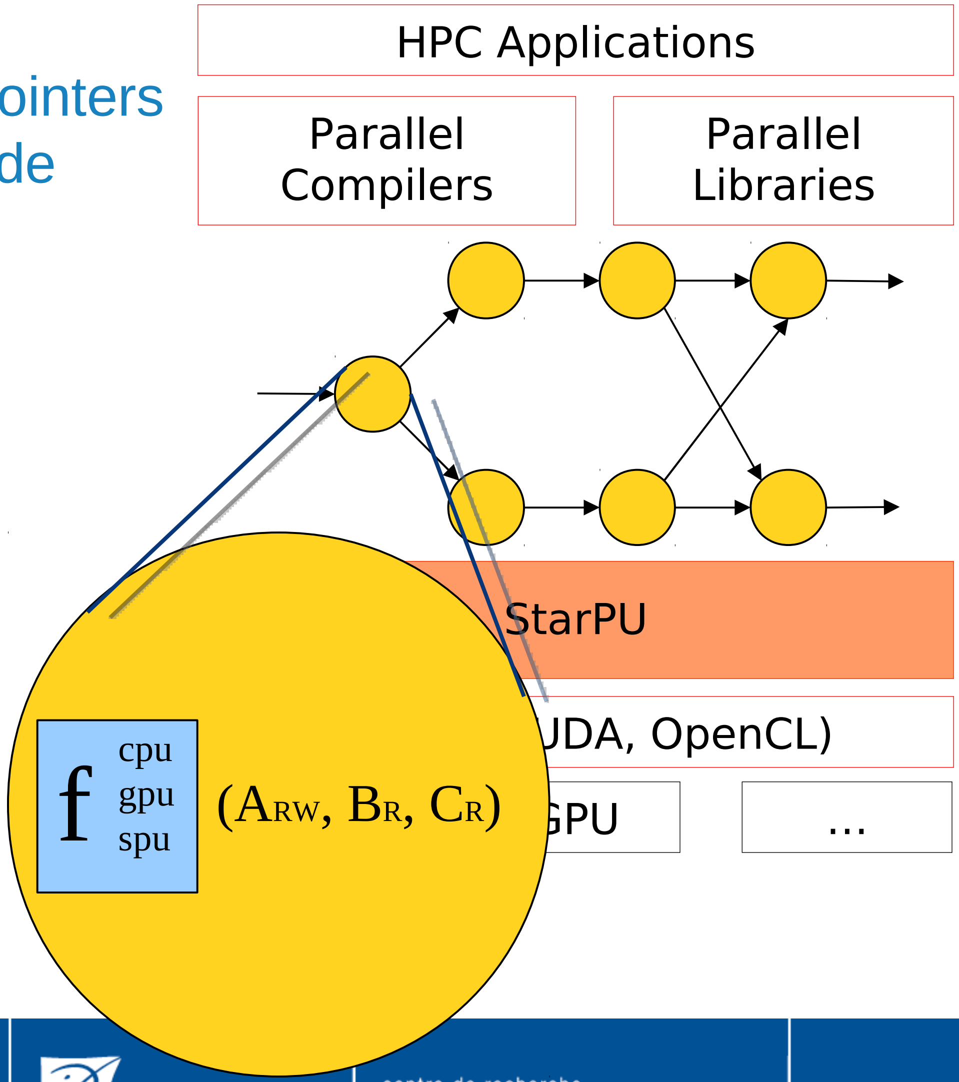
- StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins



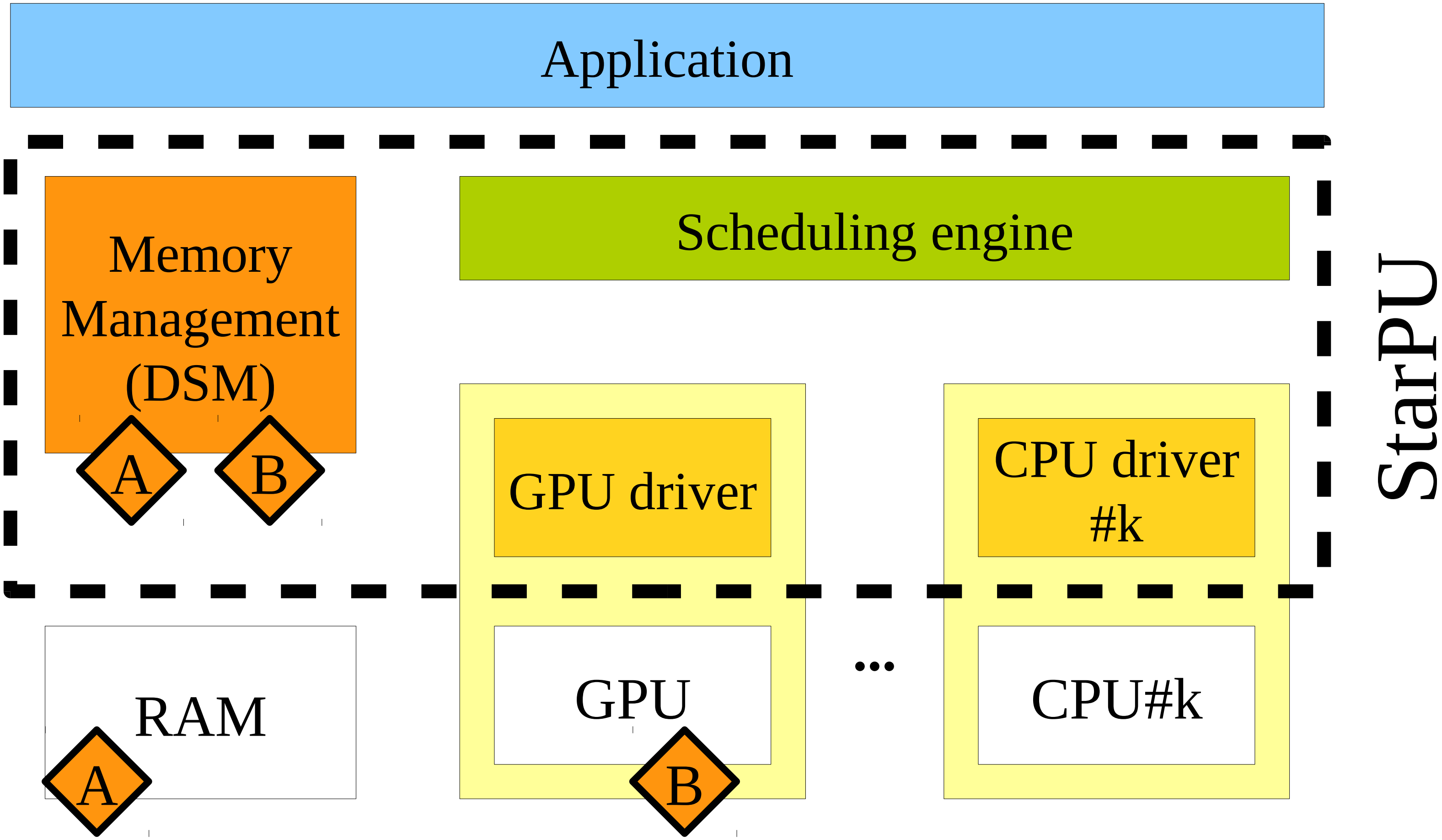
The StarPU runtime system

Task scheduling

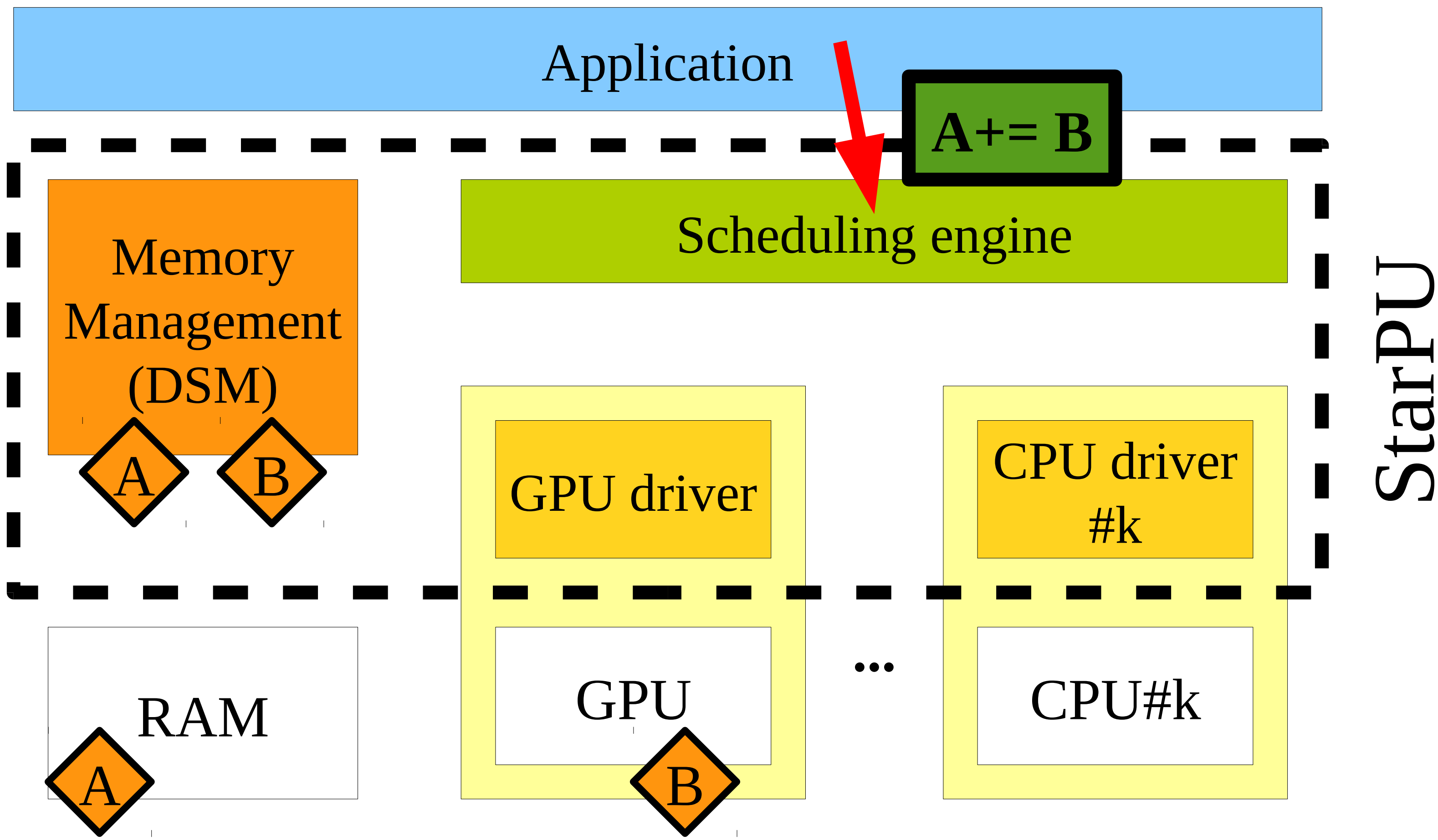
- Who generates the code ?
  - StarPU Task = ~function pointers
  - StarPU don't generates code
- Programming heros ?
- Libraries era
  - PLASMA + MAGMA
  - FFTW + CUFFT...
- Rely on compilers
  - PGI accelerators
  - CAPS HMPP...



The StarPU runtime system
Execution model



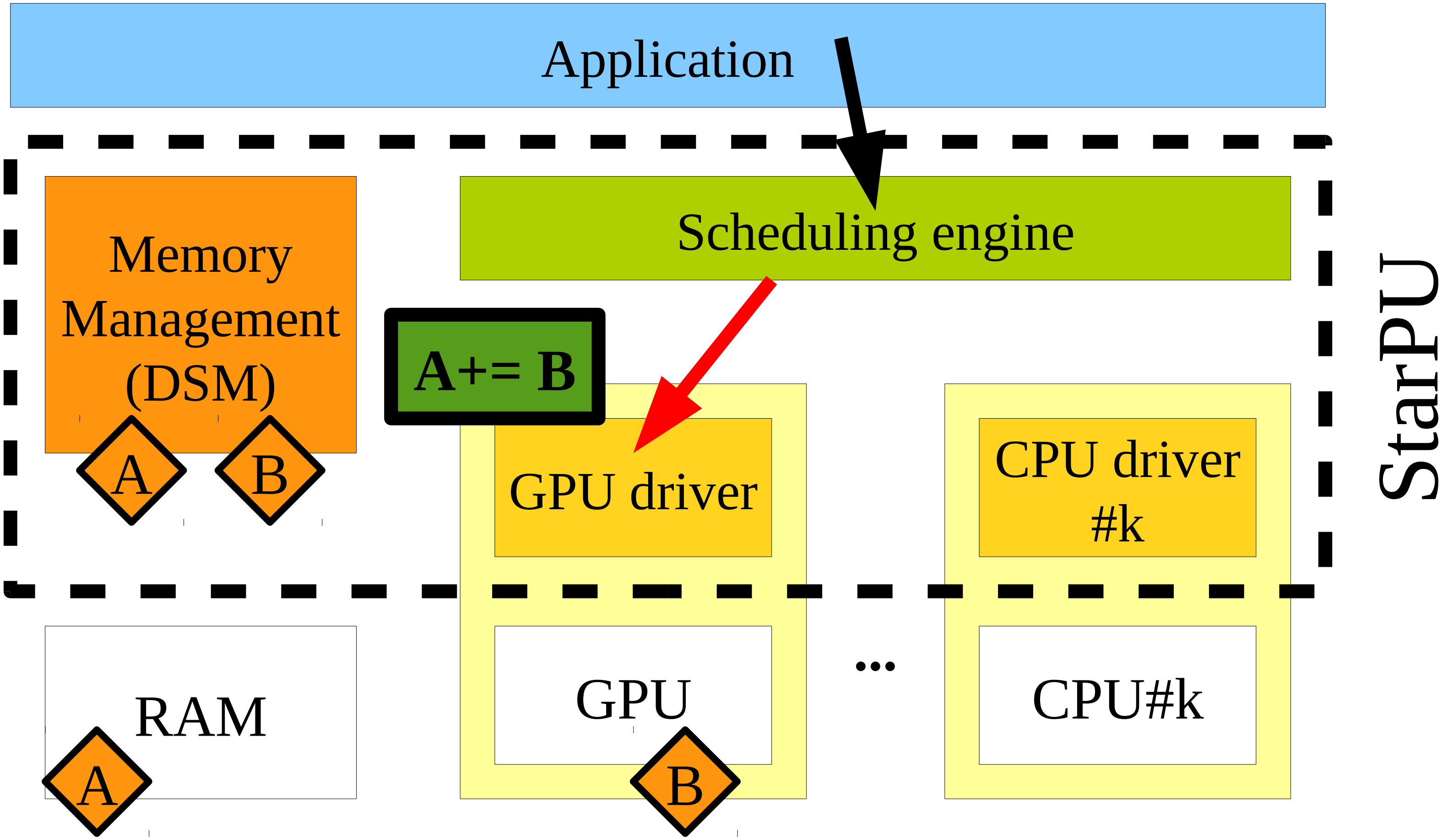
The StarPU runtime system
Execution model



