Composing multiple StarPU applications over heterogeneous machines: a supervised approach

Andra Hugo

With Abdou Guermouche, Pierre-André Wacrenier, Raymond Namyst

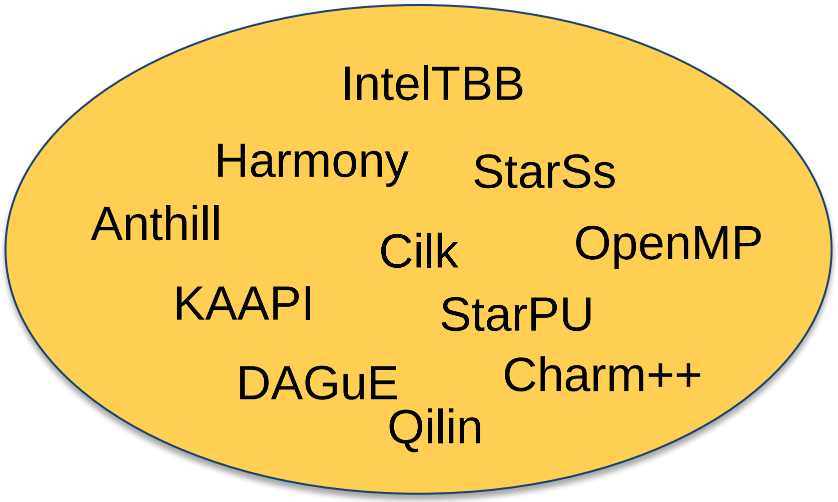
Inria, LaBRI, University of Bordeaux

RUNTIME
INRIA Group
INRIA Bordeaux Sud-Ouest

The increasing role of runtime systems

Code reusability

- Many HPC applications rely on specific parallel libraries
  - Linear algebra, FFT, Stencils
- Efficient implementations sitting on top of dynamic runtime systems
  - To deal with hybrid, multicore complex hardware
    - E.g. MKL/OpenMP, MAGMA/StarPU
  - To avoid reinventing the wheel!
- Some application may benefit from relying on multiple libraries
  - Potentially using different underlying runtime systems…



The increasing role of runtime systems

Code reusability

- Many HPC applications rely on specific parallel libraries
  - Linear algebra, FFT, Stencils
- Efficient implementations sitting on top of dynamic runtime systems
  - To deal with hybrid, multicore complex hardware
    - E.g. MKL/OpenMP, MAGMA/StarPU
  - To avoid reinventing the wheel!
- Some application may benefit from relying on multiple libraries
  - Potentially using different underlying runtime systems…

IntelTBB
Harmony
StarSs
Anthill
Cilk
OpenMP
KAAPI
StarPU
DAGuE
Charm++
Qilin

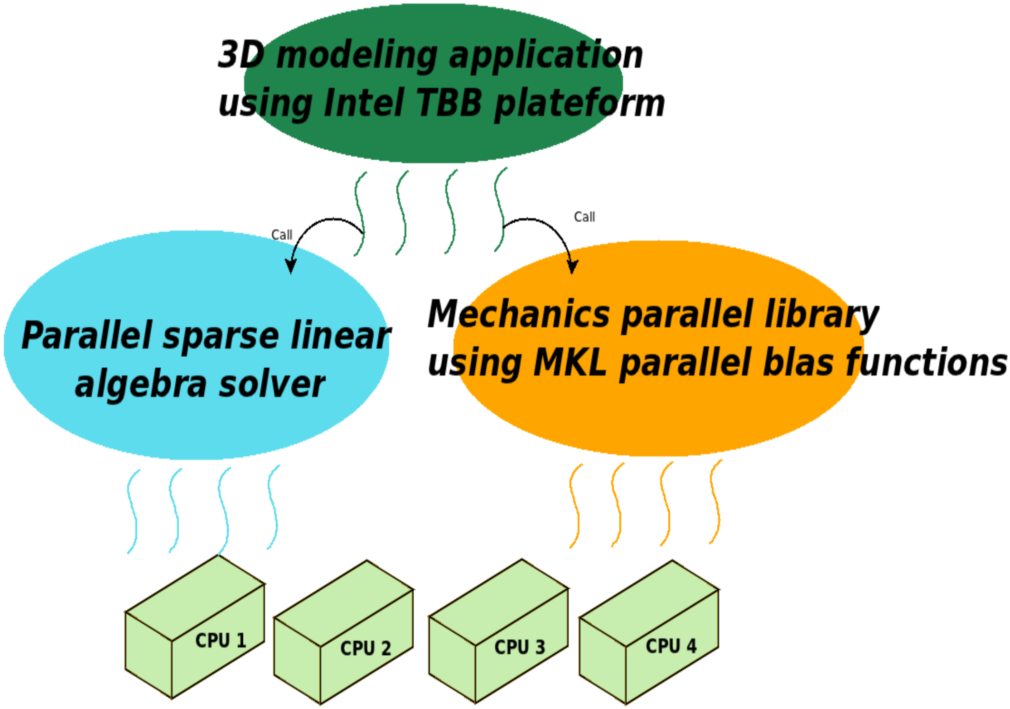
=> Are parallel libraries ready to run simultaneously over the same hardware resources ?

Struggle for resources

- Parallel libraries typically allocate and bind one thread per core
  - Bypass the OS scheduler
  - Control cache utilization
- Each library is unaware of the resource management of the other ones

  => resource oversubscription
- Optimizations (cache affinity, memory reuse, etc.) are strongly affected

Interferences between parallel libraries

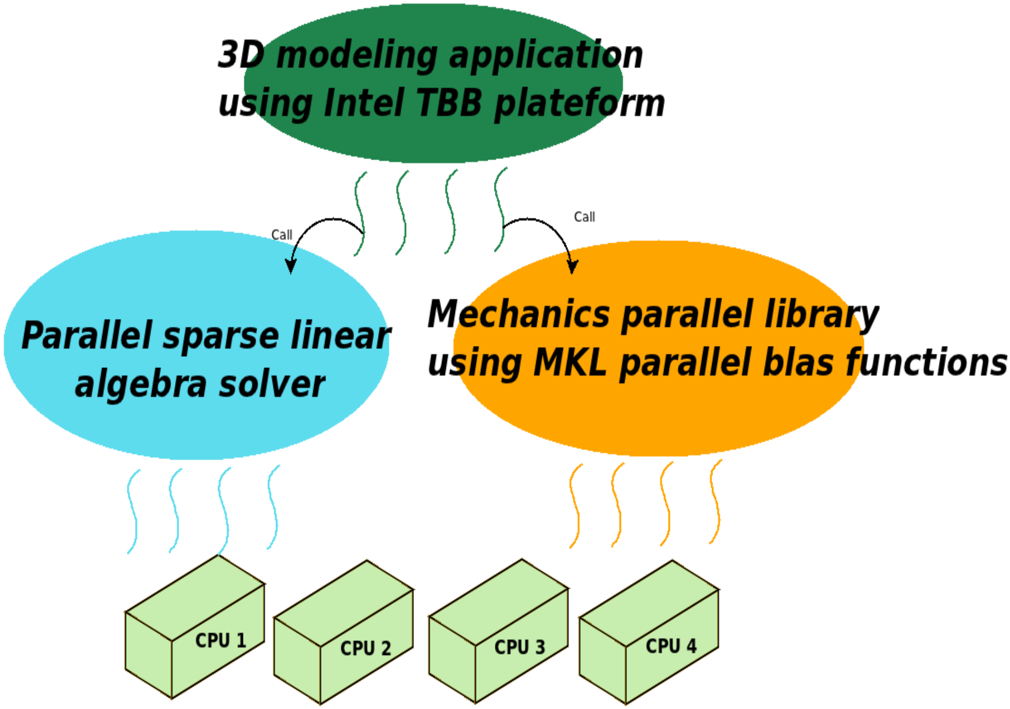


Struggle for resources

Interferences between parallel libraries

- Parallel libraries typically allocate and bind one thread per core
  - Bypass the OS scheduler
  - Control cache utilization
- Each library is unaware of the resource management of the other ones

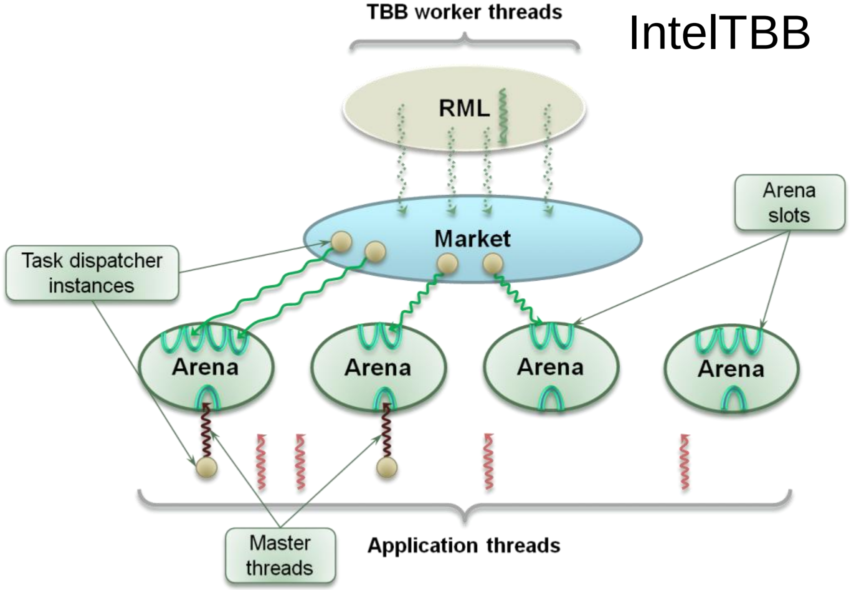
  => resource oversubscription
- Optimizations (cache affinity, memory reuse, etc.) are strongly affected



