Composing multiple StarPU applications over heterogeneous machines: a supervised approach

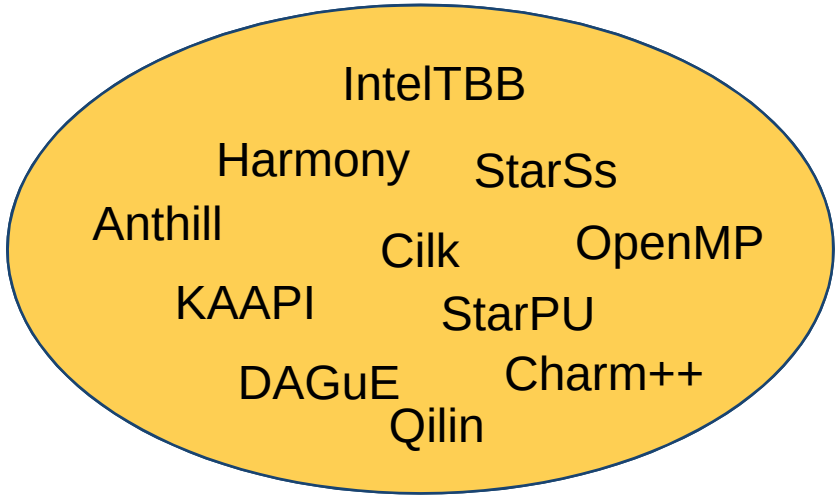
Andra Hugo
With Abdou Guermouche, Pierre-André Wacrenier, Raymond Namyst
Inria, LaBRI, University of Bordeaux

RUNTIME
INRIA Group
INRIA Bordeaux Sud-Ouest

The increasing role of runtime systems

Code reusability

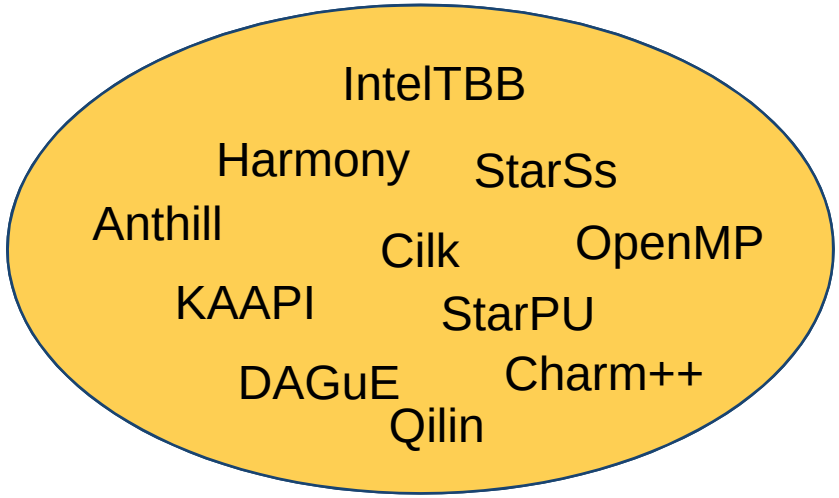
- Many HPC applications rely on specific parallel libraries
  - Linear algebra, FFT, Stencils
- Efficient implementations sitting on top of dynamic runtime systems
  - To deal with hybrid, multicore complex hardware
    - E.g. MKL/OpenMP, MAGMA/StarPU
  - To avoid reinventing the wheel!
- Some application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



=> And the performance of the application ?

Struggle for resources

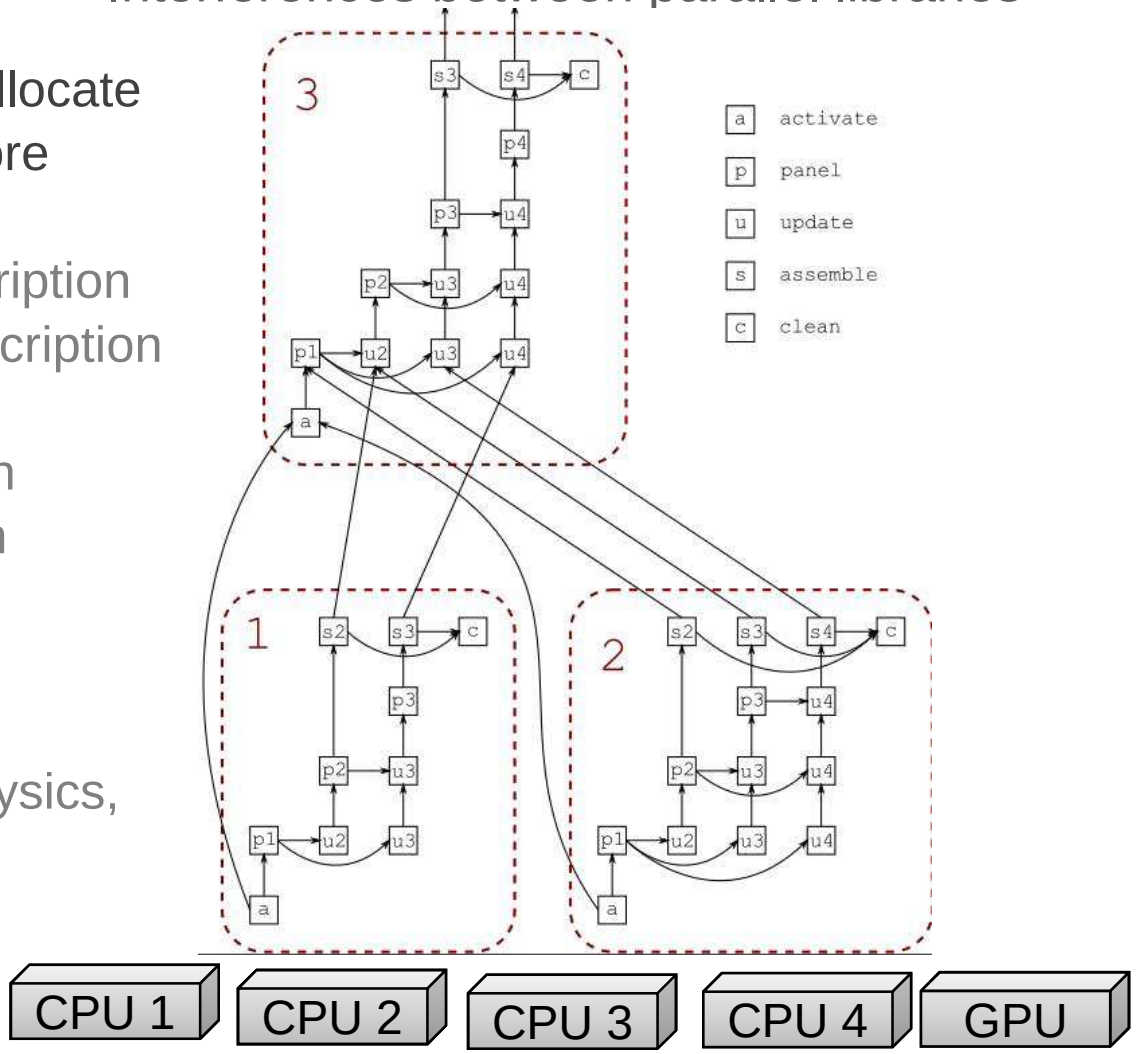
- Parallel libraries typically allocate and bind one thread per core