Submit task « A += B »

The StarPU runtime system

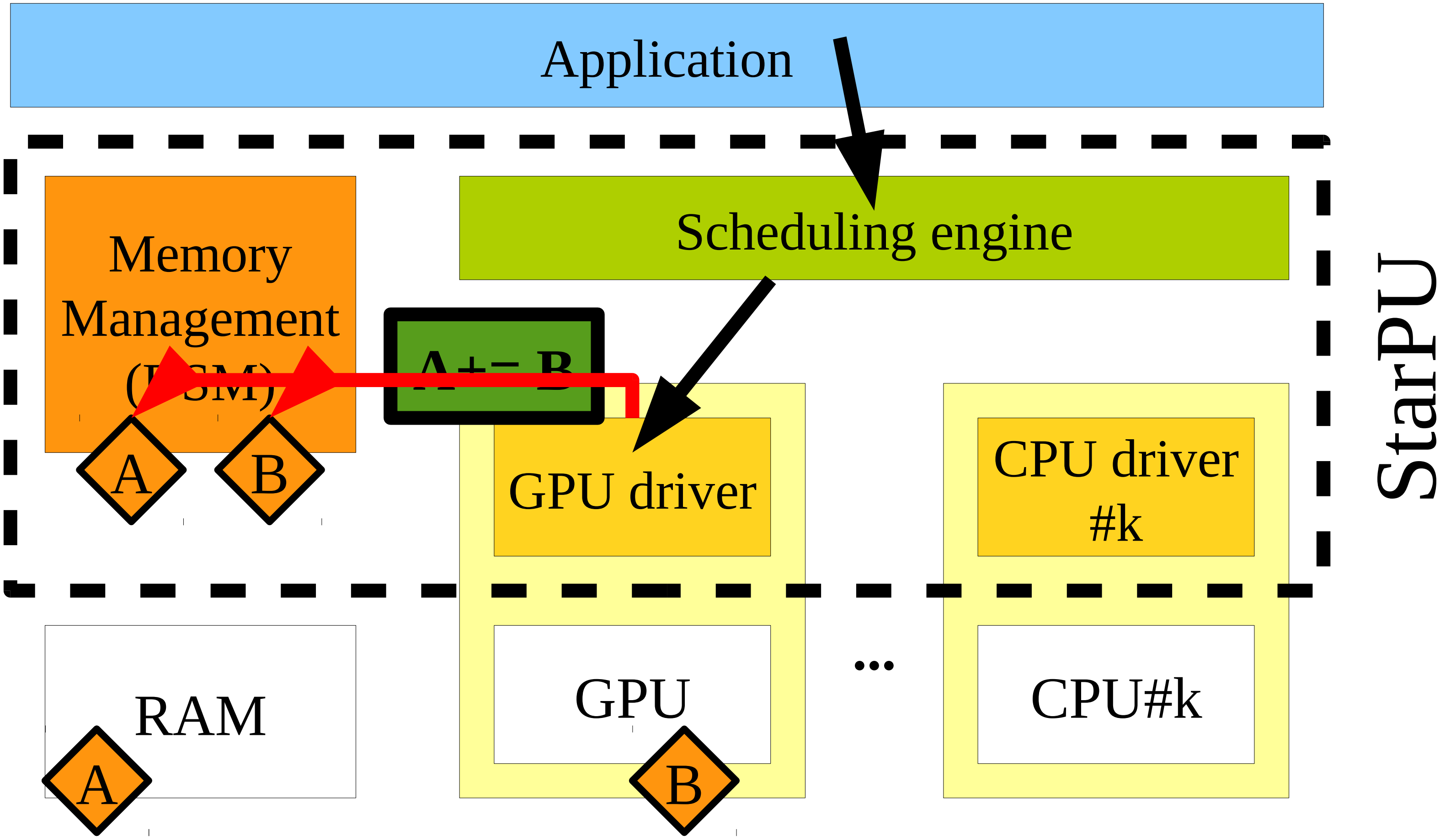
Execution model



Schedule task

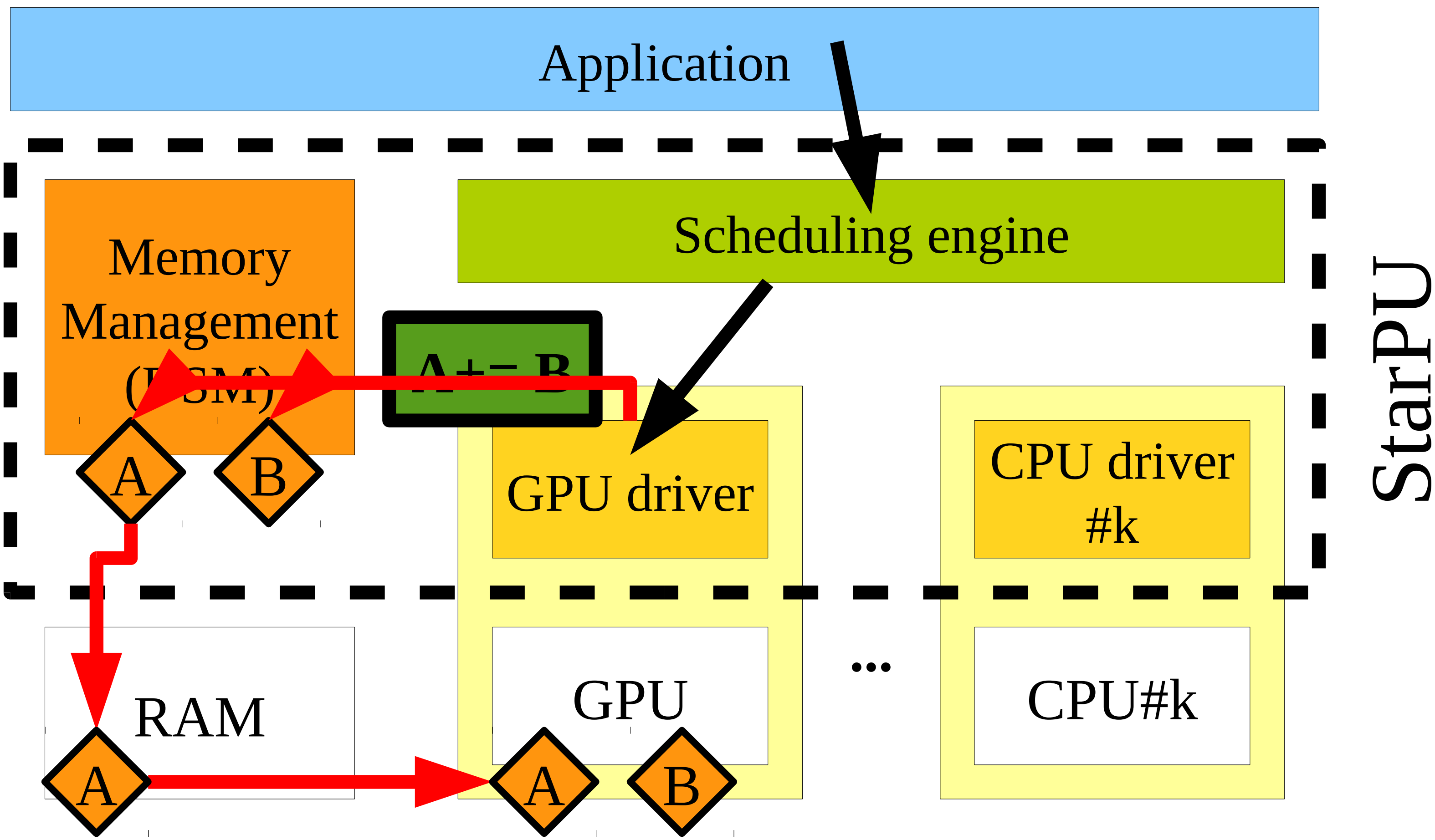
The StarPU runtime system

Execution model



Fetch data

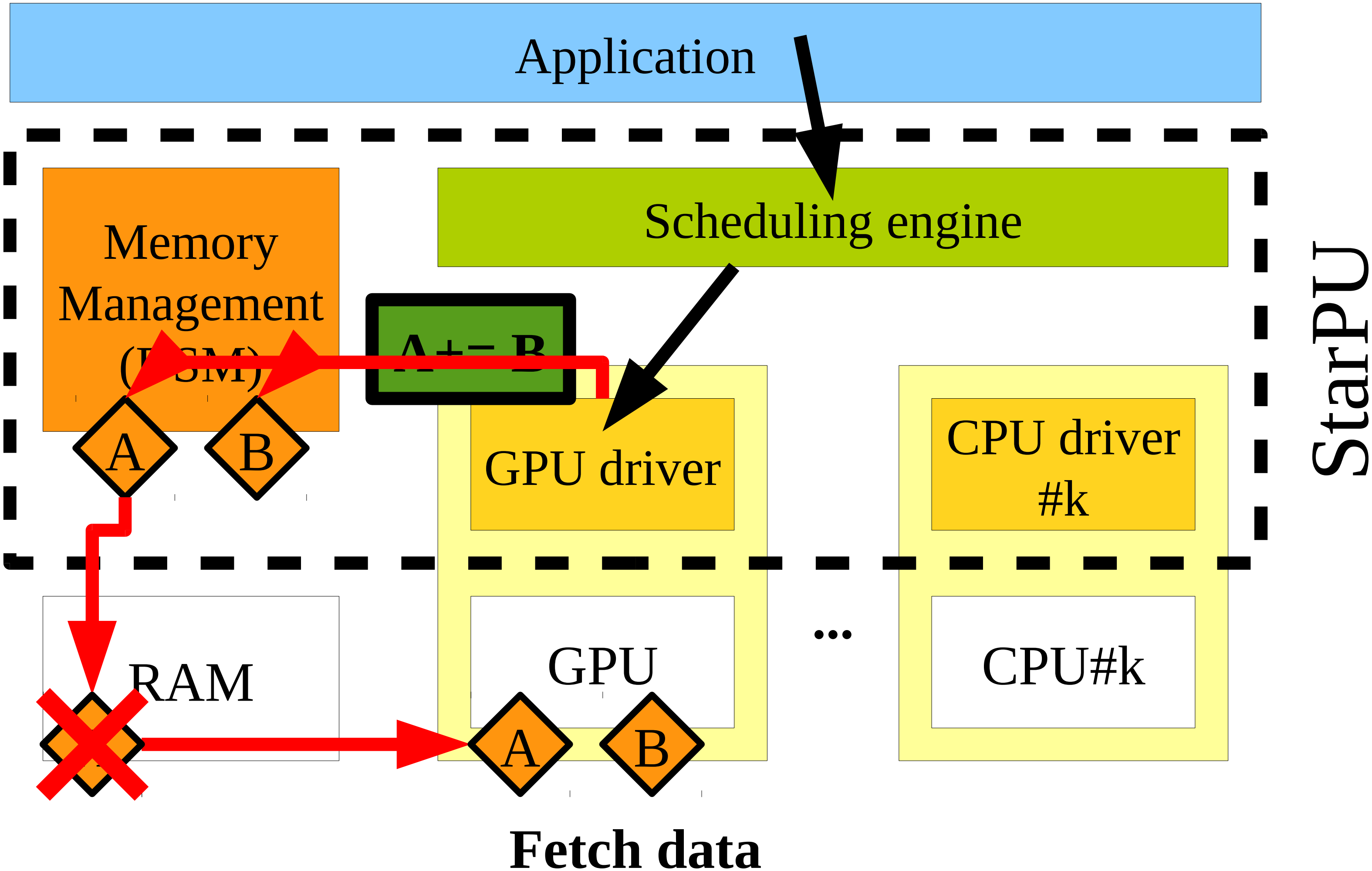
The StarPU runtime system
Execution model