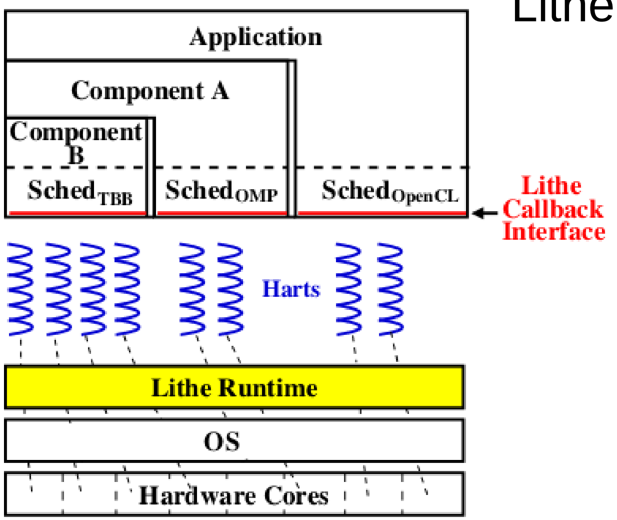
=> Composability problem

Composability problem

How to deal with it?



software.intel.com



Phd Thesis : Cooperative Hierarchical Resource Management for Efficient Composition of Parallel Software, Heidi Pan, 2010

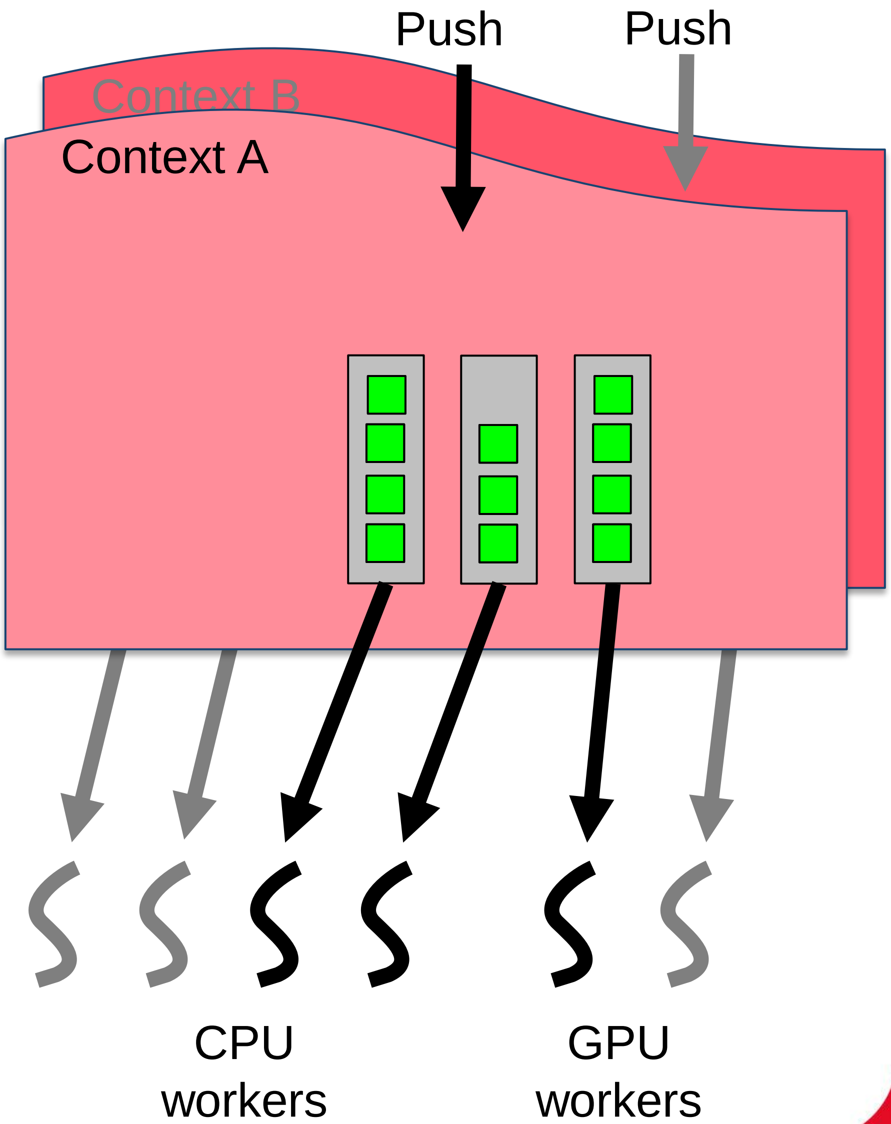
- Advanced environments allow partitioning hardware resources
  - IntelTBB:
    - The pool of workers are split in arenas
  - Lithe
    - Resource sharing management interface
    - Harts are transferred between parallel libraries
- Main challenge: Automatically adjusting the amount of resources allocated to each library

Scheduling Contexts

- Isolate concurrent parallel codes
- Similar to lightweight virtual machines
- Run on top of subsets of available PUs
- Minimize interferences
- Enforce data locality

- Contexts may expand and shrink
  - Hypervised approach
  - Maximize overall throughput
  - Use dynamic feedback both from application and runtime

Toward code composability

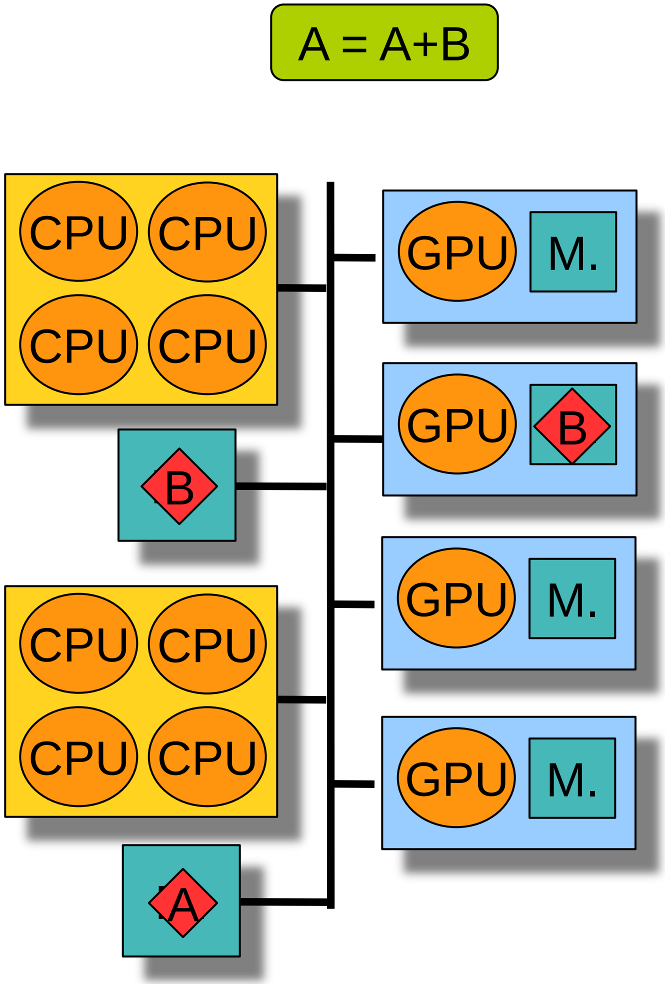


Using StarPU as an experimental platform

A runtime system for *PU architectures

for studying resource negociation

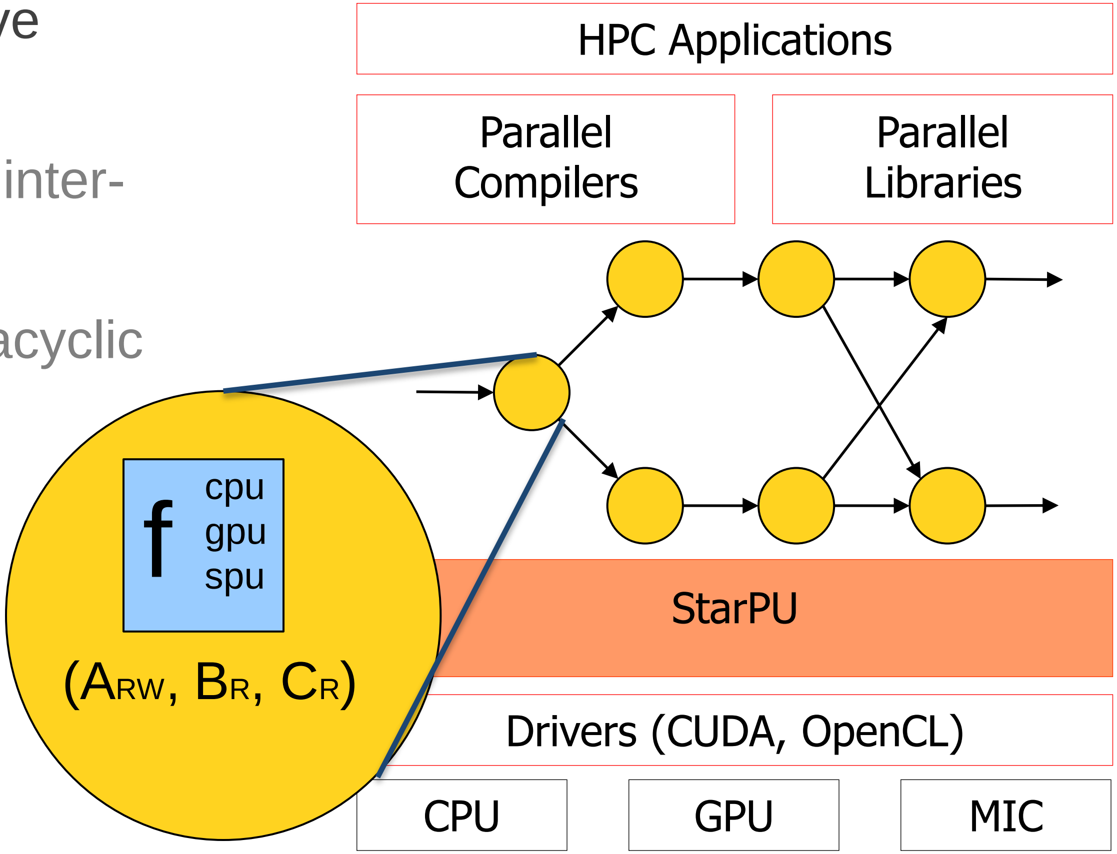
- The StarPU runtime system
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous processing units
  - Avoid unnecessary data transfers between accelerators
    - Software VSM for heterogeneous machines