  Problems:
  - Resource over-subscription
  - Resource under-subscription

  Solutions:
  - Stand-alone allocation
  - Hand-made allocation

- Examples:
  - Sparse direct solvers
  - Code coupling (multi-physics, multi-scale)
  - Etc…

Interferences between parallel libraries



Example: qr_mumps

Struggle for resources

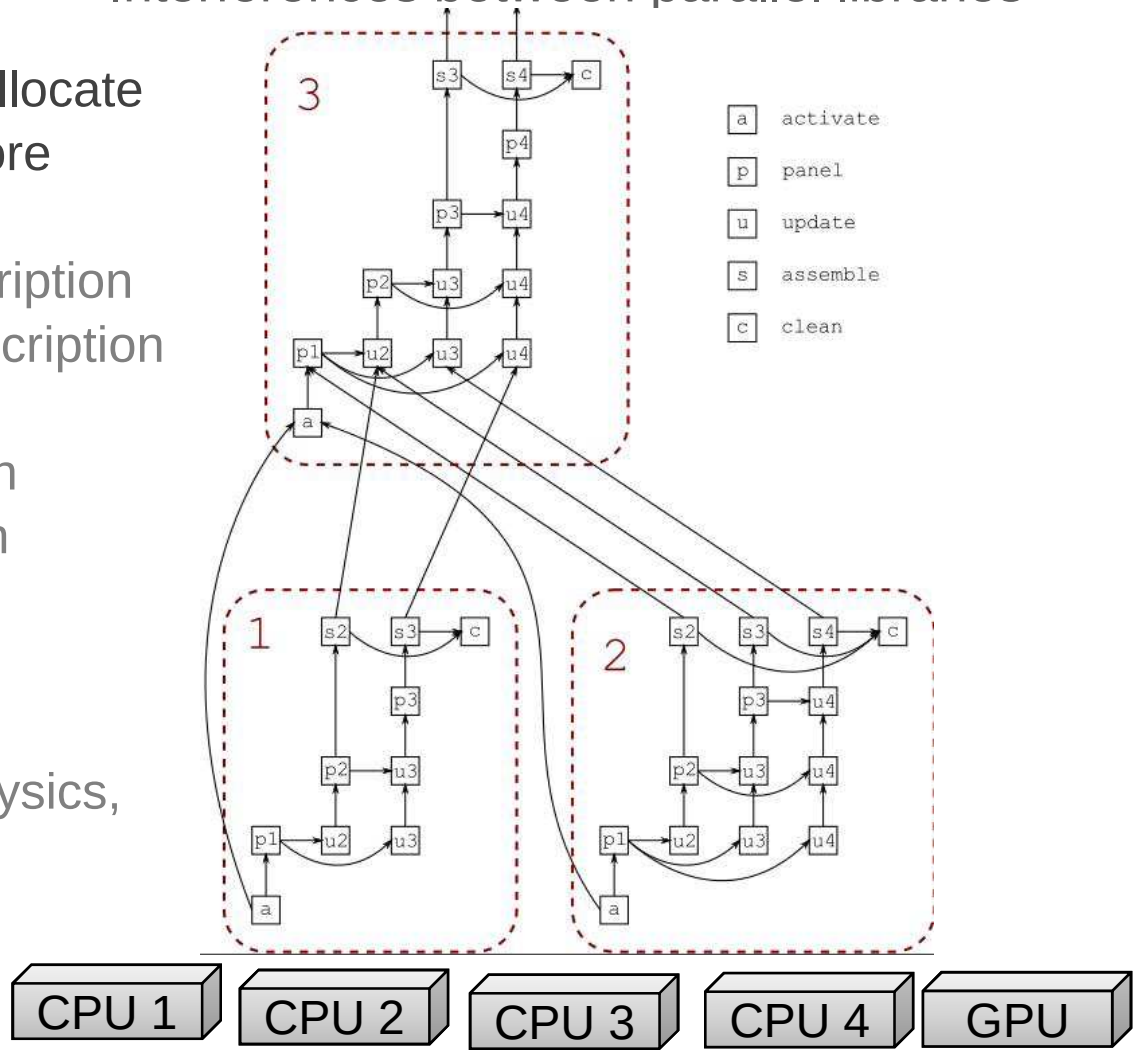
- Parallel libraries typically allocate and bind one thread per core

  Problems:
  - Resource over-subscription
  - Resource under-subscription

  Solutions:
  - Stand-alone allocation
  - Hand-made allocation

- Examples:
  - Sparse direct solvers
  - Code coupling (multi-physics, multi-scale)
  - Etc…