Fetch data

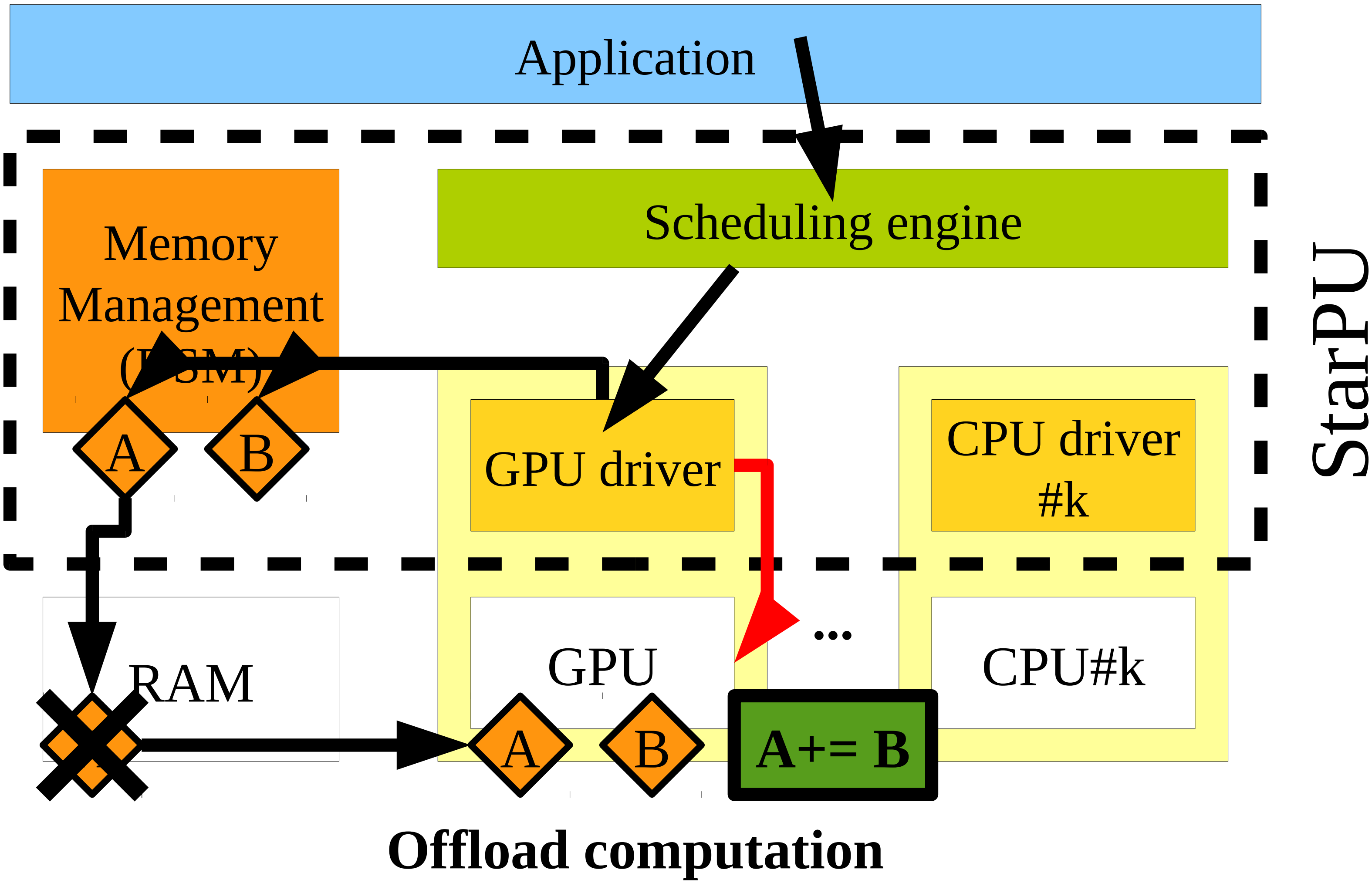
The StarPU runtime system

Execution model

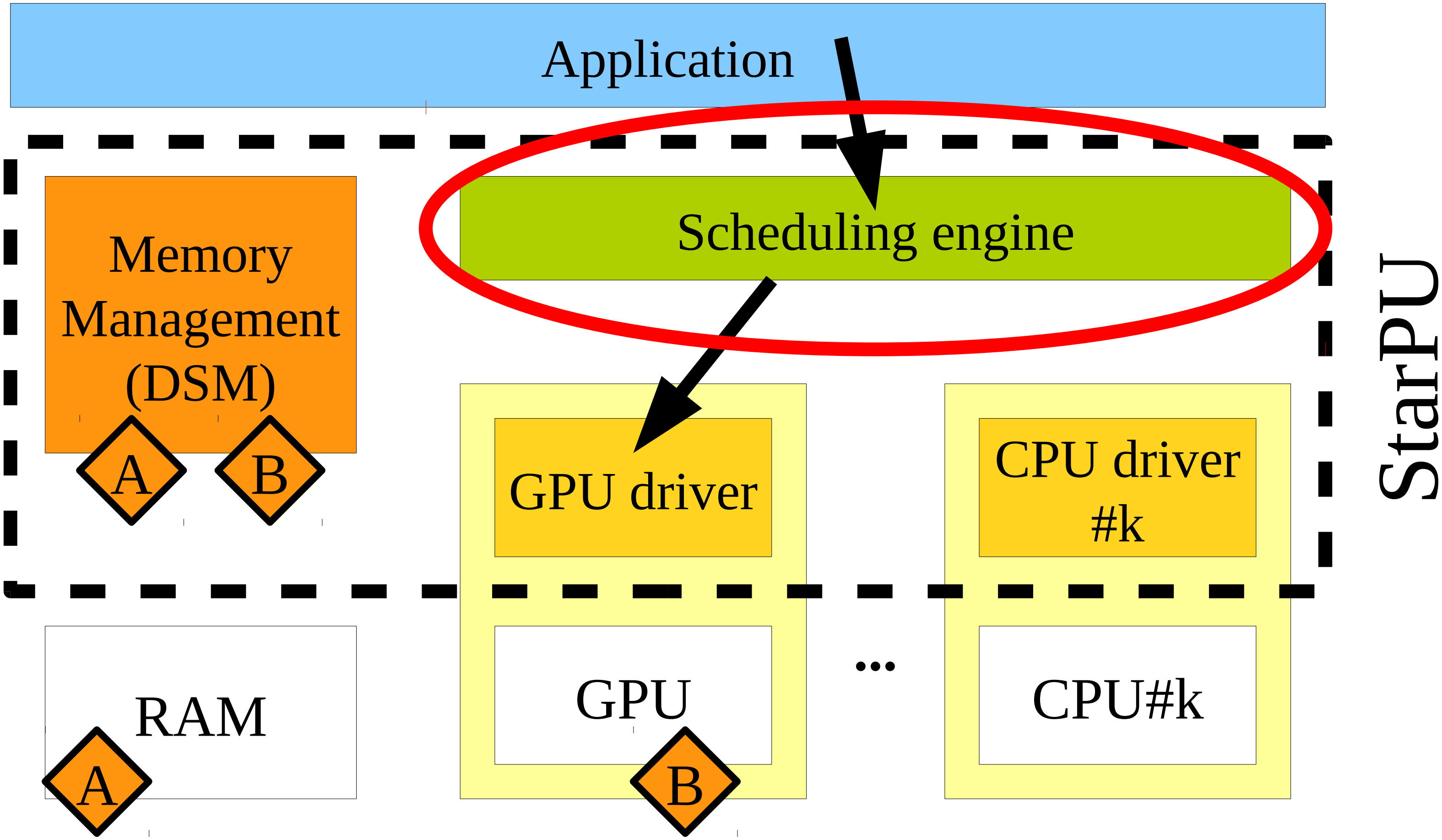


The StarPU runtime system

Execution model



The StarPU runtime system
Execution model



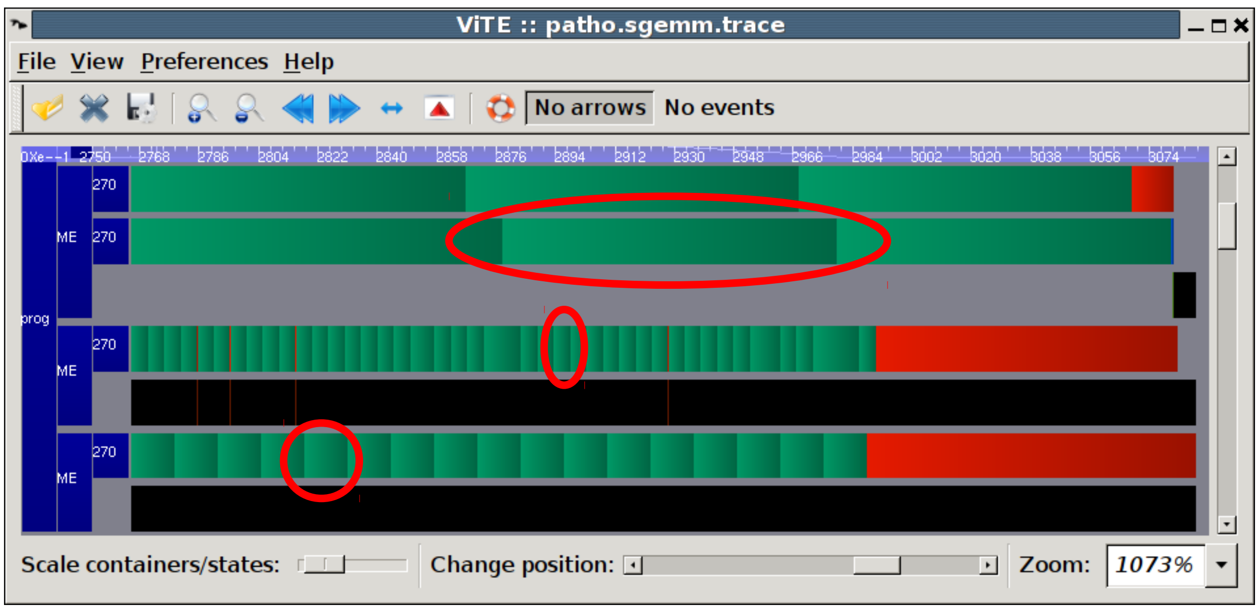
Task Scheduling

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



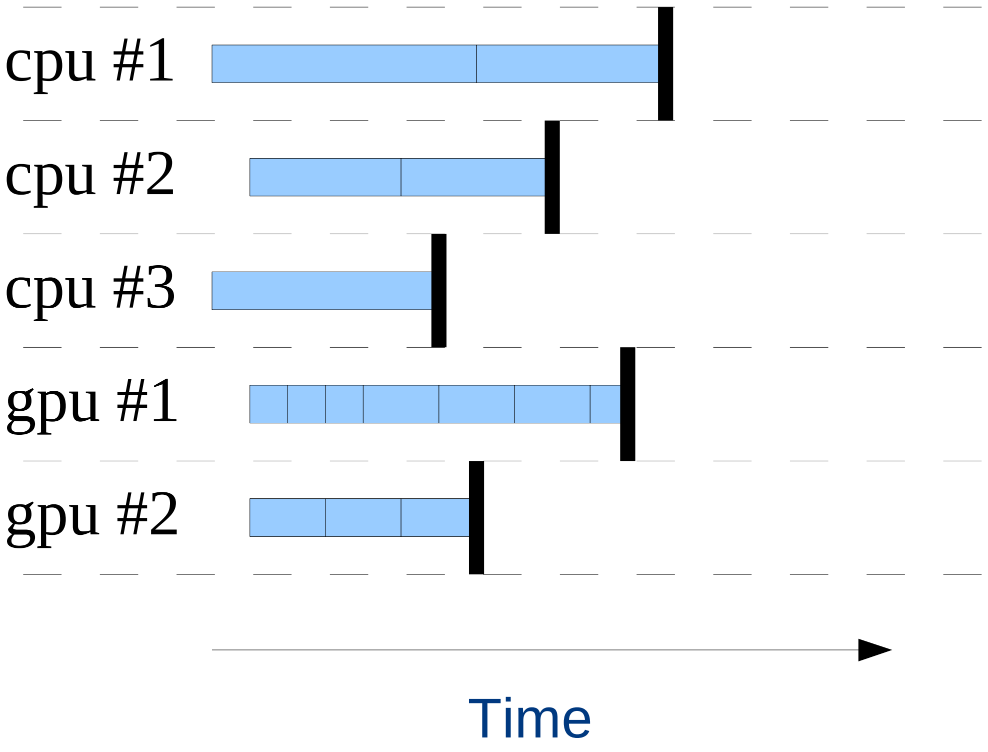
2 Xeon cores

Quadro FX5800

Quadro FX4600

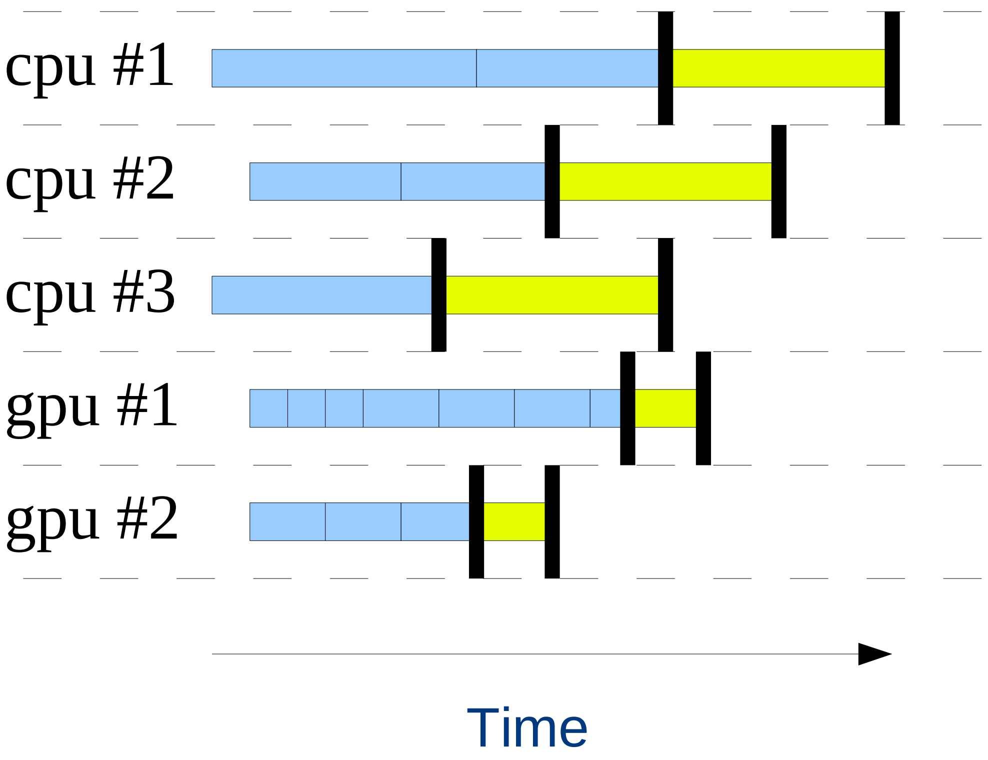
Predicting task duration
Load balancing

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



Predicting task duration
Load balancing

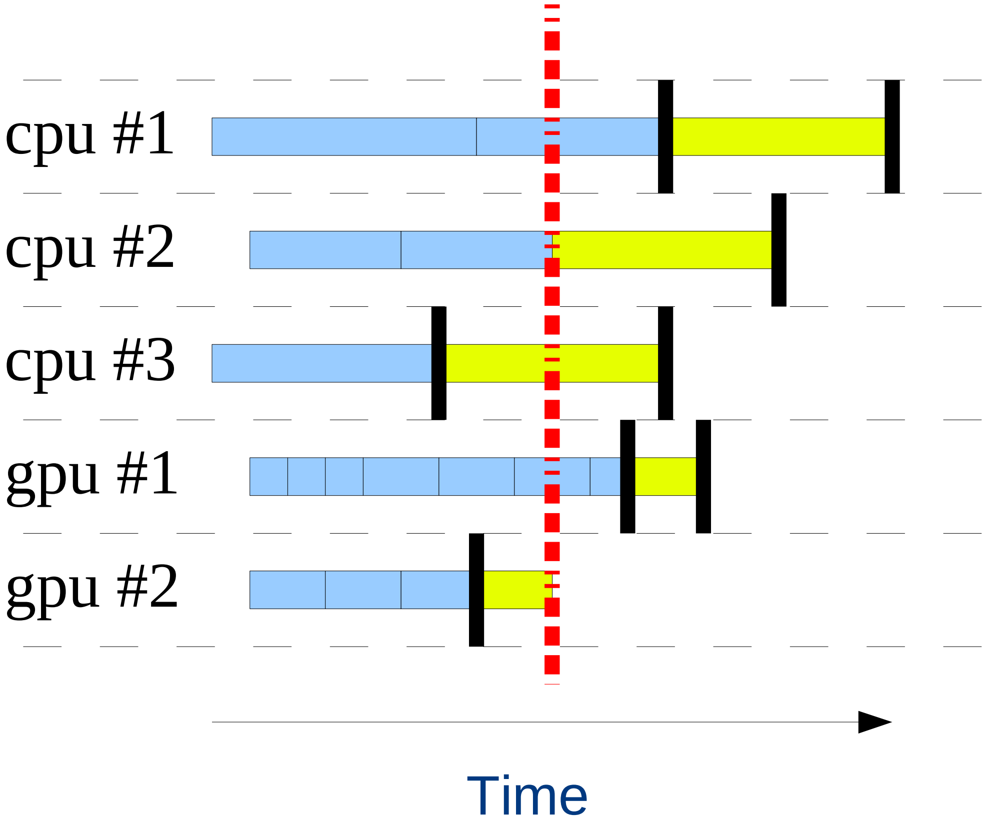
- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model



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

Predicting task duration
Load balancing

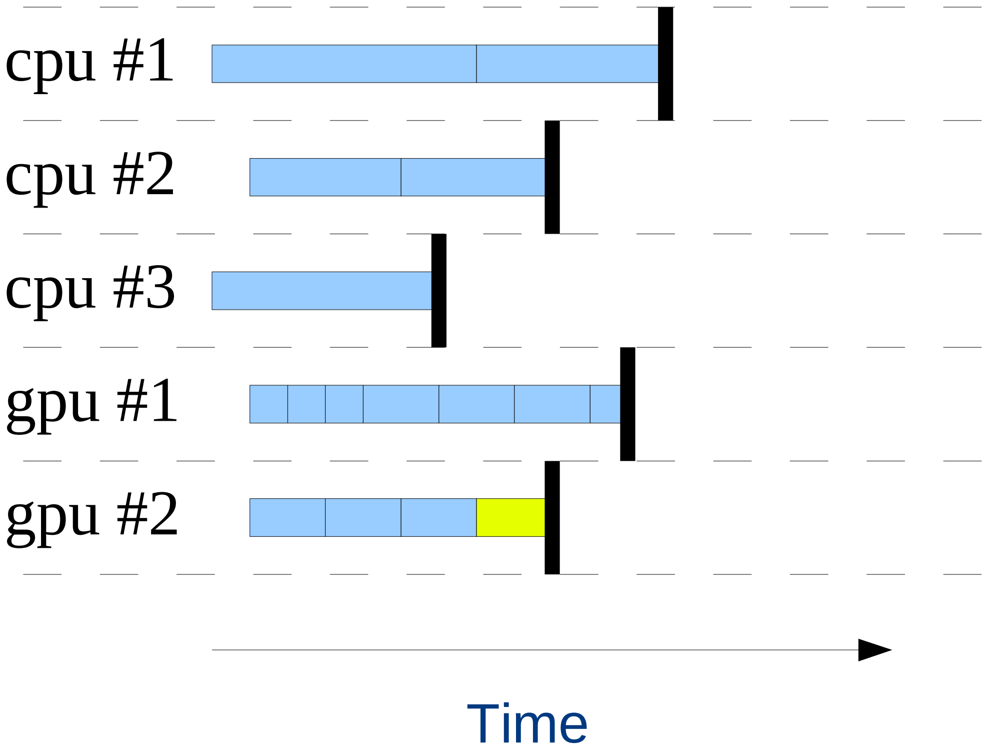
- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model