Overview of StarPU

Maximizing PU occupancy, minimizing data transfers

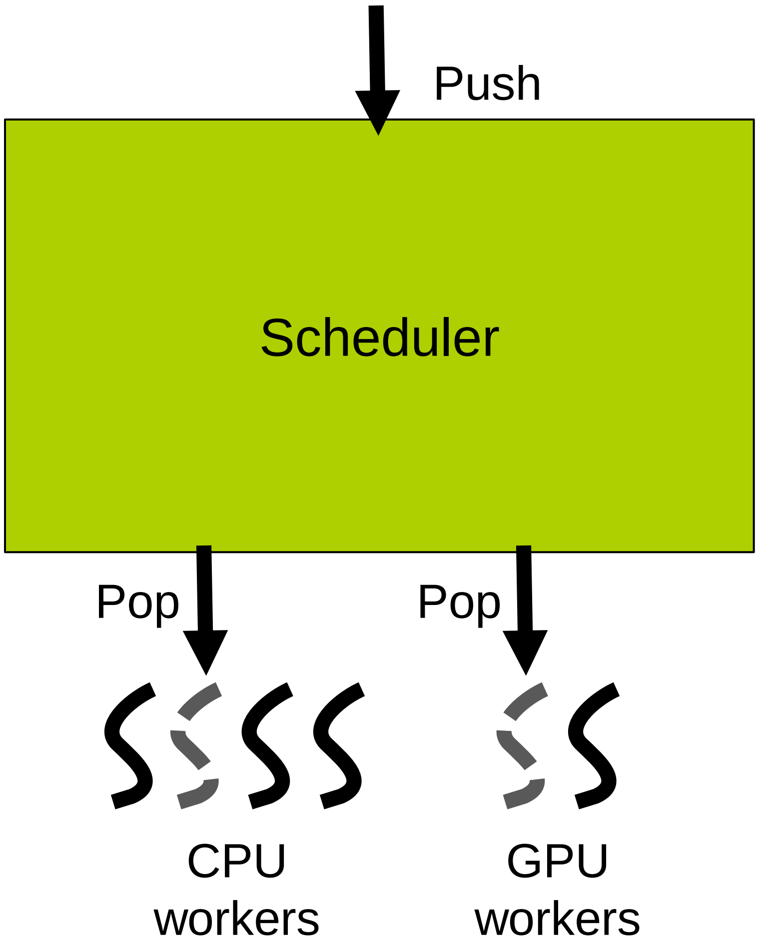
- Accept tasks that may have multiple implementations
  - Together with potential inter-dependencies
    - Leads to a dynamic acyclic graph of tasks
    - Data-flow approach
  - Scheduling hints
- Open, general purpose scheduling platform
  - Scheduling policies = plugins



Tasks scheduling

How does it work?

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



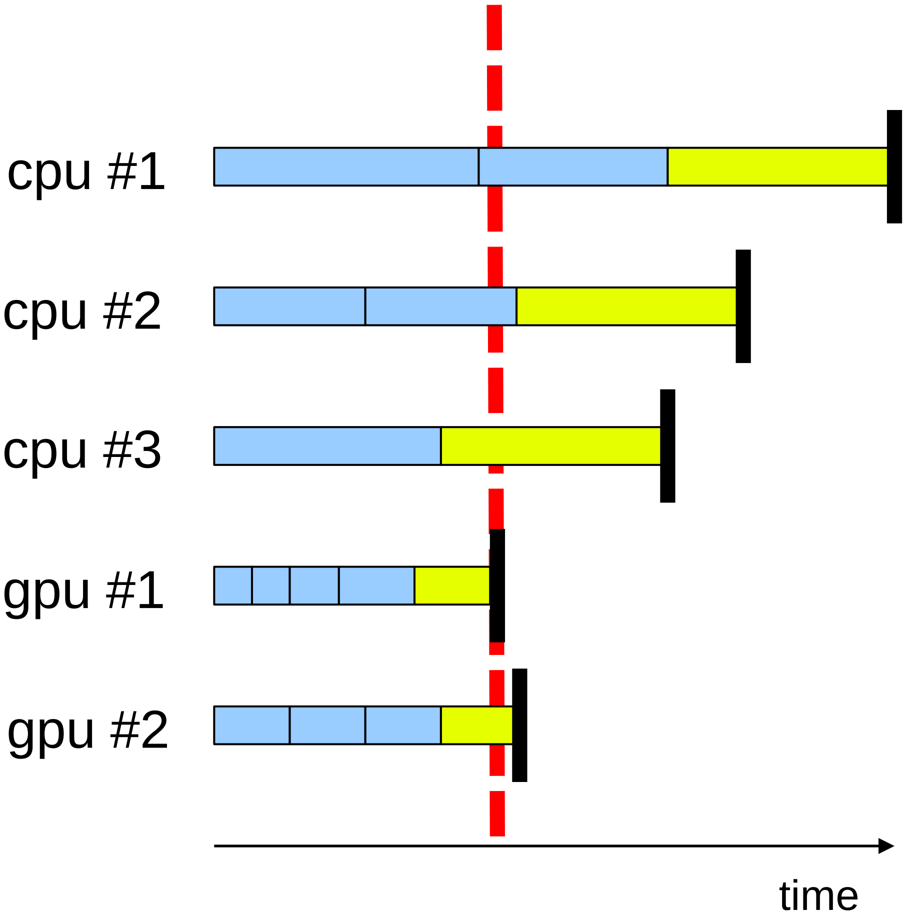
Dealing with heterogeneous architectures