Interferences between parallel libraries

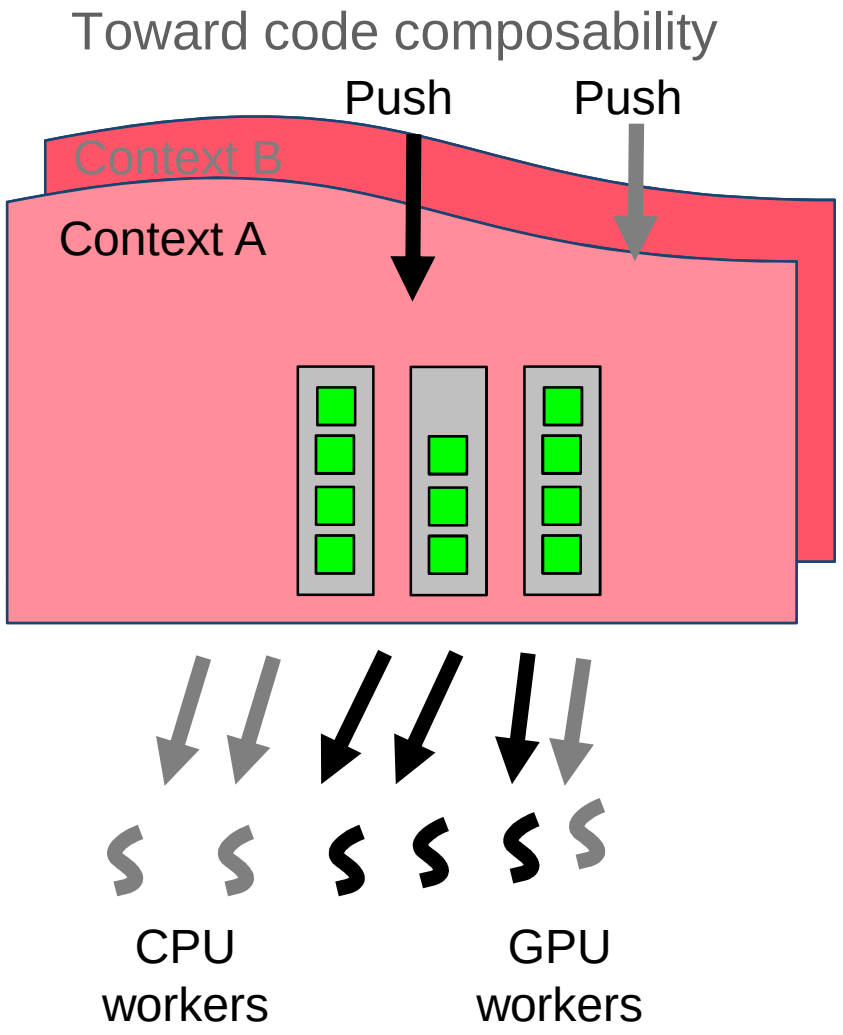


=> Composability problem

Example: qr_mumps

Our approach: Scheduling Contexts

- Isolate concurrent parallel codes
- Similar to lightweight virtual machines



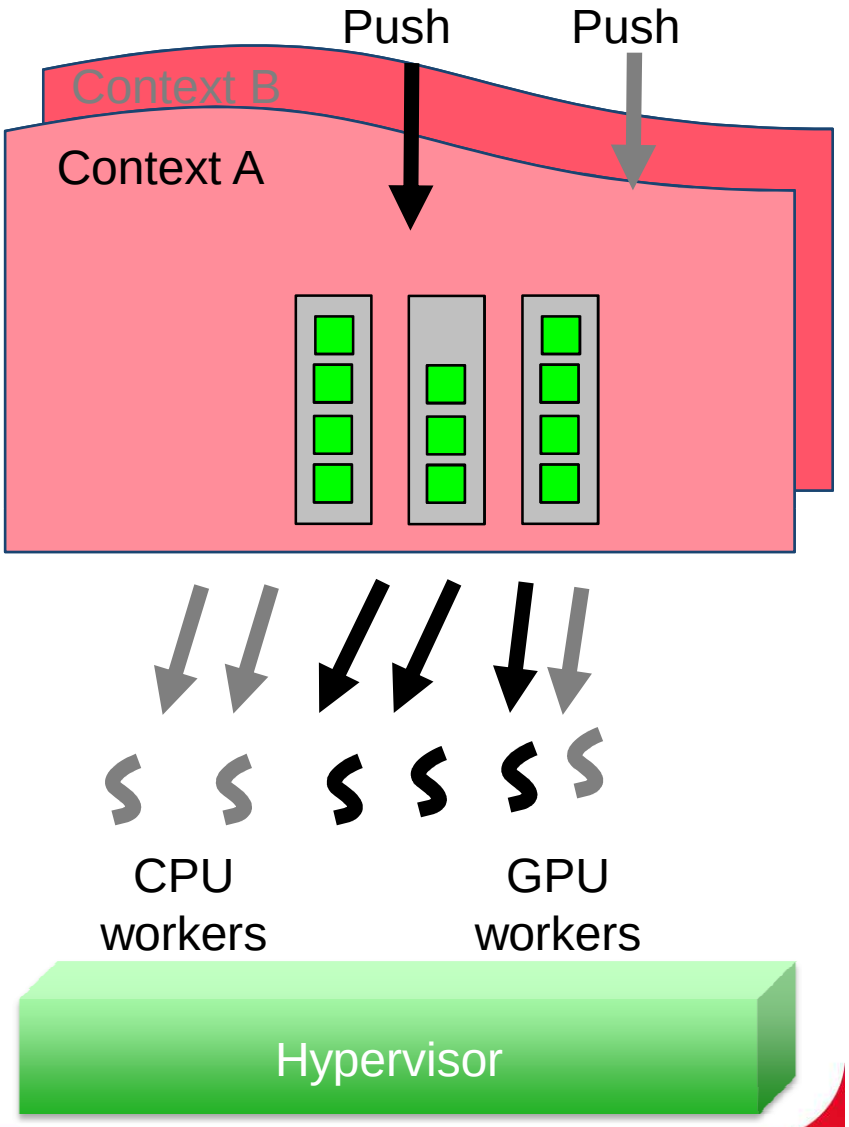
Our approach: Scheduling Contexts

- Isolate concurrent parallel codes
- Similar to lightweight virtual machines

- Contexts may *expand* and *shrink*
  - **Hypervised approach**
    - Resize contexts
    - Share resources

  - Maximize overall throughput

  - Use dynamic feedback both from application and runtime



Tackle the Composability problem

- *Runtime System* to validate our proposal
- *Scheduling contexts* to isolate parallel codes
- *The Hypervisor* to (re)size scheduling contexts

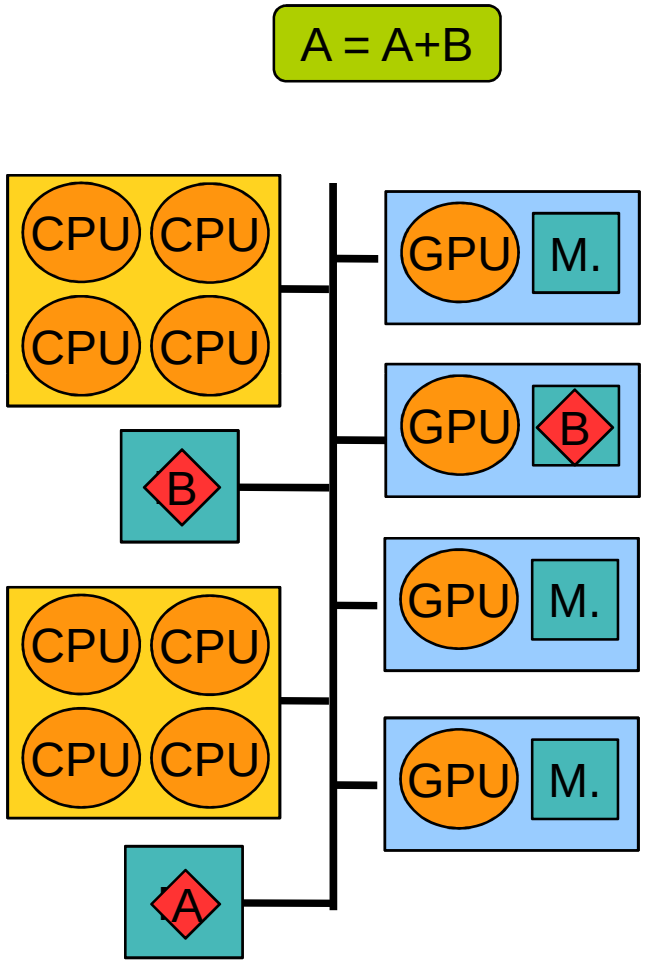
Tackle the Composability problem

- ***Runtime System* to validate our proposal**
- *Scheduling contexts* to isolate parallel codes
- *The Hypervisor* to (re)size scheduling contexts