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

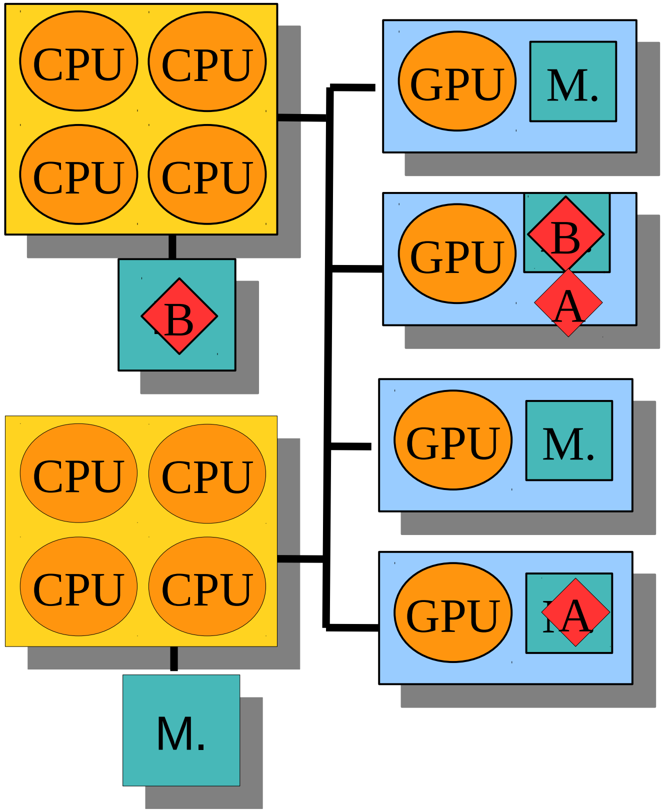
Predicting task duration
Load balancing

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



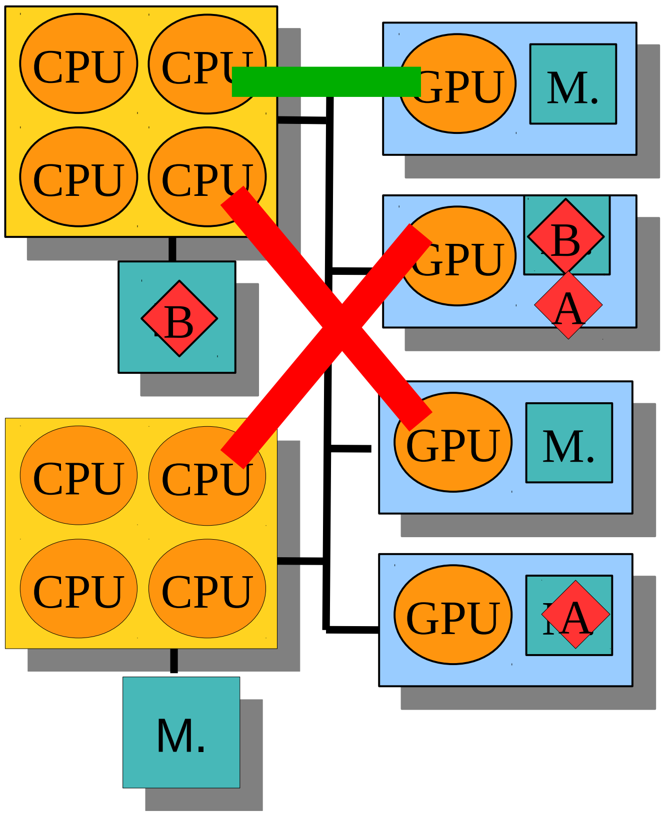
Predicting data transfer overhead
Motivations

- 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-goig data movements



Predicting data transfer overhead
Offline bus benchmarking

- Offline bus benchmarking
  - When StarPU is launched for the first time
  - Measure bandwidth and latency
    - Stored as files
  - Loaded when StarPU is initialized



- Detect CPU/GPU affinity
  - Control a GPU from the closest CPU
  - Significant impact on bus usage

- Straightforward cost prediction
  - Latency + size * bandwidth
  - Could be improved in many ways

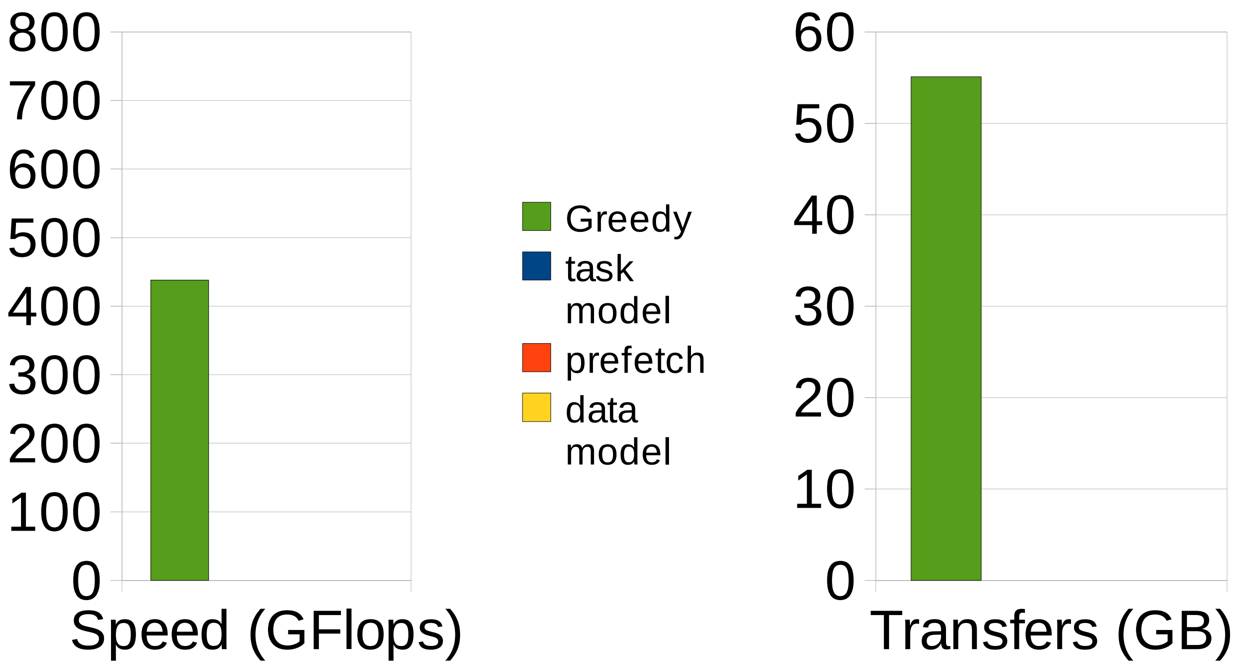
Impact of scheduling policy on a synthetic LU decomposition

(without pivoting !)