Performance prediction

- Build-in schedulers:
  - E.g. Minimum Completion Time heuristic

- Task completion time estimation
  - History-based performance models

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

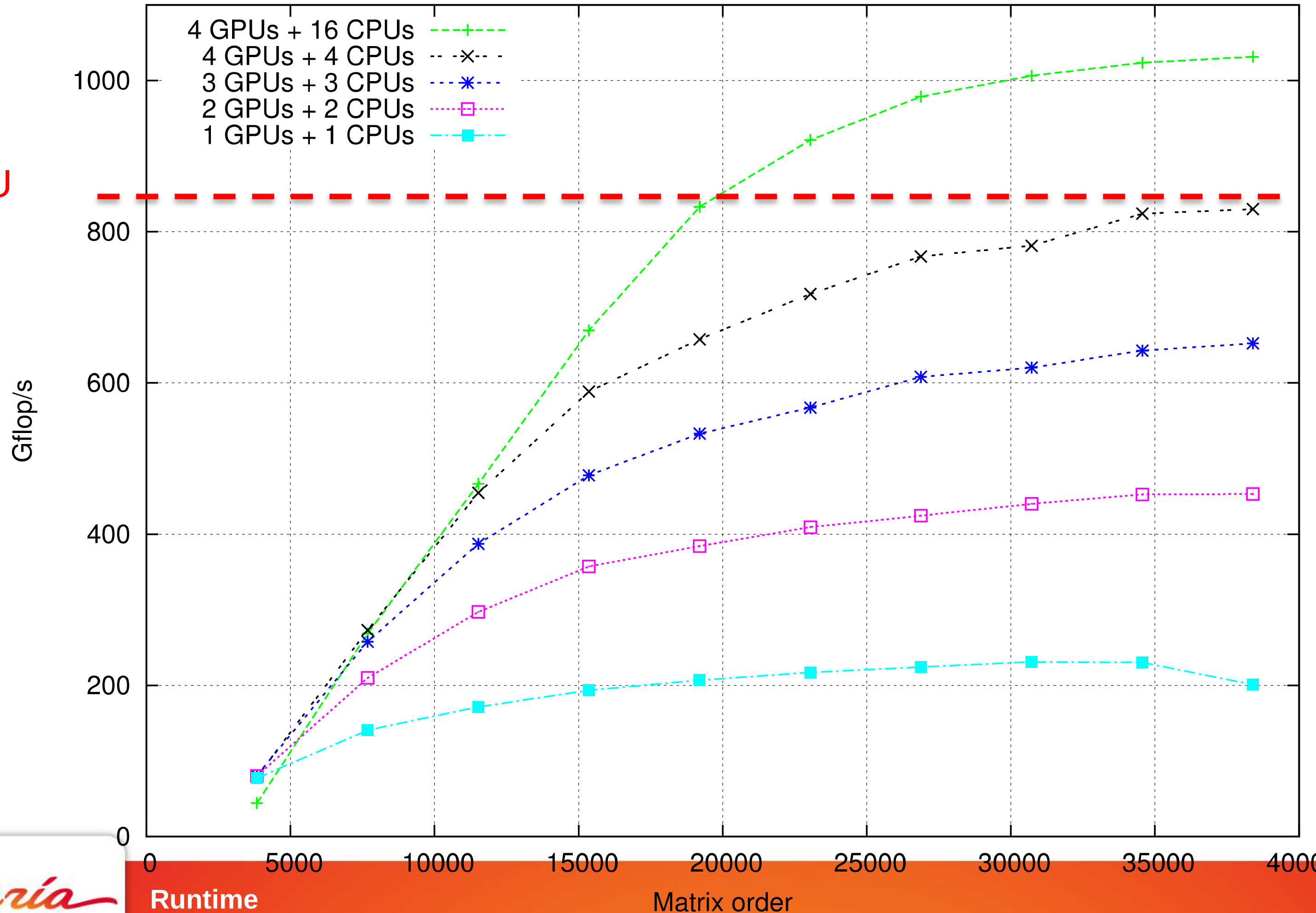


MAGMA with StarPU

With University of Tennessee & INRIA HiePACS

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

GPU only



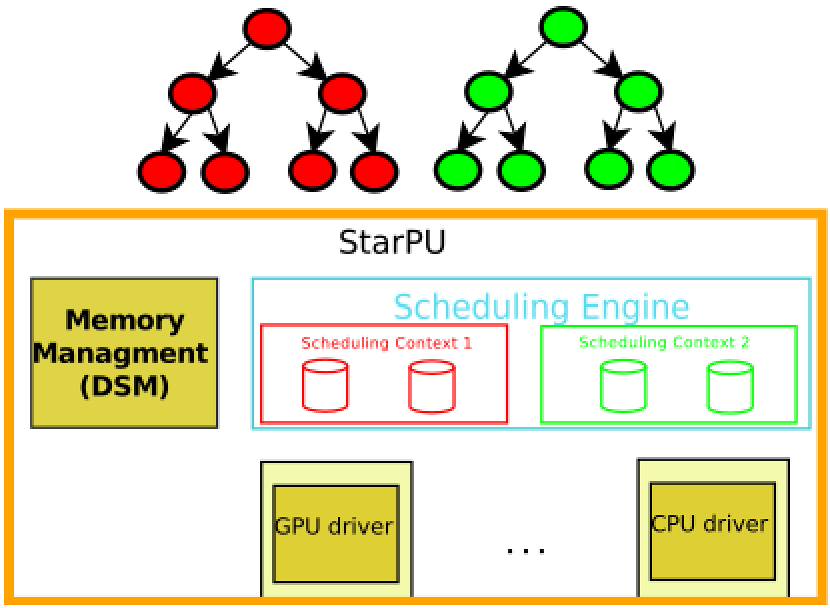
+12 CPUs
~200 GFlops

(although
12 CPUs alone
~150 Gflops)

Scheduling Contexts in StarPU

Extension of StarPU

- Heterogeneous machines
- Virtualization of resources
- Each context features its own scheduler
- Scalability workaround
- Contexts may share processing units
  - Avoid underutilized resources
- Allocation of resources
  - The programmer entirely defines it
  - The programmer specifies an interval and leaves the contexts negotiate
  - The hypervisor computes it



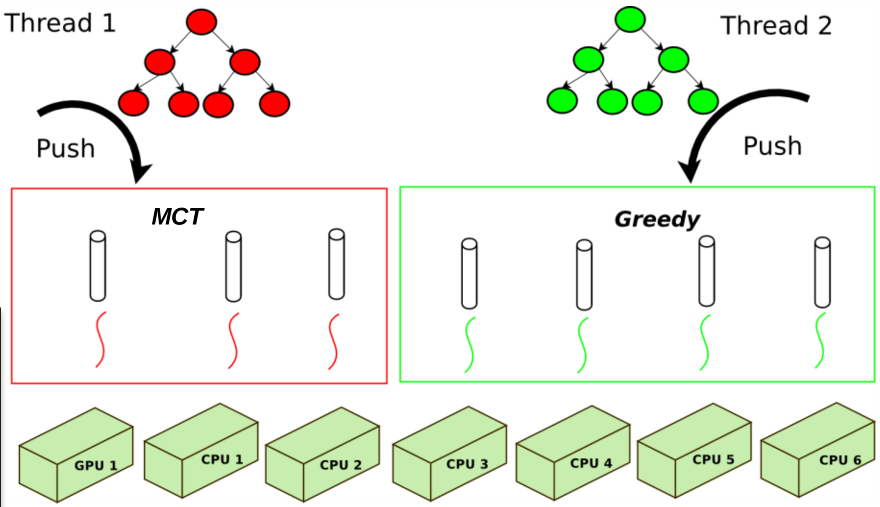
⇒ They try to be considerate with each other’s needs

Scheduling contexts in StarPU

Easily use contexts in your application

```
int resources1[3] = {CPU_1, CPU_2, GPU_1};
int resources2[4] = {CPU_3, CPU_4, CPU_5,
CPU_6};

/* define the scheduling policy and the table
  of resource ids */
```

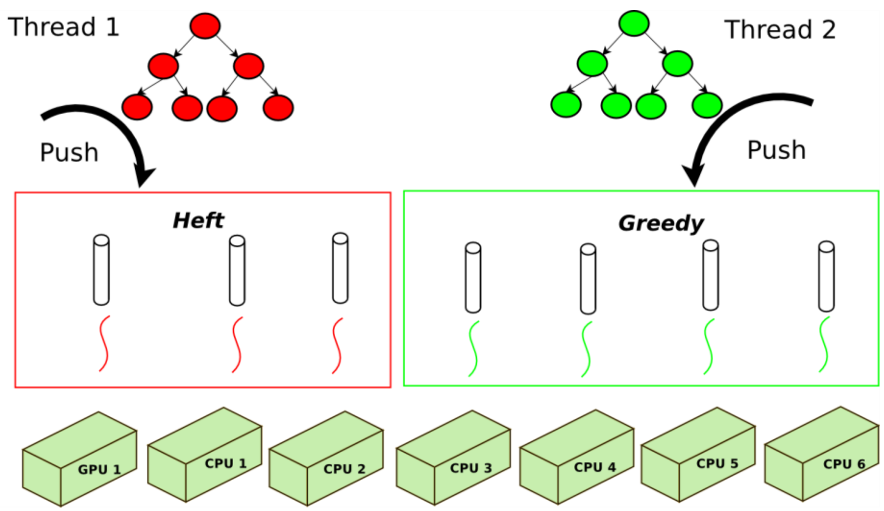
```
sched_ctx1 =
starpu_create_sched_ctx("mct",resources1,3);

sched_ctx2 =
starpu_create_sched_ctx("greedy",resources2,4);
```



Scheduling contexts in StarPU

Easily use contexts in your application

```
int resources1[3] = {CPU_1, CPU_2, GPU_1};
int resources2[4] = {CPU_3, CPU_4, CPU_5,
CPU_6};

/* define the scheduling policy and the table
  of resource ids */

sched_ctx1 =
starpu_create_sched_ctx("heft",resources1,3);

sched_ctx2 =
starpu_create_sched_ctx("greedy",resources2,4);
```



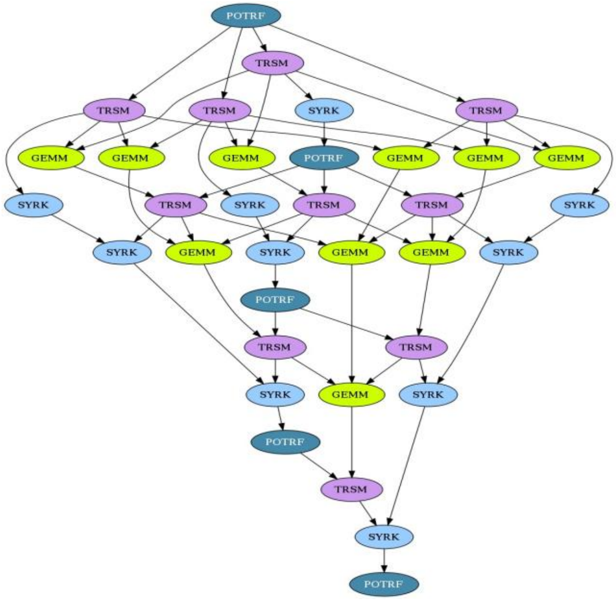

```
// thread 1:
/* define the context associated to kernel 1 */
starpu_set_sched_ctx(sched_ctx1);

/* submit the set of tasks of the parallel kernel
1*/
for( i = 0; i < ntasks1; i++)
   starpu_task_submit(tasks1[i]);
```

```
// thread 2:
/* define the context associated to kernel 2 */
starpu_set_sched_ctx(sched_ctx2);

/* submit the set of tasks of parallel kernel 2*/
for( i = 0; i < ntasks2; i++)
   starpu_task_submit(tasks2[i]);
```

Experimental evaluation

Platform and Application

- **9 CPUs** (two Intel hexacore processors, 3 cores devoted to execute GPU drivers) + **3 GPUs**
- MAGMA Linear Algebra Library
  - Implementation based on StarPU
  - Cholesky Factorization kernel
- Euler3D solver
  - Computational Fluid Dynamic benchmark
  - Rodinia benchmark suite
  - Iterative solver for 3D Euler equations for compressible fluids
  - Implementation based on StarPU