Using StarPU as an experimental platform

A runtime system for *PU architectures
for studying resource negociation

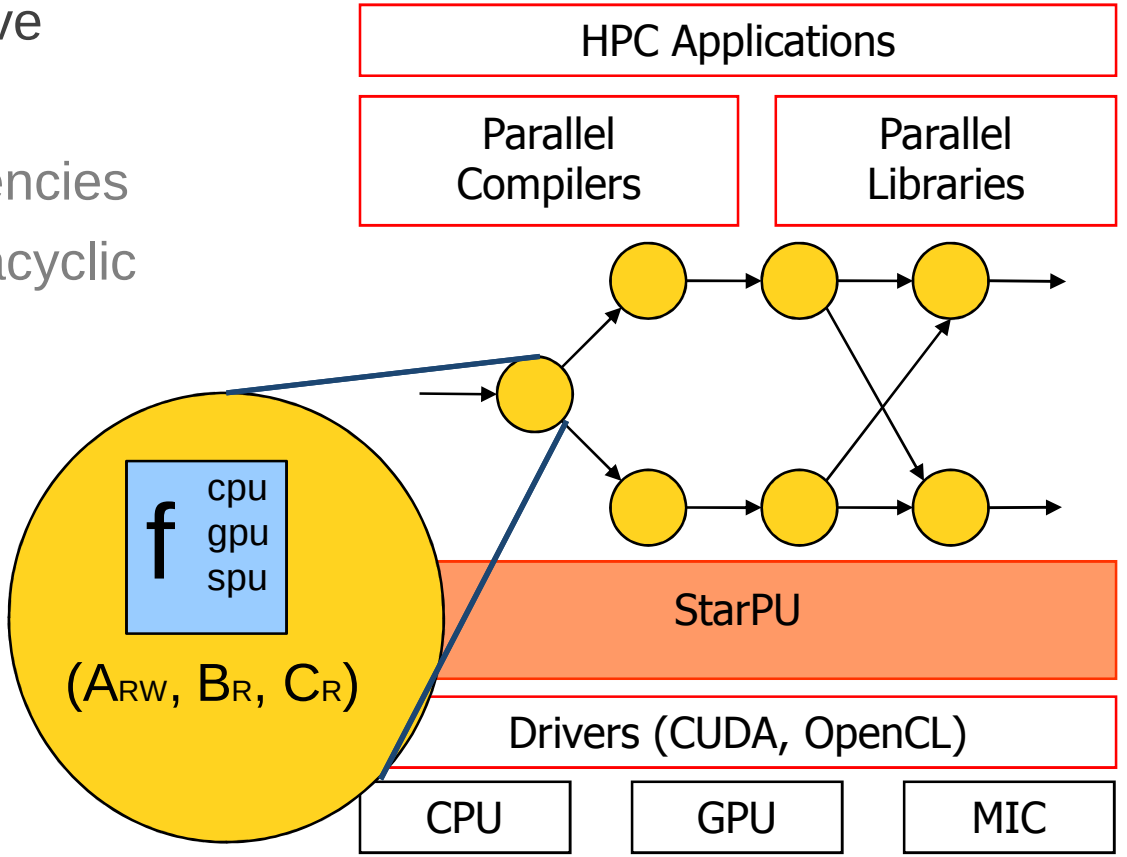
- The StarPU runtime system
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous processing units
  - Avoid unnecessary data transfers between accelerators
    - Software VSM for heterogeneous machines



Overview of StarPU

Maximizing PU occupancy, minimizing data transfers

- Accept tasks that may have multiple implementations
  - Potential inter-dependencies
    - Leads to a directed acyclic graph of tasks
    - Data-flow approach

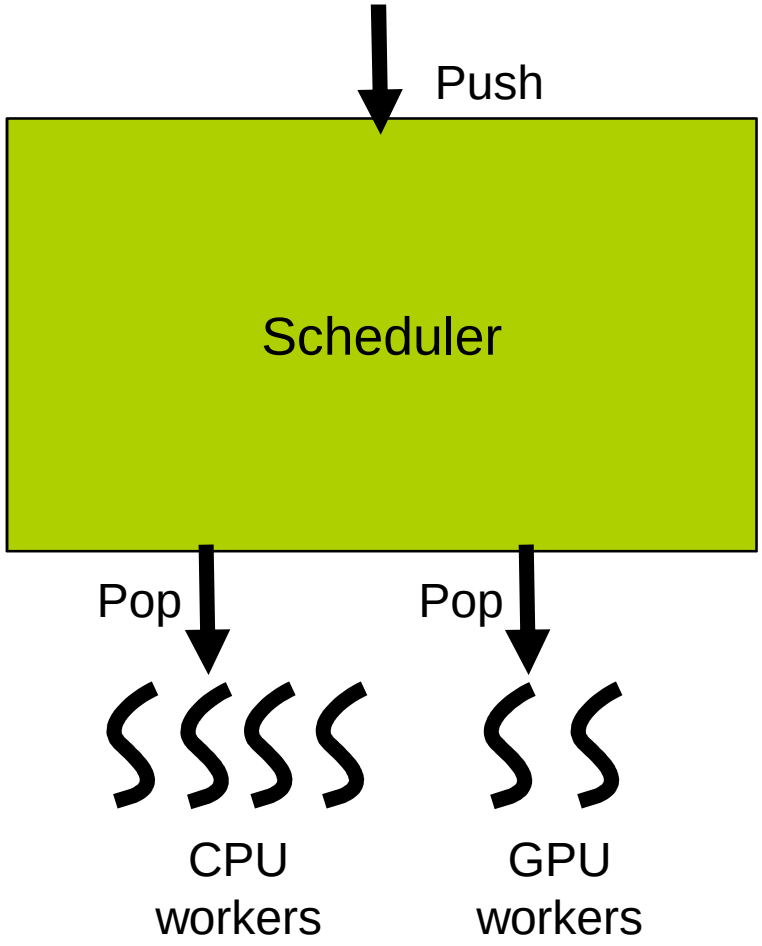


- Open, general purpose scheduling platform
  - Scheduling policies = plugins

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!
- Examples:
  - mct, work stealing, eager, priority