Scheduling in a hybrid environment

Performance models

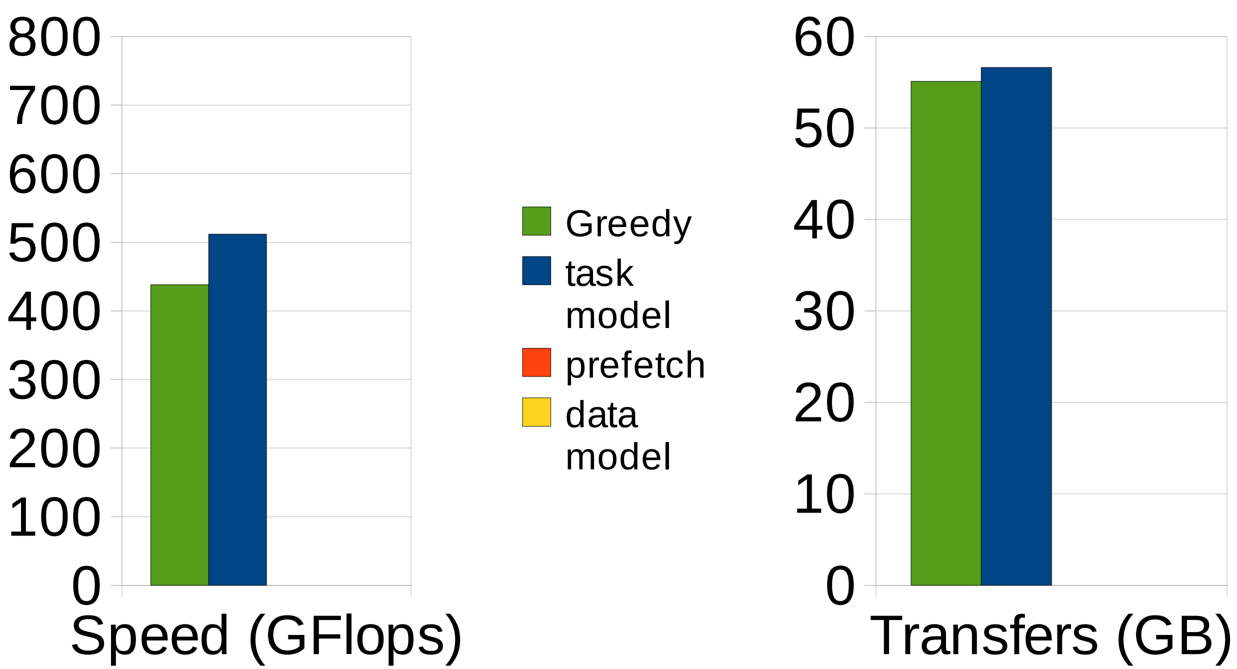
- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)



Scheduling in a hybrid environment

Performance models

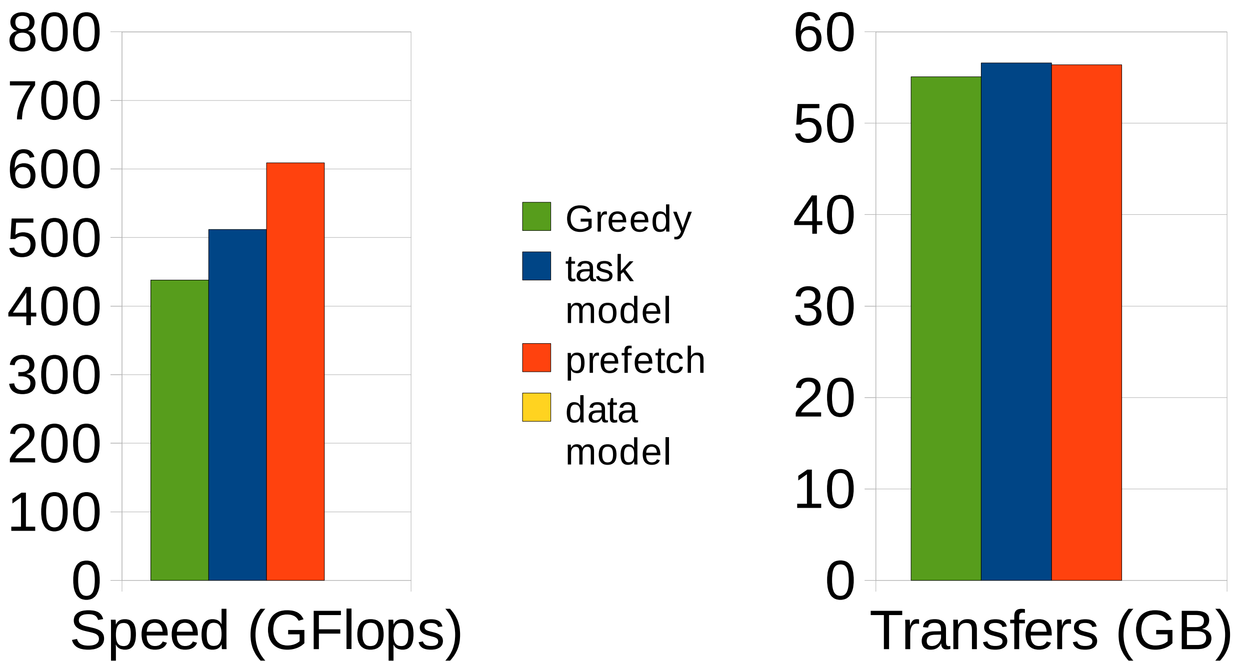
- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)



Scheduling in a hybrid environment

Performance models

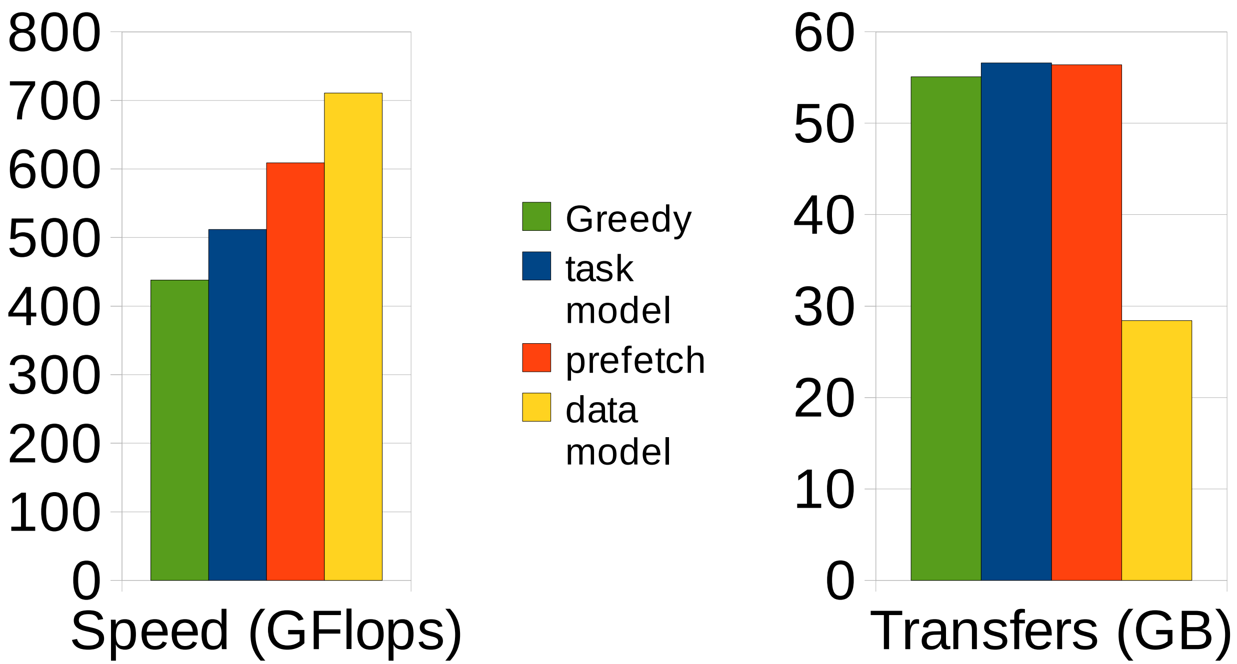
- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)



Scheduling in a hybrid environment

Performance models

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



Mixing PLASMA and MAGMA with StarPU

(in collaboration with UTK)

Cholesky & QR decompositions

Mixing PLASMA and MAGMA with StarPU

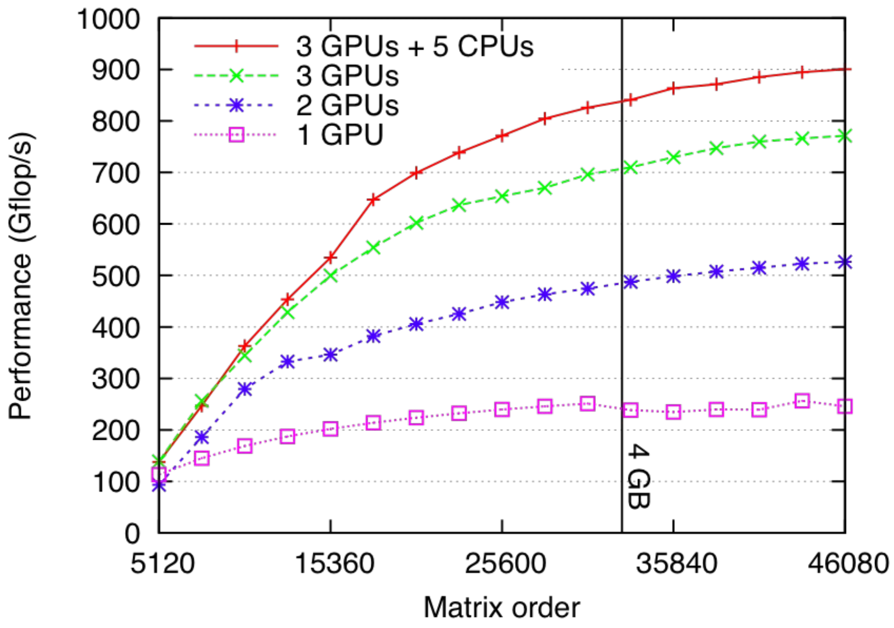
- State of the art algorithms
  - PLASMA (Multicore CPUs)
    - Dynamically scheduled with Quark
  - MAGMA (Multiple GPUs)
    - Hand-coded data transfers
    - Static task mapping

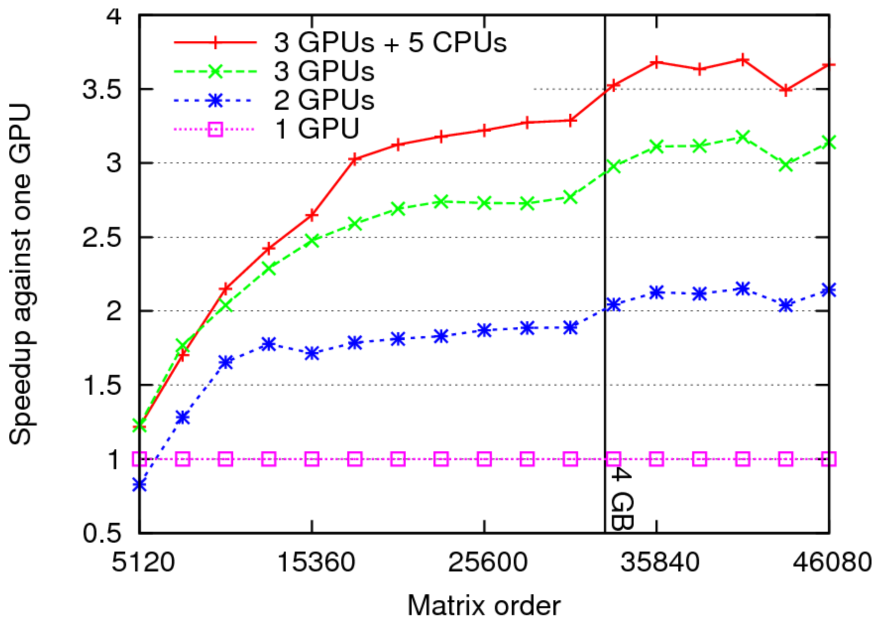
- General SPLAGMA design
  - Use PLASMA algorithm with « magnum tiles »
  - PLASMA kernels on CPUs, MAGMA kernels on GPUs
  - Bypass the QUARK scheduler