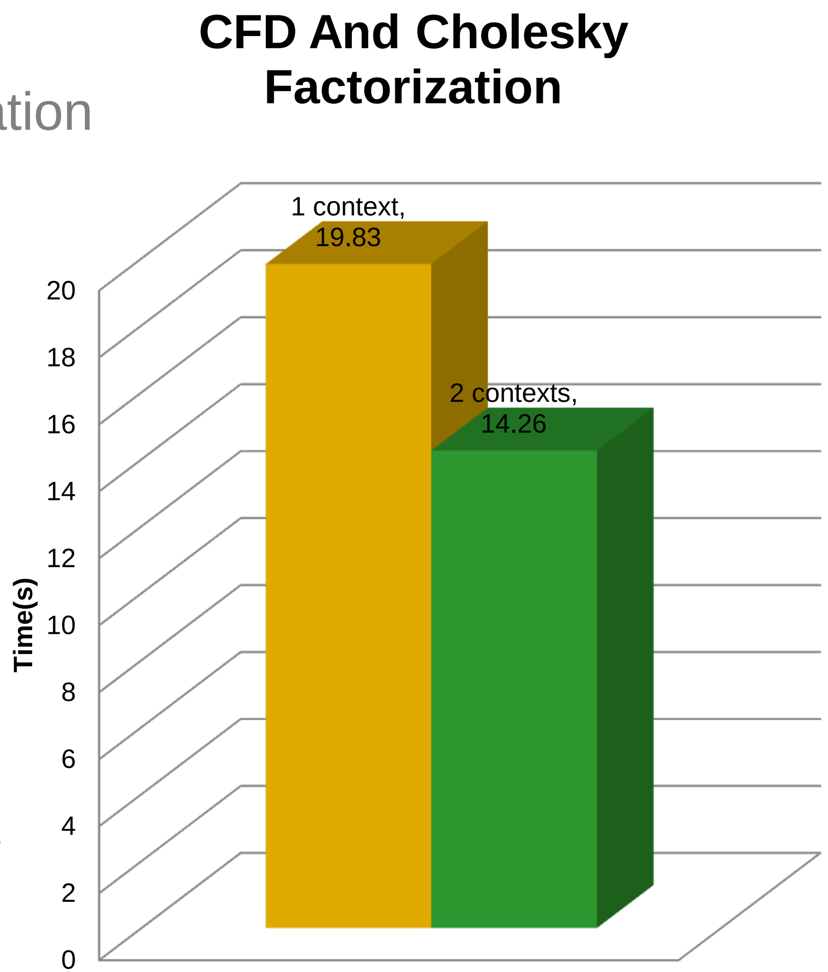


MAGMA – Cholesky Factorization

Composing Magma and the Euler3D solver

Different parallel kernels

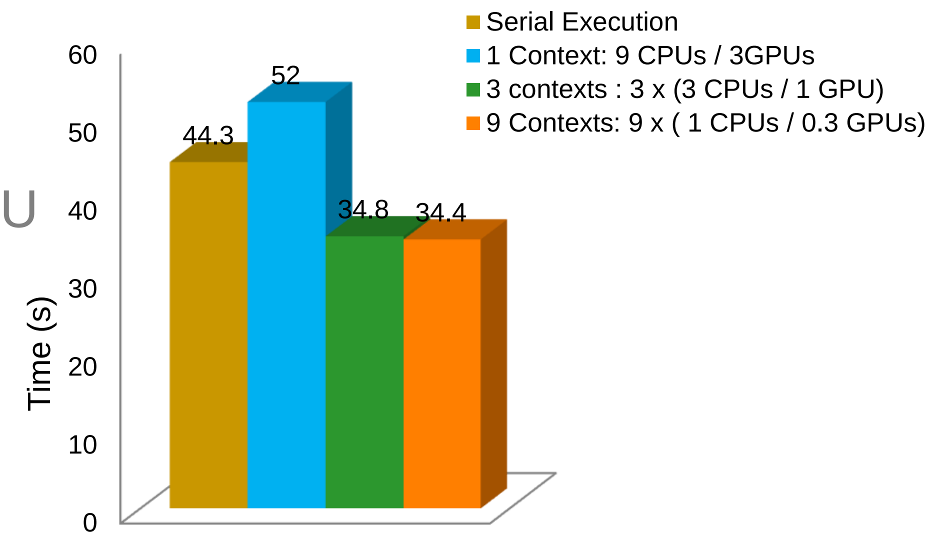
- CFD:
  - Domain decomposition parallelization
  - Independent tasks per iteration
  - Dependencies between iterations
  - Strong affinity with GPUs
  - 2 sub-domains: 2 GPUs
- Cholesky Factorization:
  - Scalable on both CPUs & GPUs
  - 1GPU & 9 CPUs
  - Large number of tasks
- Contexts enforce locality constraints

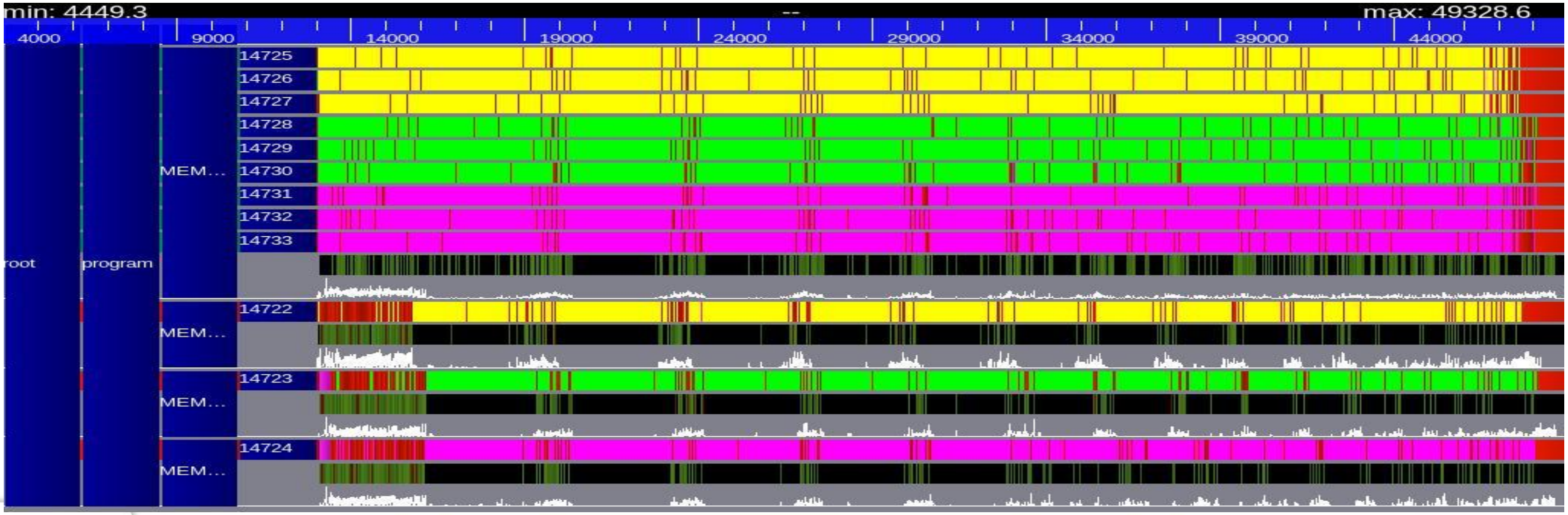


What about 9 Cholesky factorizations…?

Gain performance from data locality

- Mixing parallel kernels:
  - Unnecessary data transfers between Host Memory & GPU memory -> blocking waits
  - Memory flushes