Tackle the Composability problem

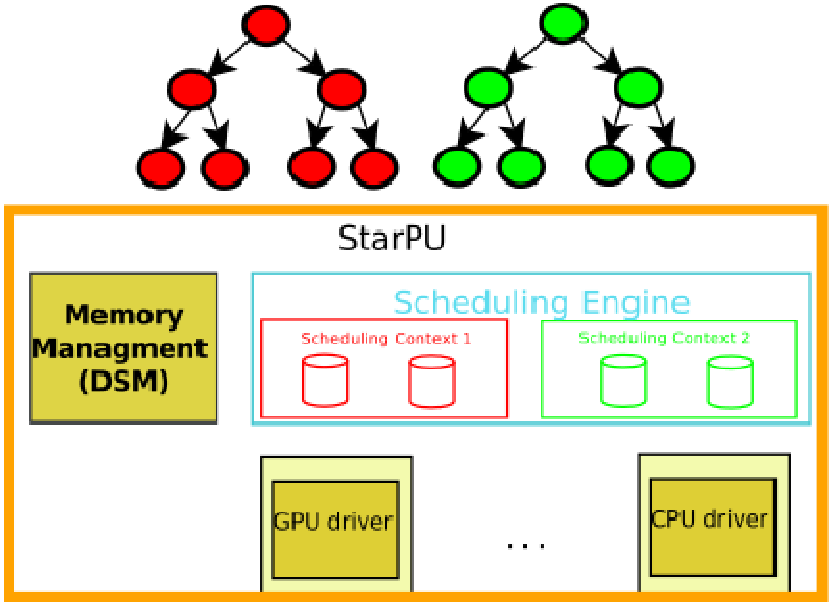
- *Runtime System* to validate our proposal
- ***Scheduling contexts*** **to isolate parallel codes**
- *The Hypervisor* to (re)size scheduling contexts

Scheduling Contexts in StarPU

Extension of StarPU

- “Virtual” StarPU machines
  - Feature their own scheduler
  - Minimize interferences
  - Enforce data locality

- Allocation of resources
  - Explicit:
    - Programmer’s input
  - Supervised:
    - Tips on the number of resources
    - Tips on the number of flops
  - Shared processing units



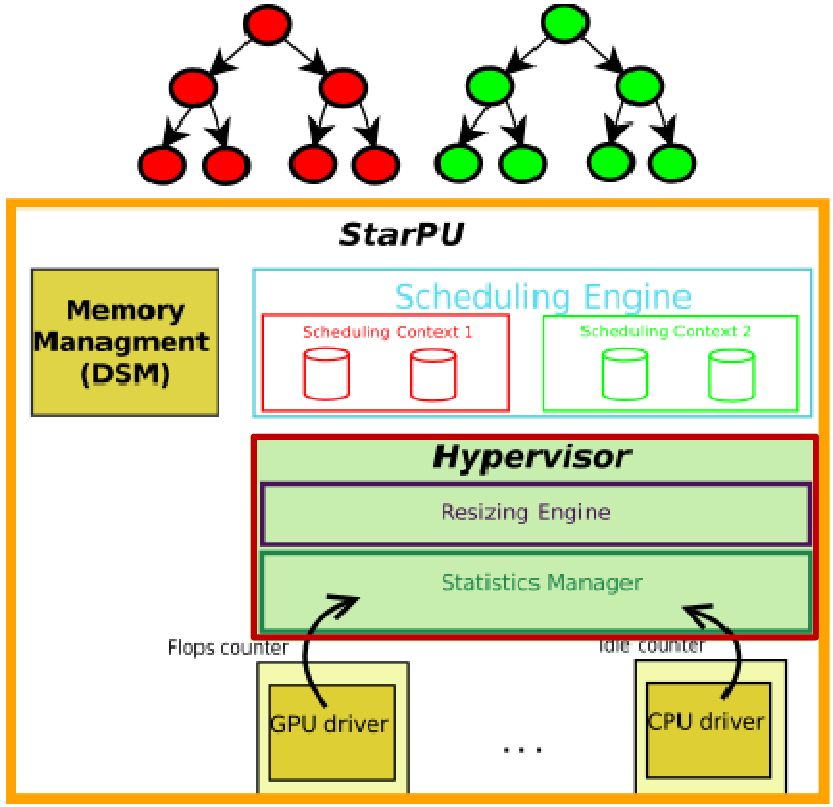
Tackle the Composability problem

- *Runtime System* to validate our proposal
- *Scheduling contexts* to isolate parallel codes
- ***The Hypervisor*** **to (re)size scheduling contexts**

The Hypervisor

What if static dimensioning doesn't work?

- Idea:
  - Monitors scheduling contexts
  - Dynamically resize scheduling contexts
  - Different resizing policies
- Optimization criteria:
  - Maximize the instant speed of the resources/contexts
  - Minimize total execution of the application



The Hypervisor

What if static dimensioning doesn't work?

- When to trigger resizing?
  - The initial configuration deteriorates the performances
  - Different metrics:
    - Idle resources
      - Triggering threshold given by the application
      - Easy to find
    - Speed of the contexts
      - Dependent on the workload of the kernels
      - Compute "right" velocity for each context
      - Outside the "right" interval => wrong behavior
      - Difficult to evaluate

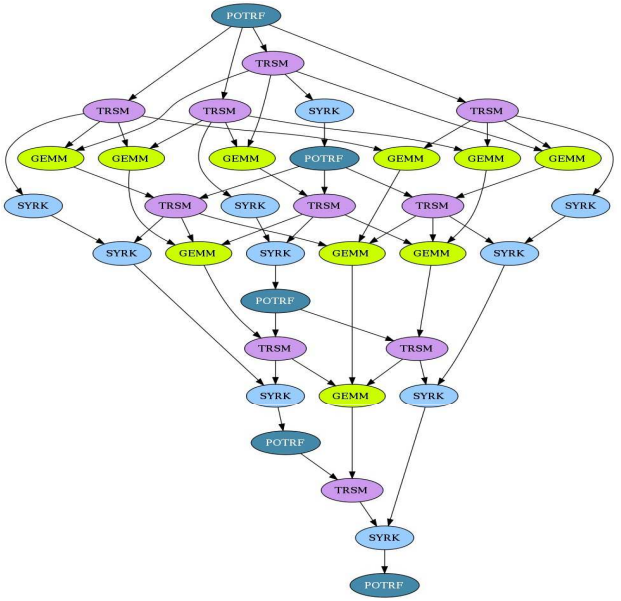
Experimental evaluation

Platform and Application

- **9 CPUs** (two Intel hexacore processors, 3 cores devoted to execute GPU drivers) + **3 GPUs**

- MAGMA Linear Algebra Library
  - StarPU Implementation
  - Cholesky Factorization kernel

- 2 Cholesky factorisations
  - 15k x 15k
  - 30k x 30k



MAGMA – Cholesky Factorization

- Best distribution
  - 1st context (15k x 15k): **9 CPUs**
  - 2nd context (30k x 30k): **3 GPUs**

- Arbitrary distribution
  - 1st context (15k x 15k): **4 CPUs**
  - 2nd context (30k x 30k): **5 CPUs + 3 GPUs**

When to resize ?