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

Mixing PLASMA and MAGMA with StarPU

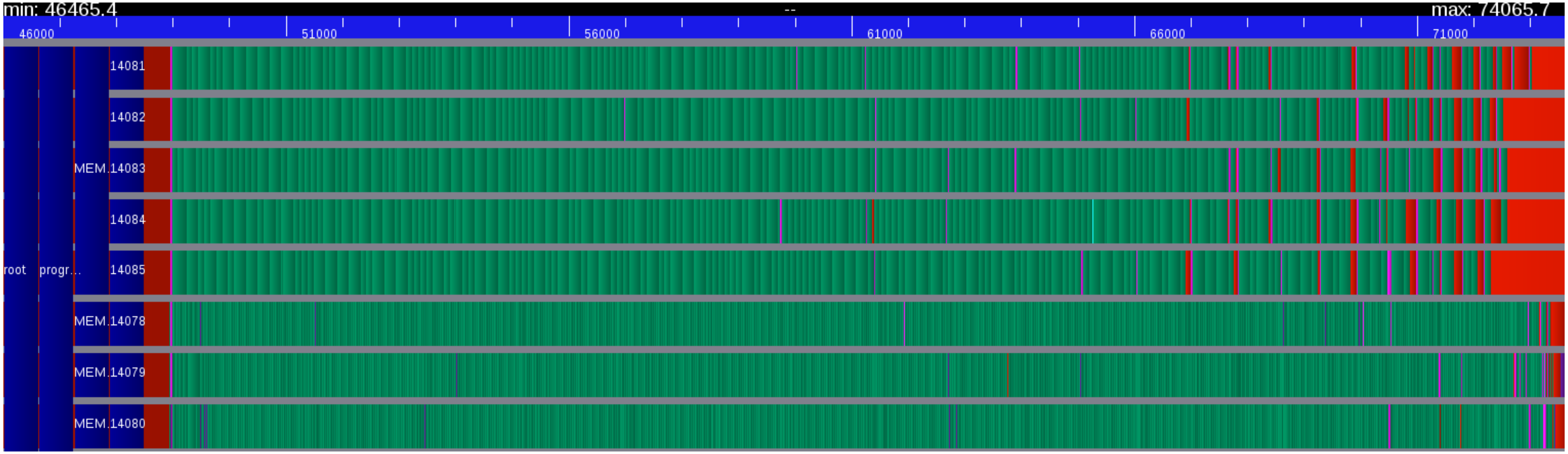
- Cholesky decomposition
  - 5 CPUs (Nehalem) + 3 GPUs (FX5800)
  - Efficiency > 100%





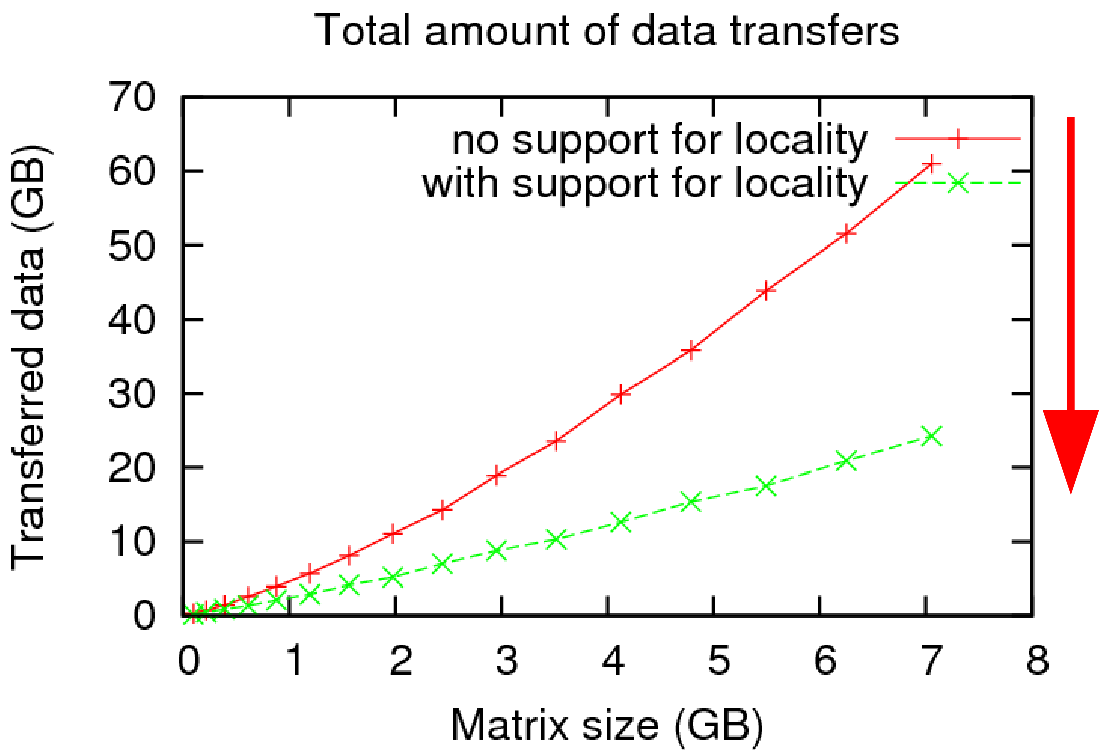
Mixing PLASMA and MAGMA with StarPU

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



Mixing PLASMA and MAGMA with StarPU

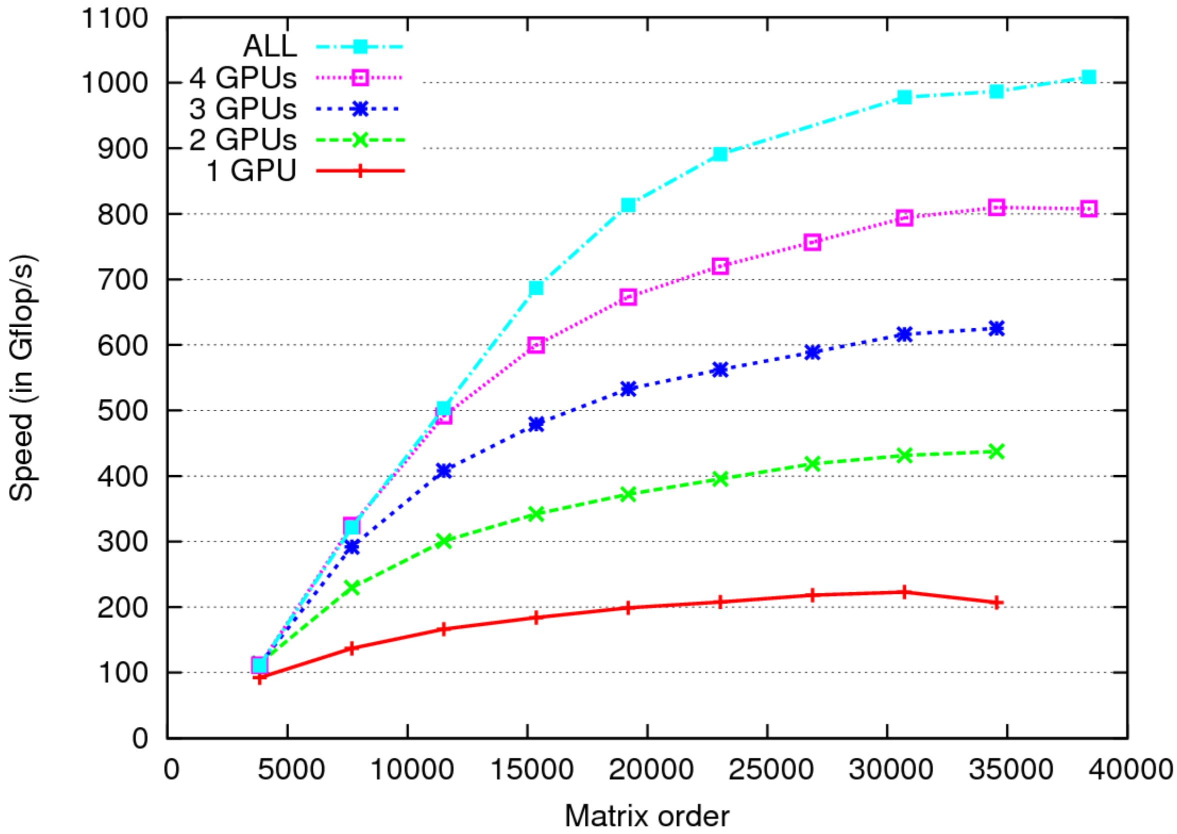
- Memory transfers during Cholesky decomposition



~2.5x less transfers

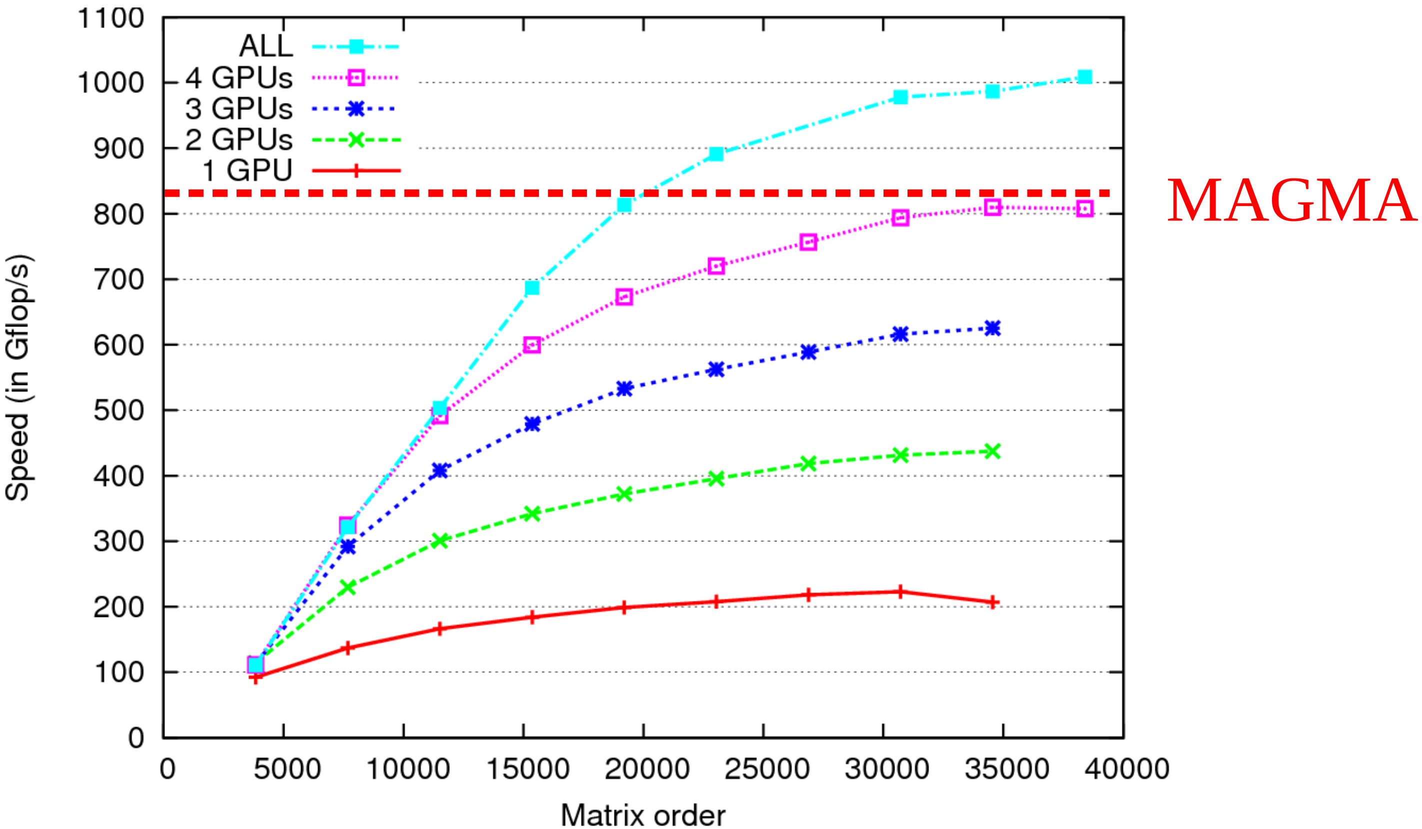
Mixing PLASMA and MAGMA with StarPU

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



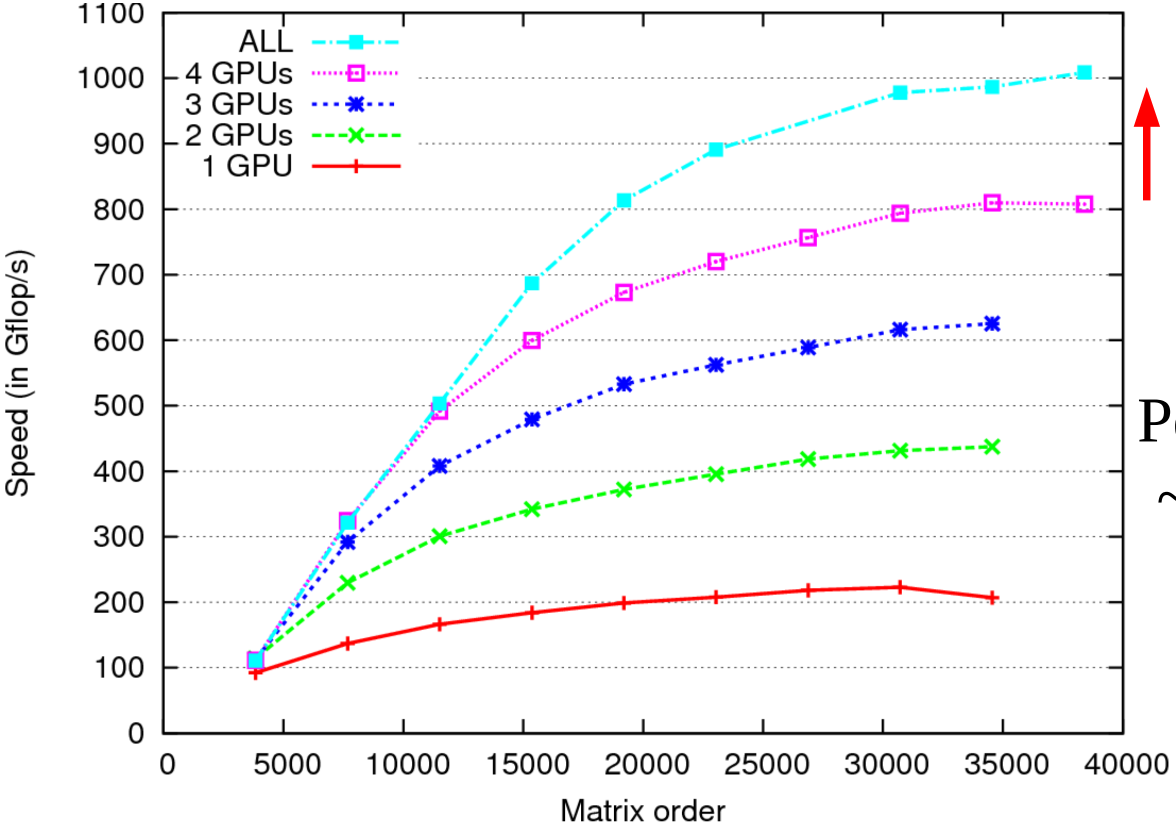
Mixing PLASMA and MAGMA with StarPU

- 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

Peak : 12 cores
~150 GFlops

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

Scheduling parallel tasks

Parallel tasks

Why do we need parallel tasks ?