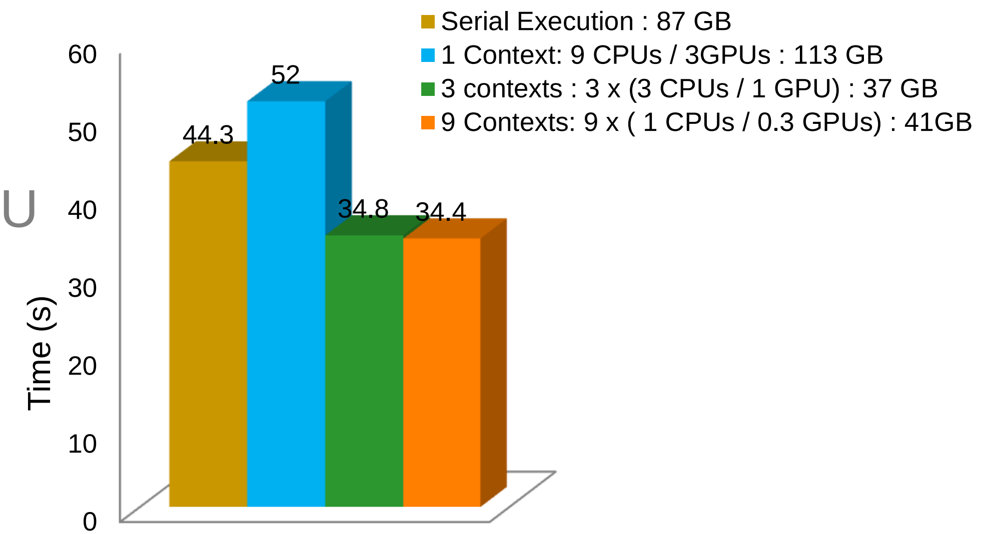


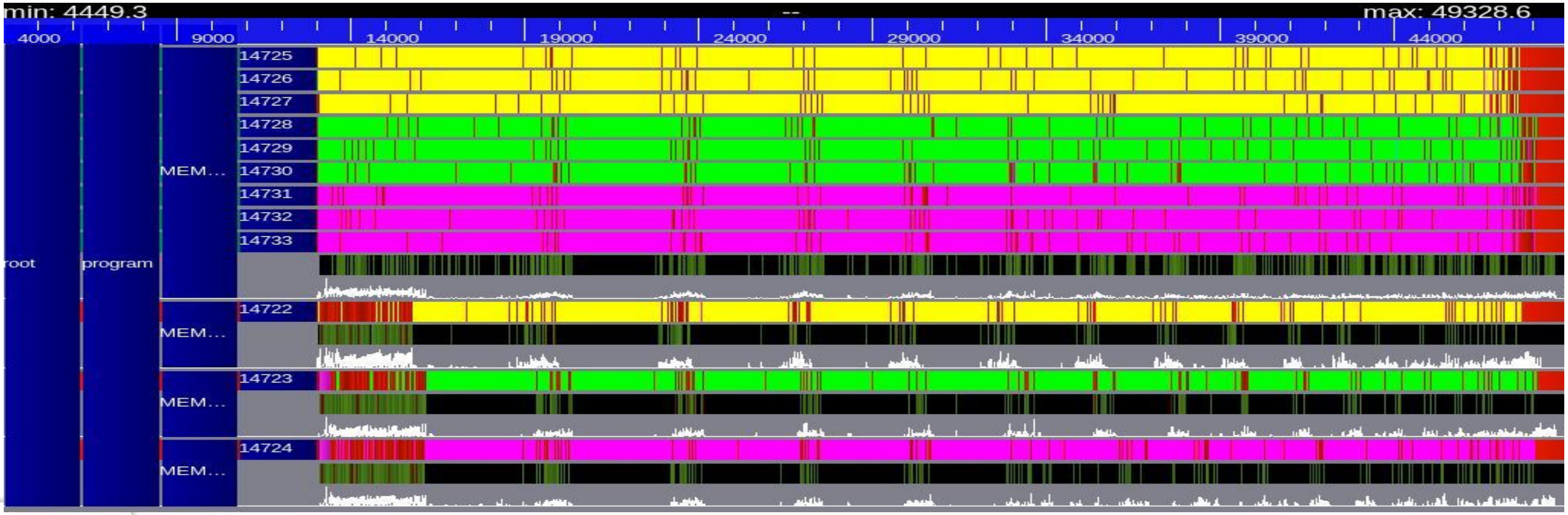


What about 9 Cholesky factorizations…?

Gain performance from data locality

- Mixing parallel kernels:
  - Unnecessary data transfers between Host Memory & GPU memory -> blocking waits
  - Memory flushes

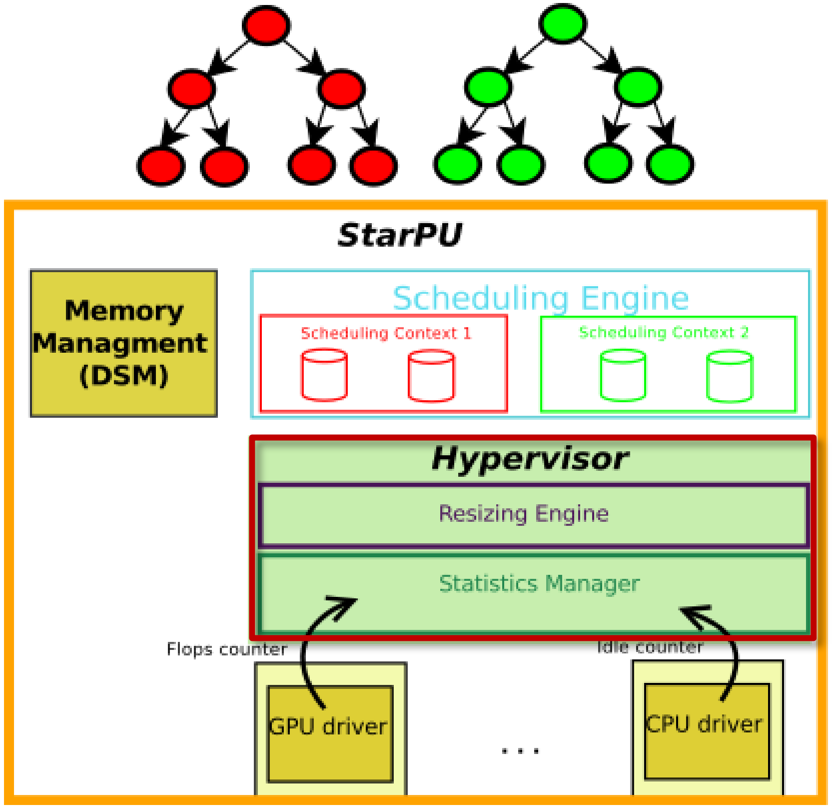




The Hypervisor

What if static dimensioning doesn't work?

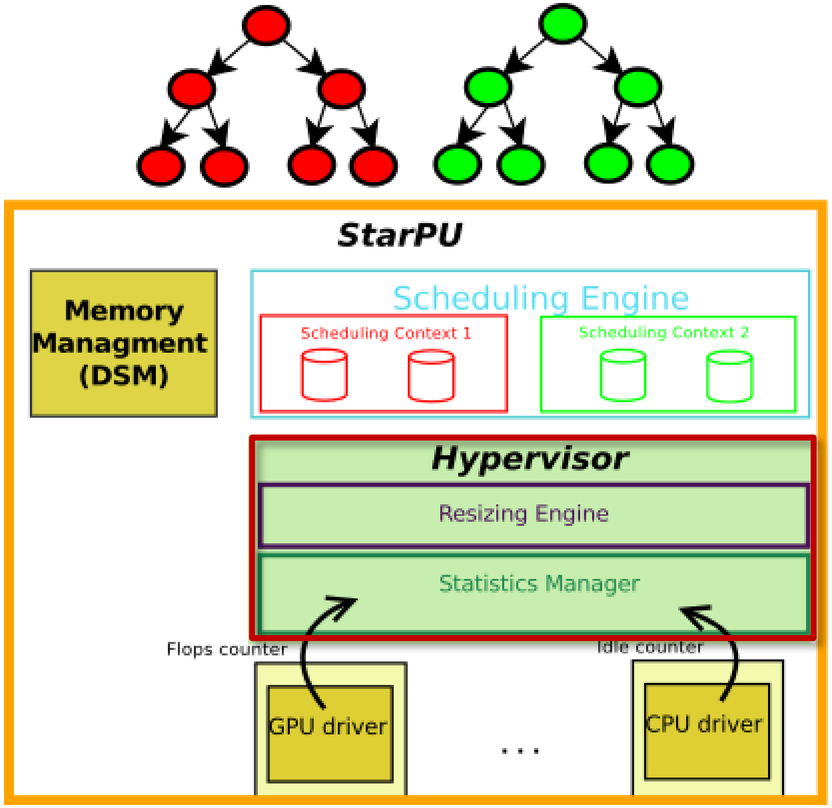
- Dynamically resizes scheduling contexts
- Triggers resizing:
  - At some instances of time
    - feedback from the application
  - When the initial configuration deteriorates the performances
    - feedback from the runtime
    - different metrics: Idle resources, Speed of the contexts
- User's constraints for the negotiation of resources
  - Idleness limit
  - Resize interval limitation



The Hypervisor

What if static dimensioning doesn't work?

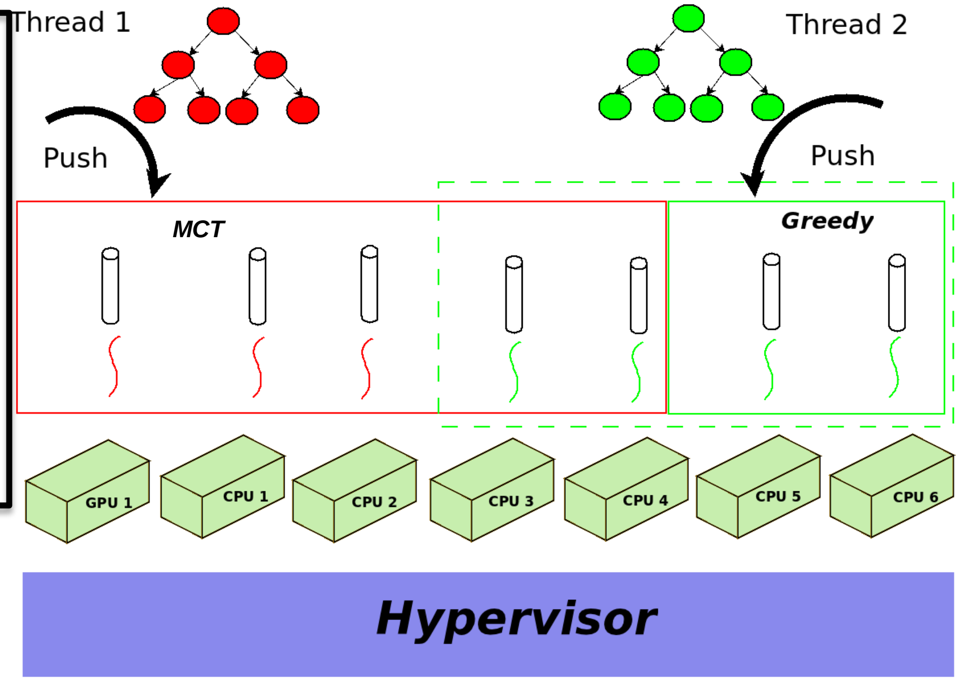
- Stores contexts information and resource performance statistics
- Policies for resizing contexts
  - Minimize the makespan
  - Need the workload of the parallel codes
  - Linear programs to compute the best allocation of resources