Criteria to trigger resizing

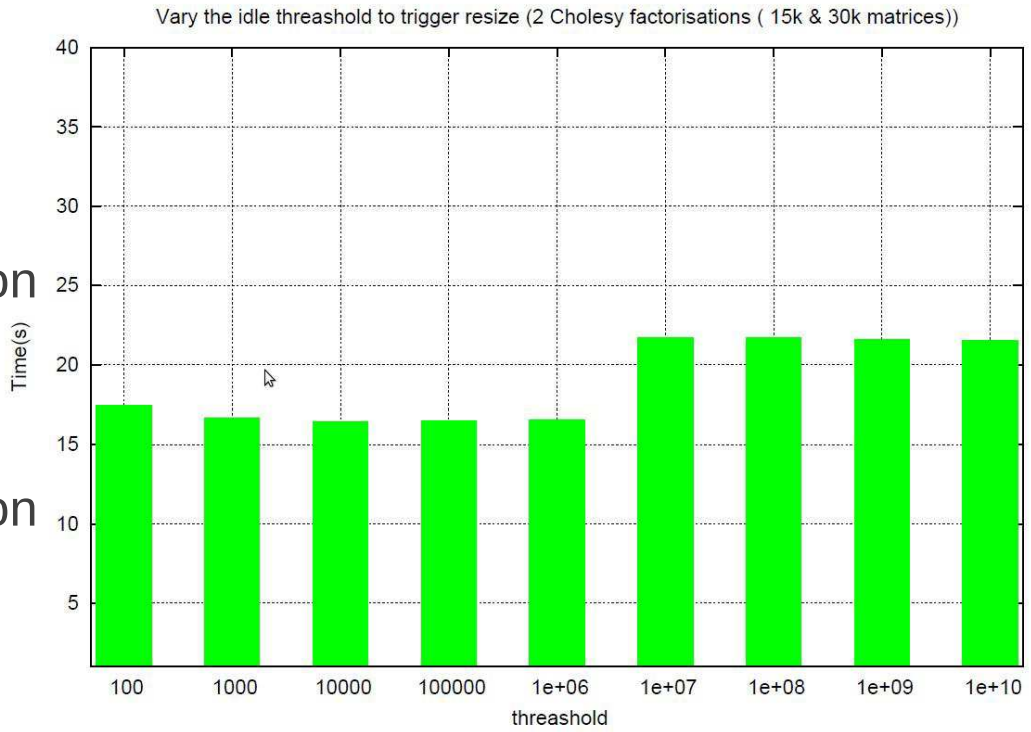
Idle resources

- No tasks to pop from a context

=> Avoid starvation

- **Small** threshold => **often** reevaluation of the distribution

  => ping pong effect
- **Large** threshold => **seldom** reevaluation of the distribution



- Benefits
  - Little input requirements from the application
  - Easy to find a good interval: 10^4 - 10^6

Drawbacks:

- To late for a load-balancing problem

How to resize ?

Maximize the throughput

Focus on the present

- Maximize the instant speed of contexts
- Don't leave anyone behind

Monitor last execution interval

Forecast next execution interval

$$\min\left(t_{min}\right) \text{ subject to } \left(\begin{array}{l} \left(\forall w \in W, \forall c \in C, \frac{1}{s_{w,c}} \cdot \theta_{w,c} \leq t_{min} \cdot x_{w,c}\right) \\ \wedge \left(\forall c \in C, \sum_{w \in W} \theta_{w,c} = \theta_c\right) \\ \wedge \left(\forall w \in W, \forall c \in C, x_{w,c} \in \{0,1\}\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} x_{w,c} = 1\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} \theta_{w,c} > 0\right) \end{array}\right)$$

Maximize the throughput

Focus on the present

- Maximize the instant speed of contexts
- Don't leave anyone behind

Monitor last execution interval

Forecast next execution interval

Speed of a worker in a context

$$\min\left(t_{min}\right) \text{ subject to } \left(\begin{array}{l} \left(\forall w \in W, \forall c \in C, \frac{1}{s_{w,c}} \cdot \theta_{w,c} \leq t_{min} \cdot x_{w,c}\right) \\ \wedge \left(\forall c \in C, \sum_{w \in W} \theta_{w,c} = \theta_c\right) \\ \wedge \left(\forall w \in W, \forall c \in C, x_{w,c} \in \{0,1\}\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} x_{w,c} = 1\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} \theta_{w,c} > 0\right) \end{array}\right)$$

Maximize the throughput

Focus on the present

- Maximize the instant speed of contexts
- Don't leave anyone behind

Monitor last execution interval

Forecast next execution interval

Speed of a worker in a context

$$\min\left(t_{min}\right) \text{ subject to } \left(\begin{array}{l} \left(\forall w \in W, \forall c \in C, \frac{1}{s_{w,c}} \theta_{w,c} \leq t_{min} \cdot x_{w,c}\right) \\ \wedge \left(\forall c \in C, \sum_{w \in W} \theta_{w,c} = \theta_c\right) \\ \wedge \left(\forall w \in W, \forall c \in C, x_{w,c} \in \{0,1\}\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} x_{w,c} = 1\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} \theta_{w,c} > 0\right) \end{array}\right)$$

Flops to be executed by a worker in a context

Maximize the throughput

Focus on the present

- Maximize the instant speed of contexts
- Don't leave anyone behind

Monitor last execution interval

Forecast next execution interval

Speed of a worker in a context

$$\min\left(t_{min}\right) \text{ subject to } \left(\begin{array}{l} \left(\forall w \in W, \forall c \in C, \frac{1}{s_{w,c}} \theta_{w,c} \leq t_{min} \cdot x_{w,c}\right) \\ \wedge \left(\forall c \in C, \sum_{w \in W} \theta_{w,c} = \theta_c\right) \\ \wedge \left(\forall w \in W, \forall c \in C, x_{w,c} \in \{0,1\}\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} x_{w,c} = 1\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} \theta_{w,c} > 0\right) \end{array}\right)$$