- Take advantage of multicore architectures

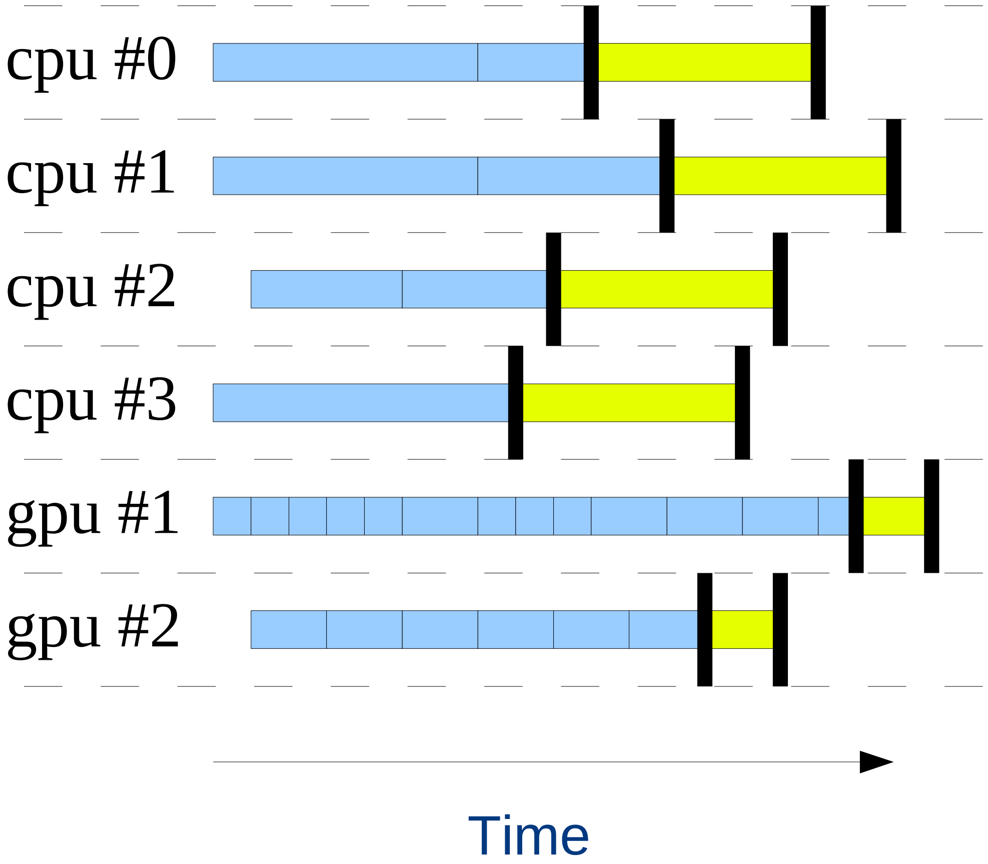
- Task parallelism may not be suited at a fine grain
  - Use other parallel paradigms
    - eg. OpenMP
  - Use existing parallel libraries
    - eg. do not reimplement parallel BLAS …

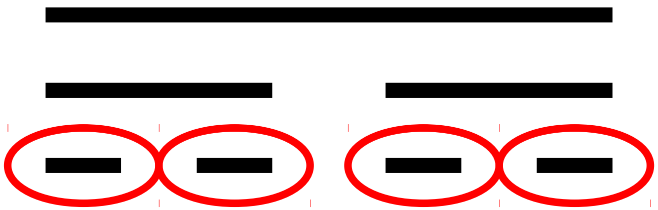
- Alleviate granularity concerns
  - Less tasks
  - Large enough tasks (suited for the GPU)

Parallel tasks

Scheduling parallel tasks

- StarPU allocates processing units

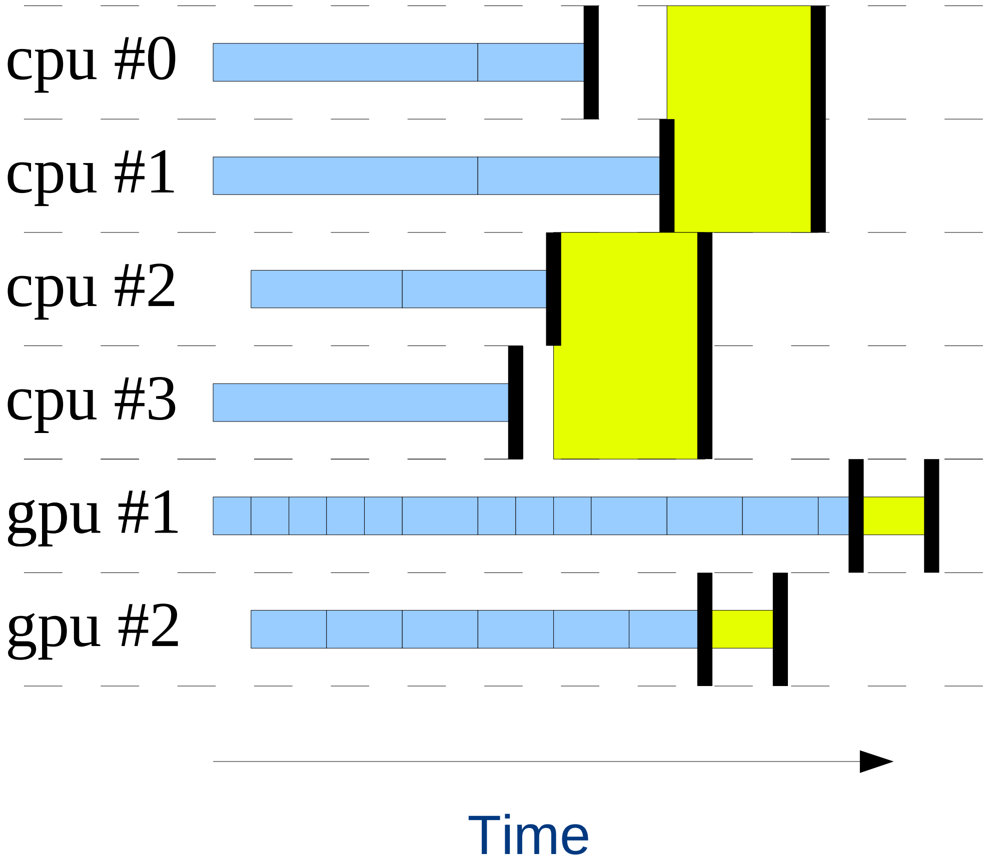


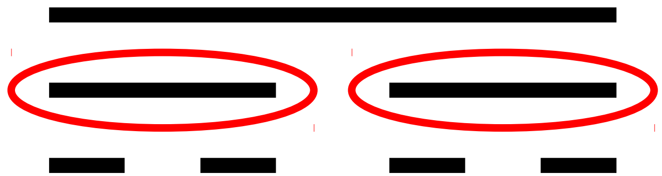


Parallel tasks

Scheduling parallel tasks

- StarPU allocates processing units

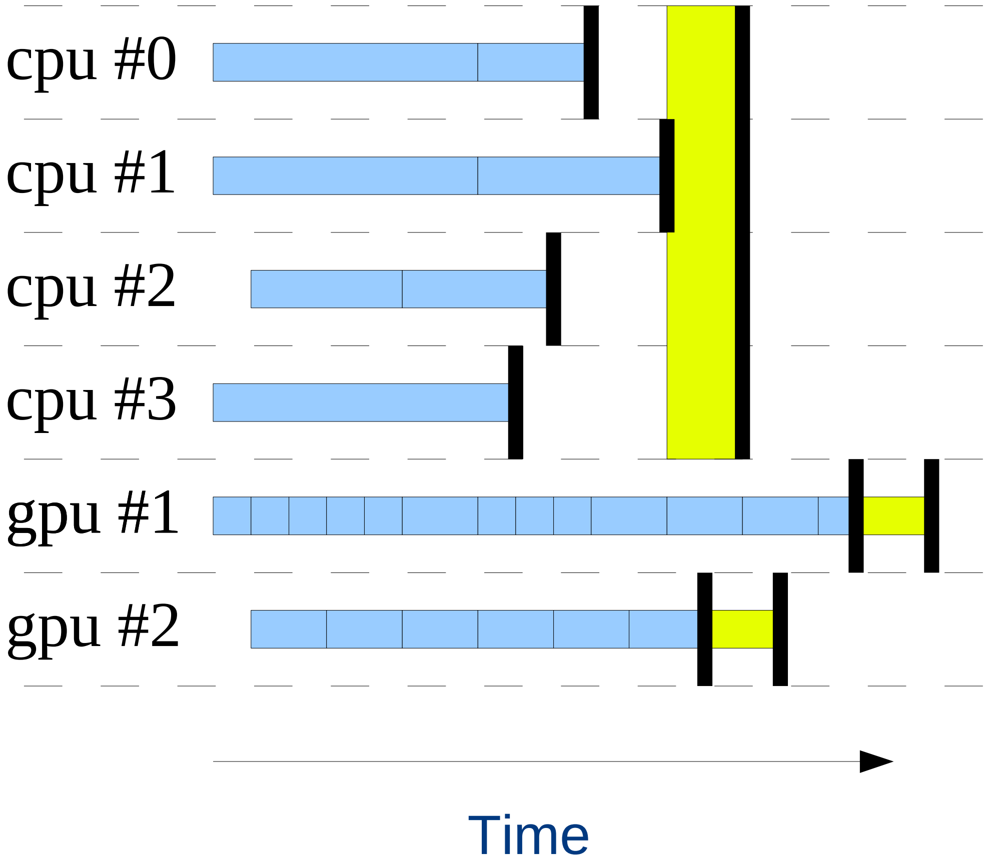


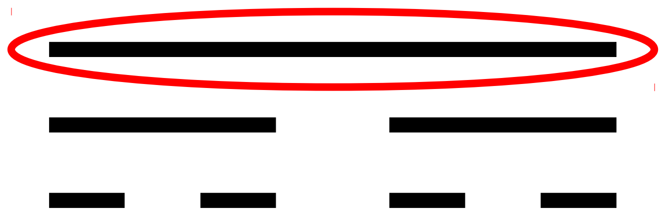


Parallel tasks

Scheduling parallel tasks

- StarPU allocates processing units

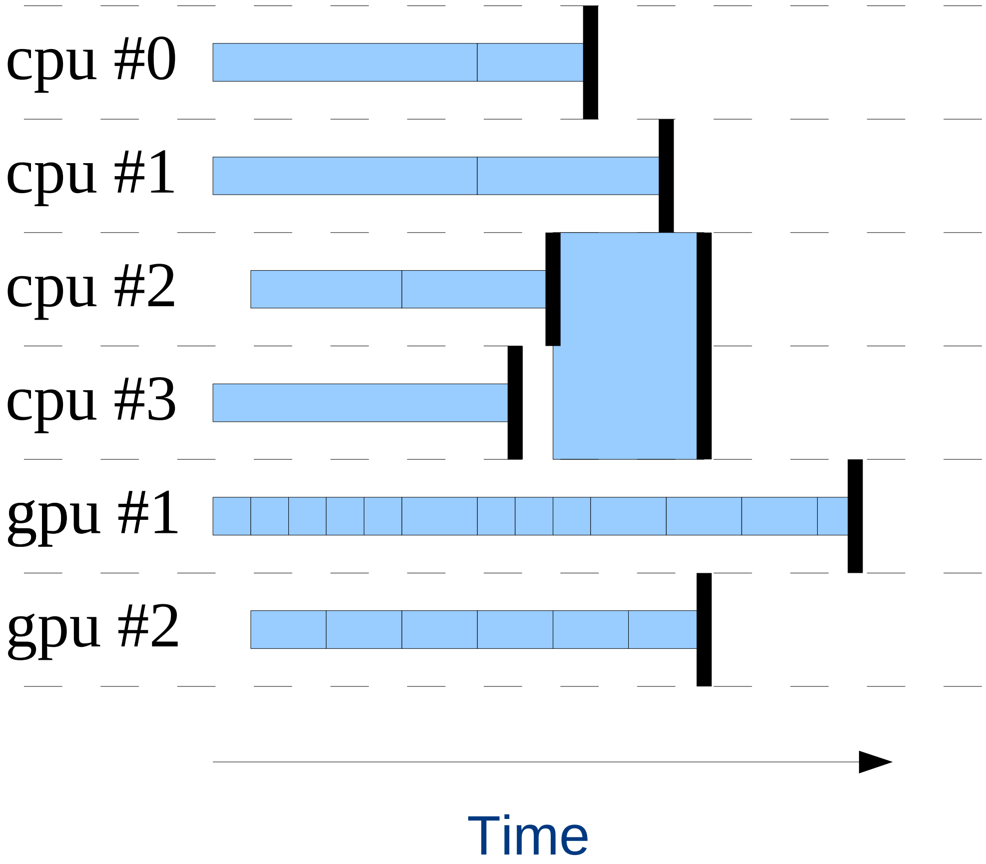




Parallel tasks

Scheduling parallel tasks

- StarPU allocates processing units



Adding support for MPI in StarPU

Accelerating MPI applications with StarPU

- Keep MPI SPMD style
  - Static distribution of data
  - Scheduling within the node only
    - No load balancing between MPI processes
- Inter-process data dependencies
  - MPI communications triggered by StarPU data availability
  - Support from StarPU's memory management
    - Automaticallly construct MPI datatype