The Hypervisor in StarPU

```
/* select an existing resizing policy */
struct hypervisor_policy policy;
policy.name = "min_makespan_policy";

/* initialize the hypervisor and
set its resizing policy */
sched_ctx_hypervisor_init(policy);
```

Simple example



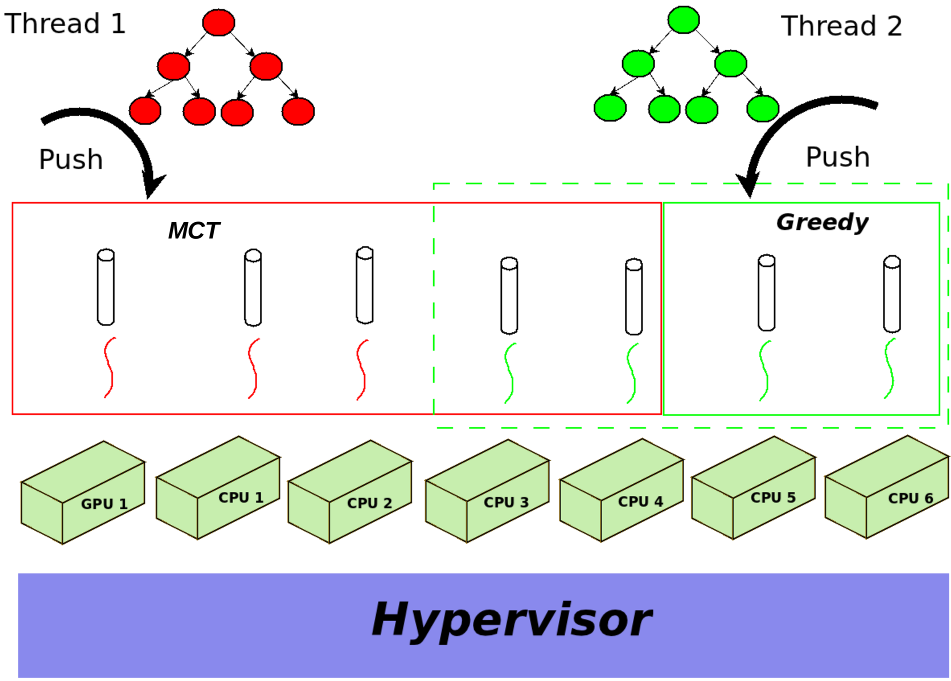

```
/* register context 1 to the hypervisor */
sched_ctx_hypervisor_register_ctx(sched_ctx1);

/* register context 2 to the hypervisor */
sched_ctx_hypervisor_register_ctx(sched_ctx2);
```

```
/* define the constraints for the resizing */
sched_ctx_hypervisor_ctl(sched_ctx1,
   HYPERVISOR_MIN_CPU_WORKERS, 3,
   HYPERVISOR_MAX_CPU_WORKERS, 7,
   NULL);
```

The Hypervisor in StarPU

Simple example

```
/* select an existing resizing policy */
struct hypervisor_policy policy;
policy.name = "min_makespan_policy";

/* initialize the hypervisor and
set its resizing policy */
sched_ctx_hypervisor_init(policy);
```

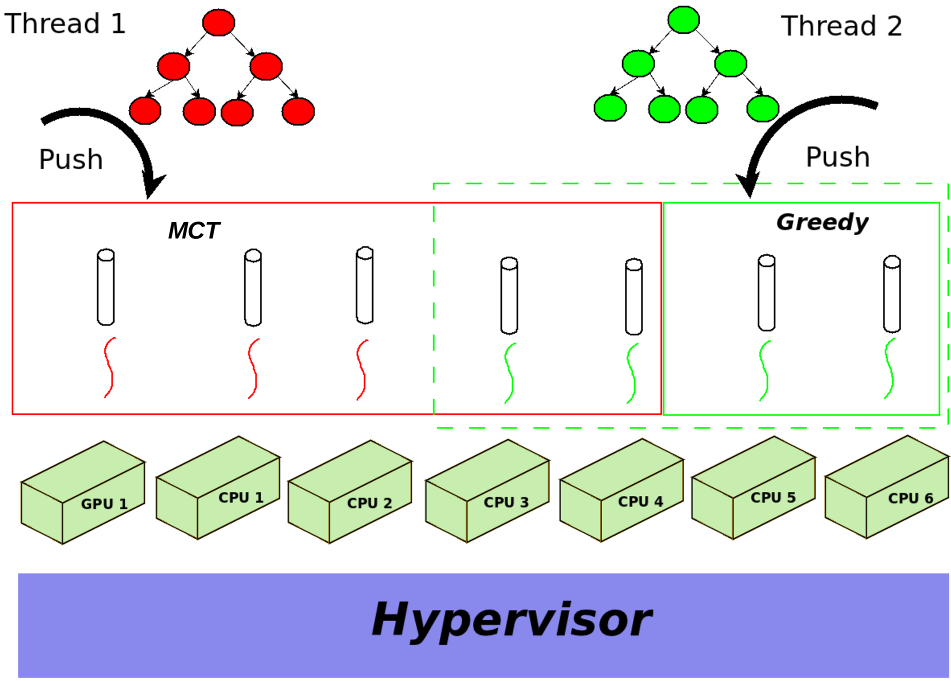



```
/* register context 1 to the hypervisor */
sched_ctx_hypervisor_register_ctx(sched_ctx1);

/* register context 2 to the hypervisor */
sched_ctx_hypervisor_register_ctx(sched_ctx2);
```

```
/* define the constraints for the resizing */
sched_ctx_hypervisor_ctl(sched_ctx1,
    HYPERVISOR_MIN_CPU_WORKERS, 3,
    HYPERVISOR_MAX_CPU_WORKERS, 7,
    NULL);
```

The Hypervisor in StarPU

Simple example



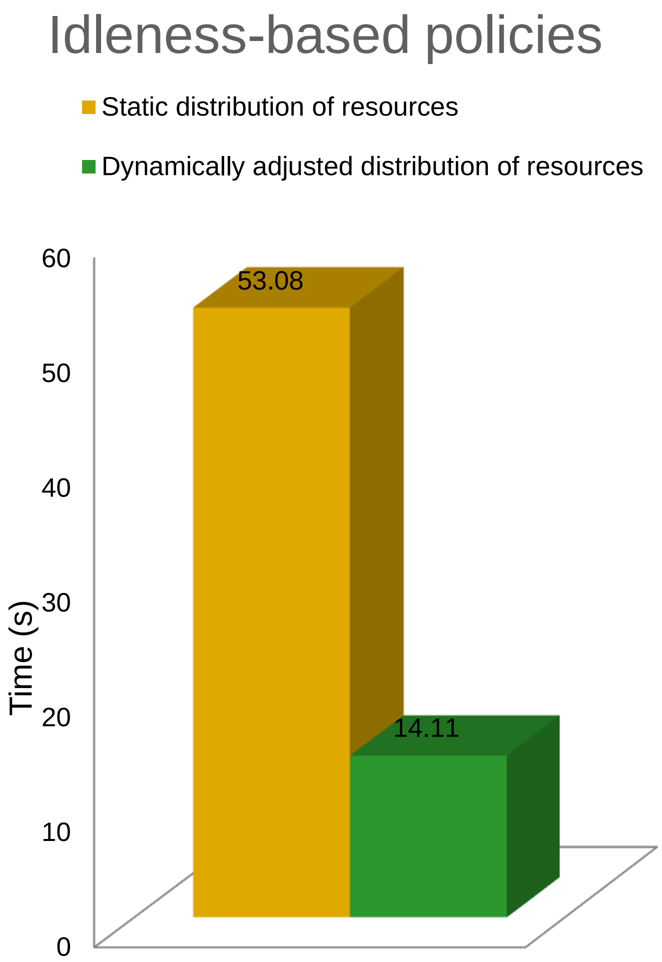

```
/* select an existing resizing policy */
struct hypervisor_policy policy;
policy.name = “min_makespan_policy";

/* initialize the hypervisor and
set its resizing policy */
sched_ctx_hypervisor_init(policy);

/* register context 1 to the hypervisor */
sched_ctx_hypervisor_register_ctx(sched_ctx1);

/* register context 2 to the hypervisor */
sched_ctx_hypervisor_register_ctx(sched_ctx2);
```

```
/* define the constraints for the resizing */
sched_ctx_hypervisor_ctl(sched_ctx1,
   HYPERVISOR_MIN_CPU_WORKERS, 3,
   HYPERVISOR_MAX_CPU_WORKERS, 7,
   NULL);
```

Dealing with non scalable kernels

- CFD decomposed in 2 sub-domains
- Static distribution:
  - CFD: 3 GPUs
  - Cholesky Factorization: 9 CPUs