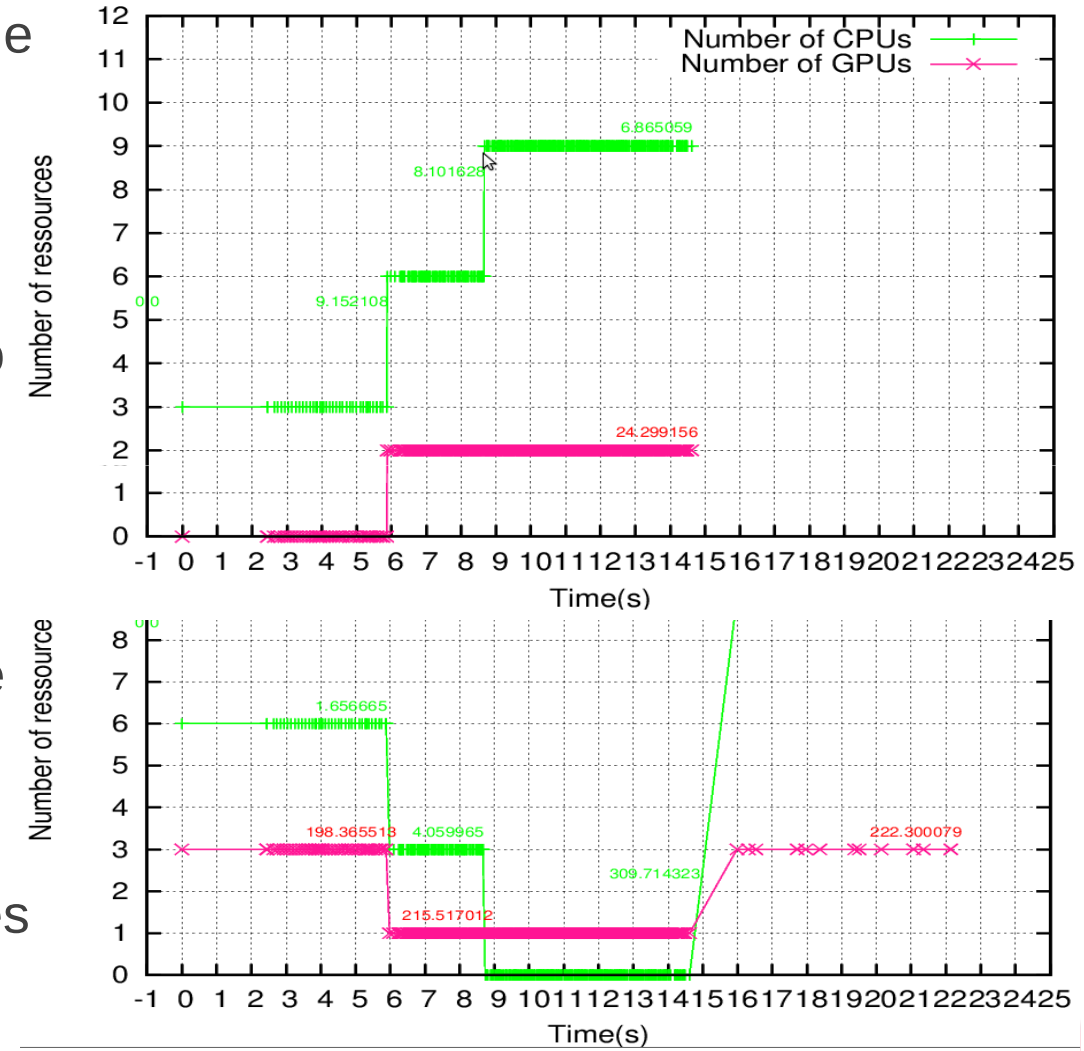
Execution time of the immediate frame

Flops to be executed by a worker in a context

Tracing the decision process

Focus on the present

- Both contexts run at the same speed
- Drawbacks:
  - Force the small kernel to run too fast
  - Important penalty on the big kernel
  - No information about the future
  - No prediction of the performance of resources



Minimize the execution time

Forecast the future

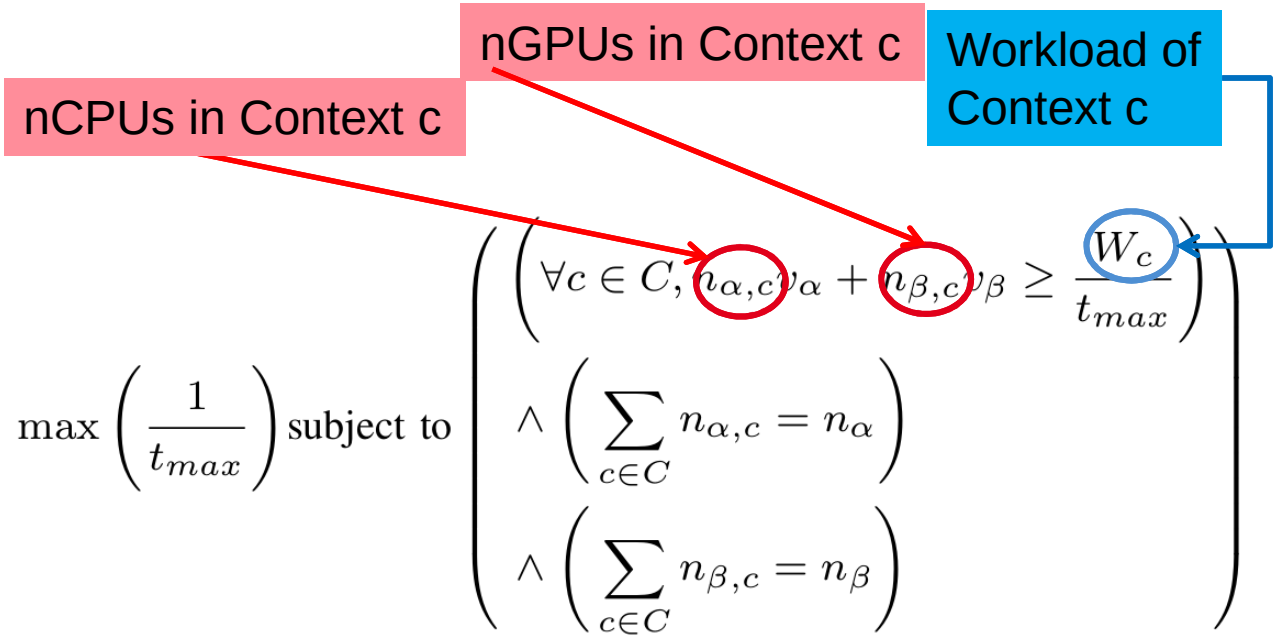
- ***Input***: the workload of the application (number of flops)
- Compute the number of resources of each type of architecture needed by each context
  - How many GPUs/CPUs ?
  - To execute in a minimal amount of time

$$\max\left(\frac{1}{t_{max}}\right) \text{ subject to } \left(\begin{array}{l}\left(\forall c \in C, n_{\alpha,c}v_\alpha + n_{\beta,c}v_\beta \geq \frac{W_c}{t_{max}}\right) \\ \wedge \left(\sum_{c\in C} n_{\alpha,c} = n_\alpha\right) \\ \wedge \left(\sum_{c\in C} n_{\beta,c} = n_\beta\right)\end{array}\right)$$