Accelerating MPI applications with StarPU

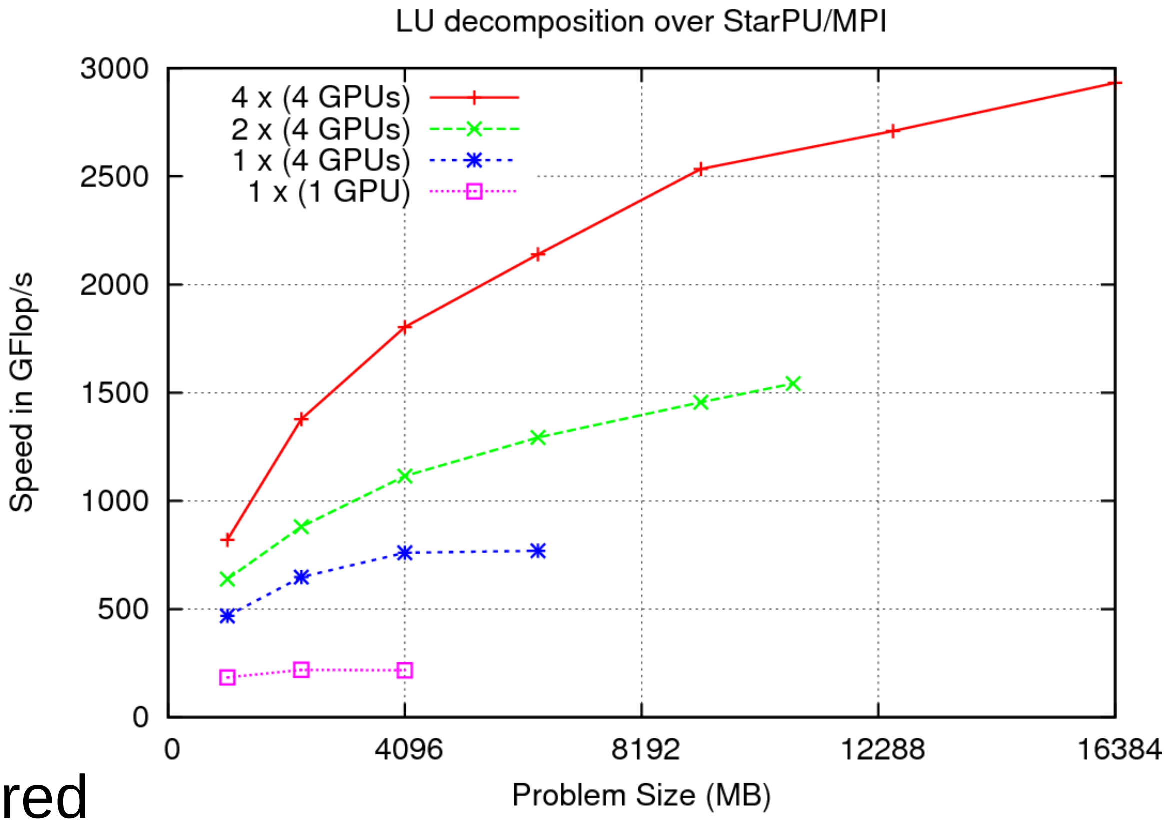
- Provided API
  - starpu_mpi_{send,recv}
  - starpu_mpi_{isend,irecv}
  - starpu_mpi_{test,wait}
  - starpu_mpi_{send,recv}_detached
  - starpu_mpi_*_array

- Detached calls
  - No need to explicitly test/wait for the request
  - Automatic progression

- Automatic data dependencies
  - MPI transfers ~ StarPU tasks
  - Accelerating legacy codes

Accelerating LU/MPI with StarPU

- LU decomposition
  - MPI+multiGPU
- Static MPI distribution
  - 2D block cyclic
  - ~SCALAPACK
  - No pivoting !
- Algorithmic work required
  - Collaboration with UTK

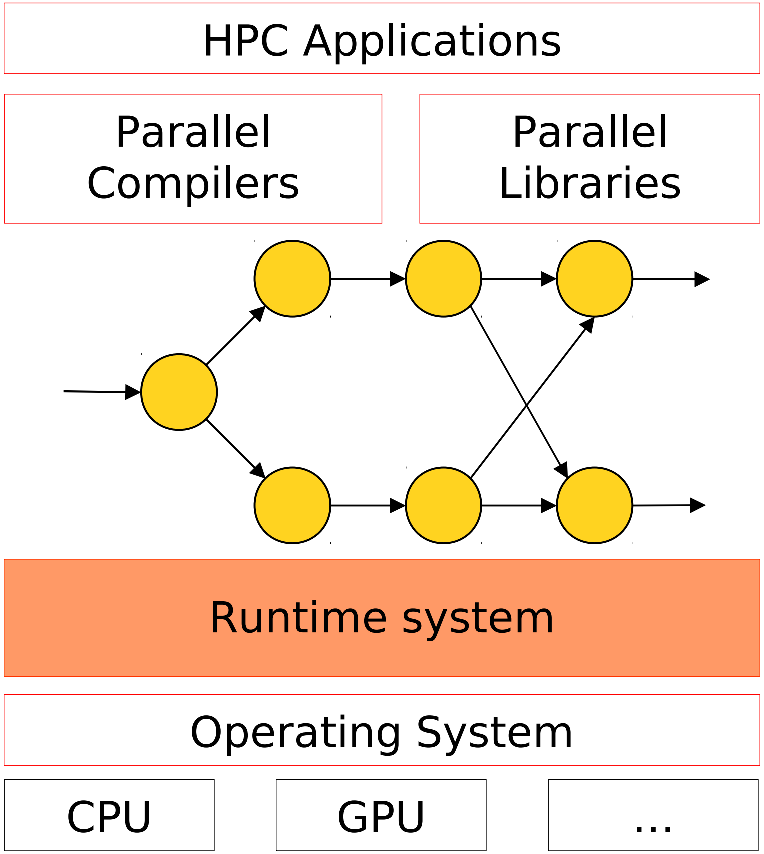


Conclusion

Conclusion

Summary

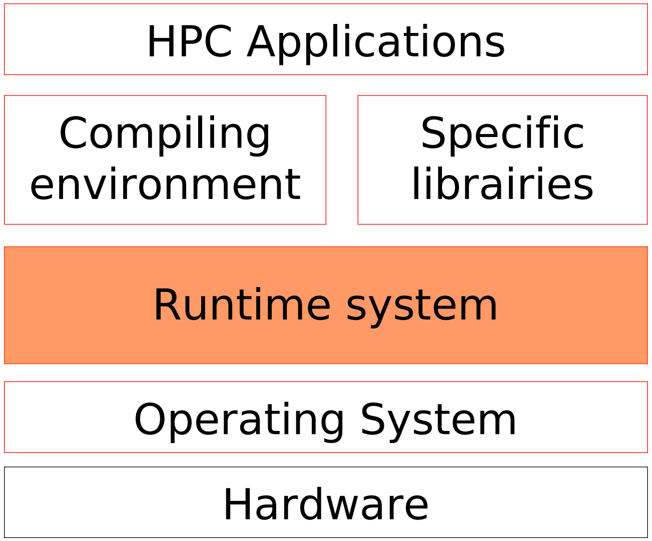
- StarPU
  - Freely available under LGPL
  - Available on Linux, OS/X, Windows
  - Open to external contributors!
- Task Scheduling
  - Required on hybrid platforms
  - Auto-tuned performance models
- Combined PLASMA and MAGMA
- Parallel tasks
- MPI extensions



Conclusion

Future work

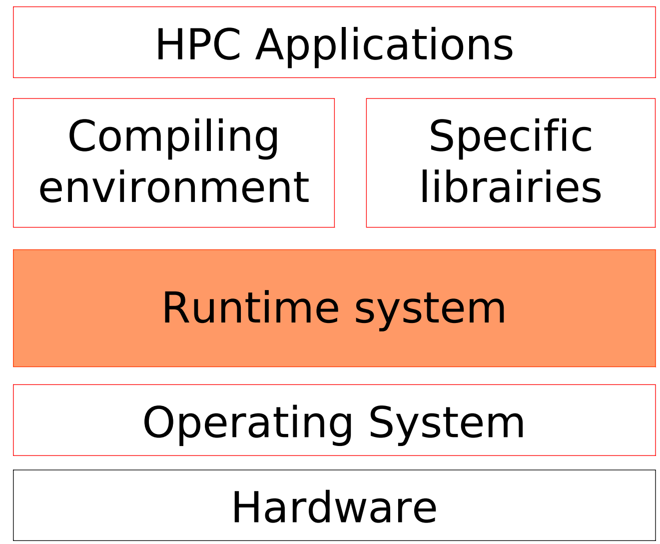
- Implement more algorithms
  - LU, Hessenberg
  - Communication Avoiding algorithms
  - Hybrid Scalapack
  - Provide higher level constructs (eg. reductions)
- Provide a back-end for compilers
  - StarSs, XscalableMP, HMPP
- Support new architectures
  - Intel SCC, Fermi cards, …
- Dynamically adapt granularity
  - Divisible tasks



Conclusion

Future work

- Implement more algorithms
  - LU, Hessenberg
  - Communication Avoiding algorithms
  - Hybrid Scalapack
  - Provide higher level constructs (eg. reductions)
- Provide a back-end for compilers
  - StarSs, XscalableMP, HMPP
- Support new architectures
  - Intel SCC, Fermi cards, …
- Dynamically adapt granularity
  - Divisible tasks

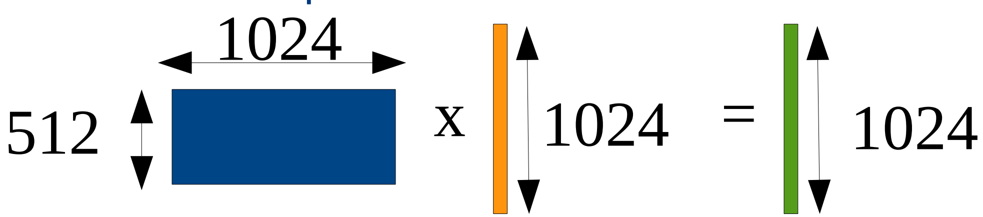


Thanks for your attention !

Any question ?

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)

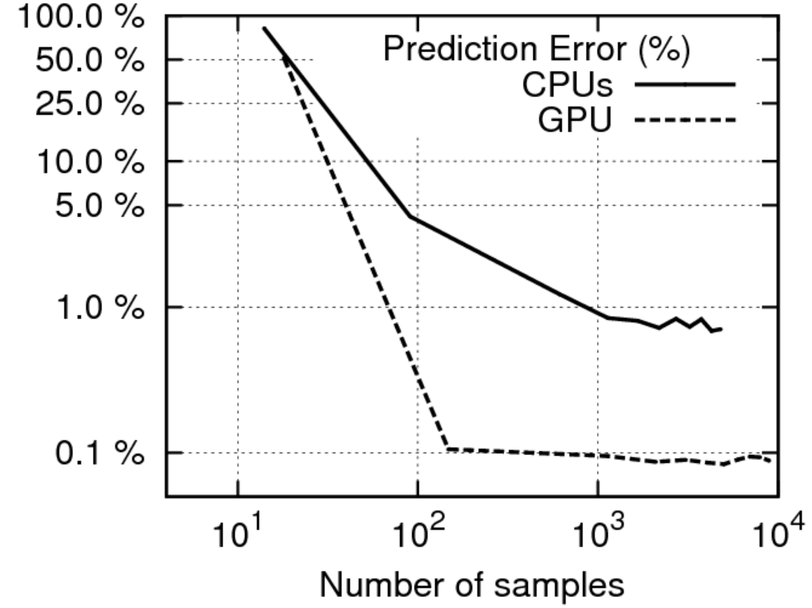
  - Data
    - Signature(Di) = CRC(p1, p2, … , pk)
  - Task ~ Series of data
    - Signature(D1, ..., Dn) = CRC(sign(D1), ..., sign(Dn))

- 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 ±□.□□**

- Faster
- No code change !
- More stable



- Dynamic calibration
- Simple, but accurate