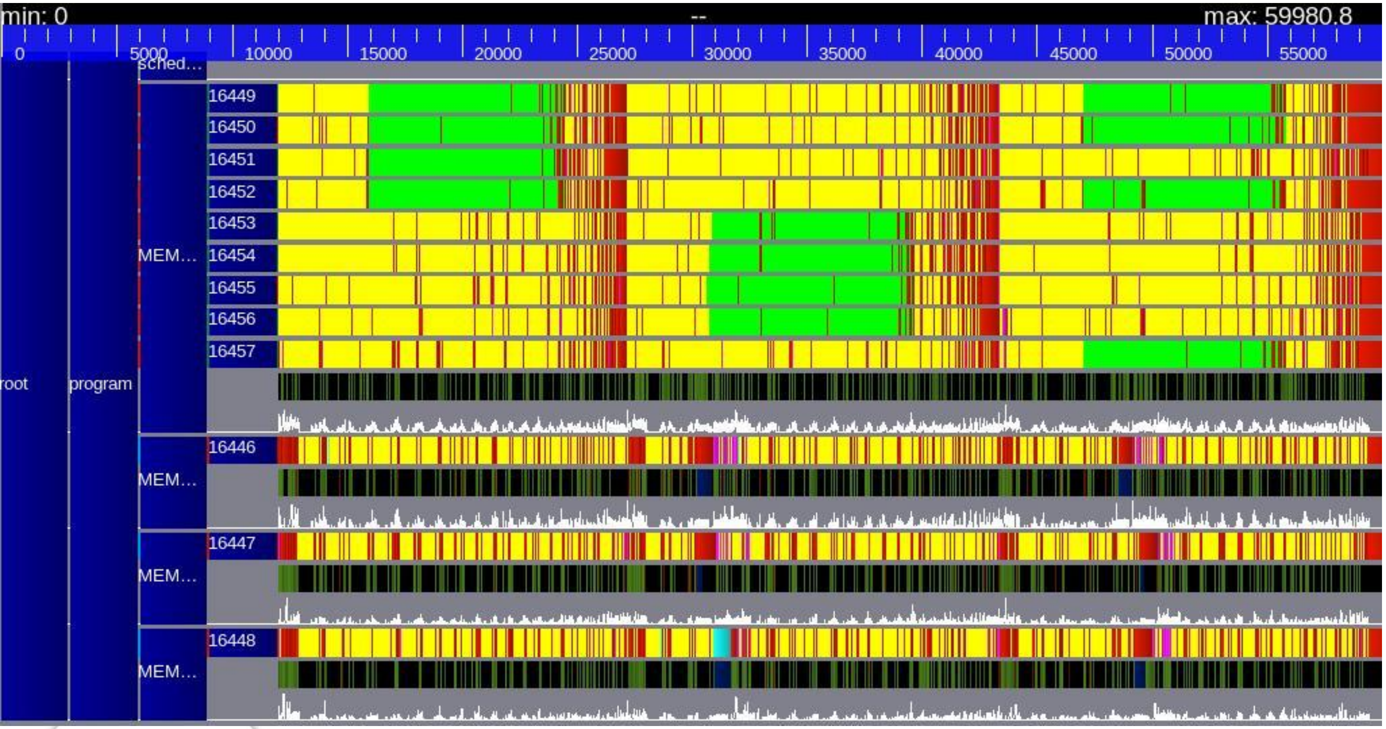
- Hypervisor's intervention:
  - CFD: 2GPUs
  - Cholesky Factorization: 1 GPU & 9 CPUs

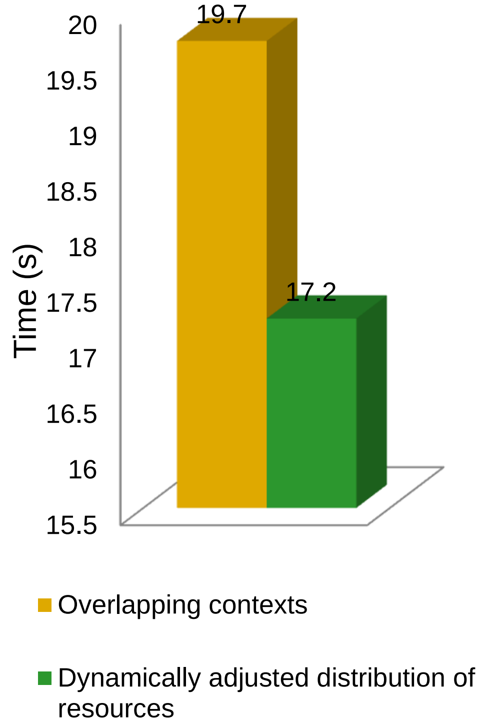


Feedback of the application

When to resize?

- 2 streams of parallel kernels
- 1 of them pops in from time to time
- The hypervisor: assigns some CPUs to the intruder

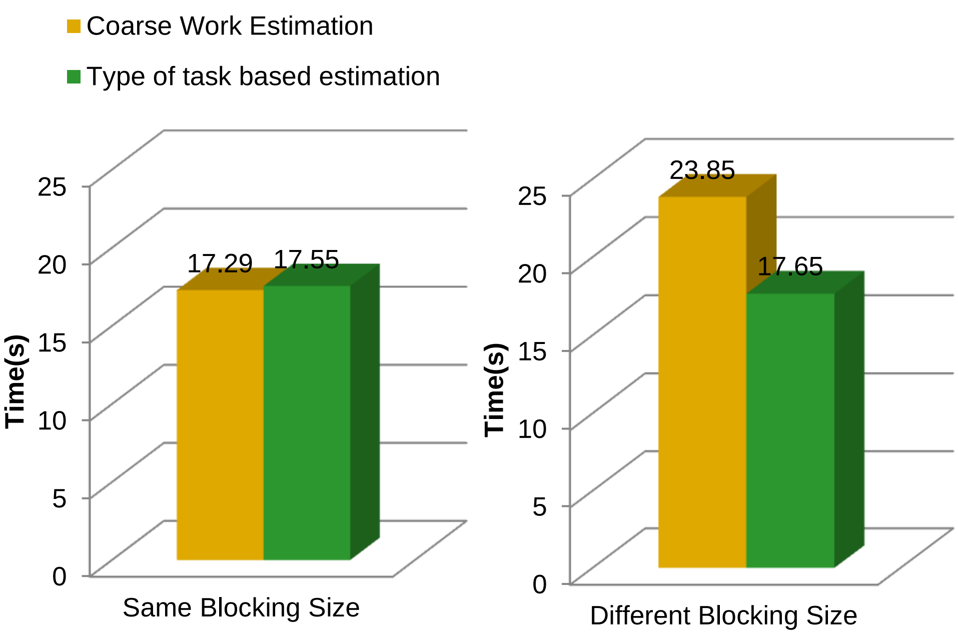




Facing irregular applications

Speed-based resizing policies

- Composing parallel kernels facing different granularities
- Same workload, different number of tasks
- Coarse work estimation:
  - Flops statistics
  - Same granularity for the concurrent parallel kernels
- Type of task based estimation:
  - Number of tasks statistics
  - Considers the type of tasks



1^{st} Factorization: 30k matrix, 960 block size
2^{nd} Factorization: 15k matrix, 960 block size

1^{st} Factorization: 30k matrix, 960 block size
2^{nd} Factorization: 15k matrix, 192 block size

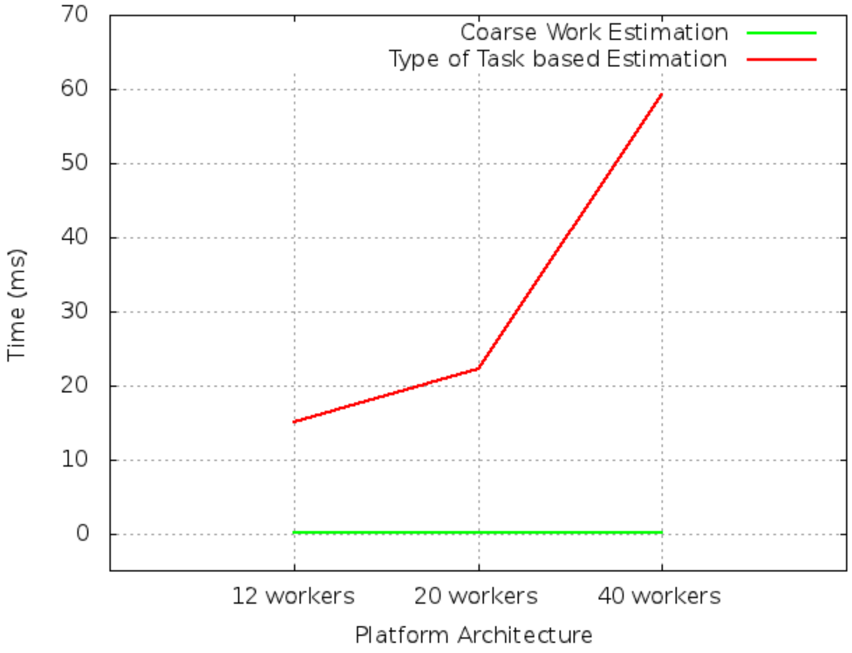
Which one is the best for our application?

Compromise between efficiency and precision?

- It depends …
- Coarse Work estimation
  - Faster
  - Less accurate
  - Irregular application may require it multiple times

- Type of Task based Estimation
  - Higher complexity
  - More precise
  - Useful if the resizing is required a few times

Cost of the policies



Conclusion & Future Work

- Scheduling Contexts allow you use multiple parallel libraries simultaneously
  - Currently implemented in StarPU runtime system
  - A Hypervisor dynamically shrinks / extends contexts

- Further Work
  - New metrics to guide resizing
  - More intelligent sharing of resources (GPUs)

  - And much more!

Potential collaborations

- Extend the techniques to other runtime systems
  - Intel TBB / OS/R / Charm++ / OCR / Unistack
- Experiment with code-coupling applications
  - Great expected benefit from running multiple kernels concurrently
- Experiment with different hypervising strategies
- Any collaboration topic related to (task-based) runtime systems for (hybrid) multicore machines is welcome!