Minimize the execution time

Forecast the future

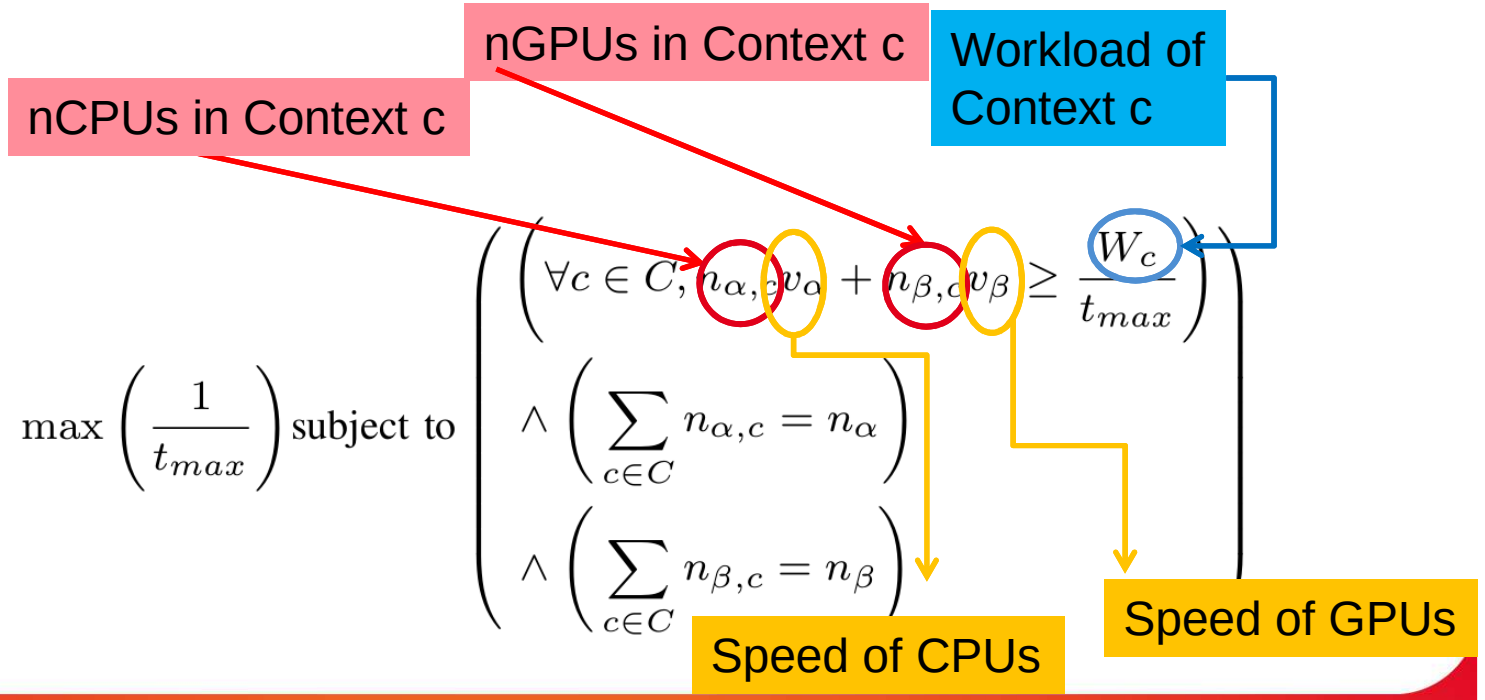
- ***Input***: the workload of the application (number of flops)
- Compute the number of resources of each type of architecture needed by each context
  - How many GPUs/CPUs ?
  - To execute in a minimal amount of time



Minimize the execution time

Forecast the future

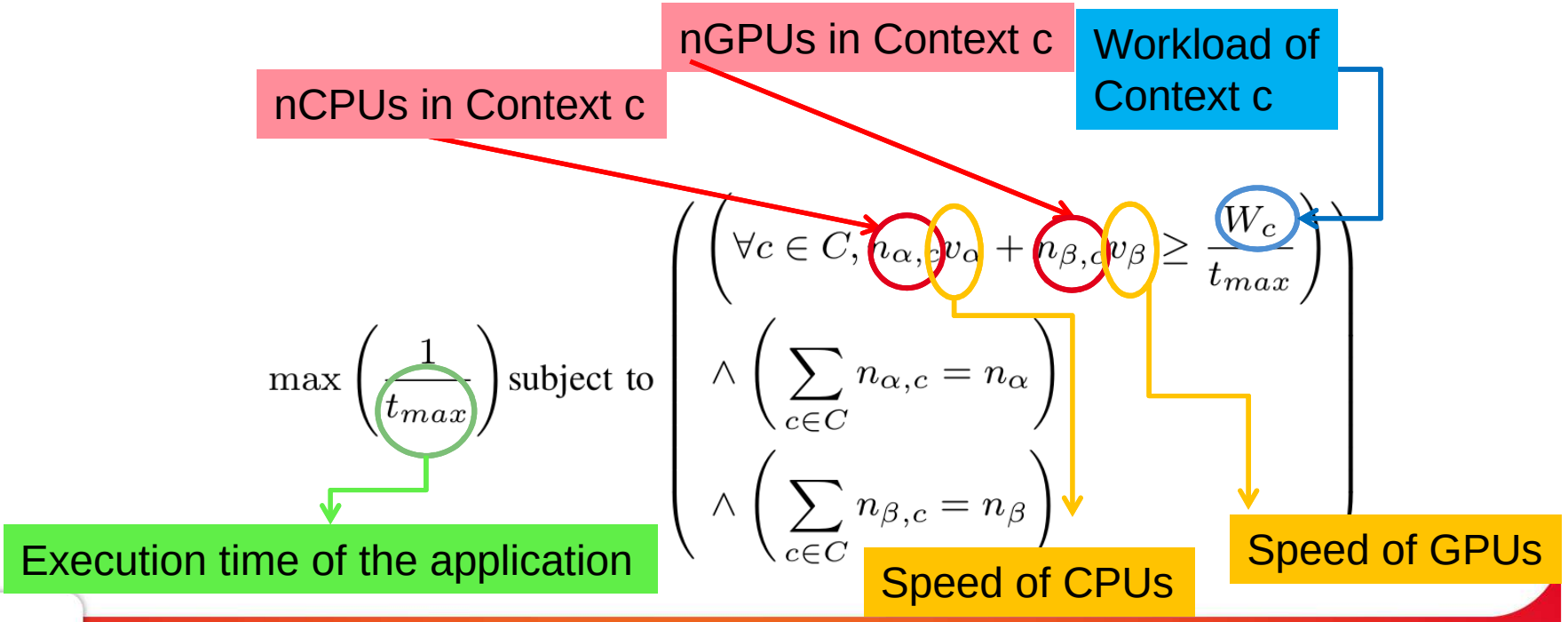
- ***Input***: the workload of the application (number of flops)
- Compute the number of resources of each type of architecture needed by each context
  - How many GPUs/CPUs ?
  - To execute in a minimal amount of time



Minimize the execution time

Forecast the future

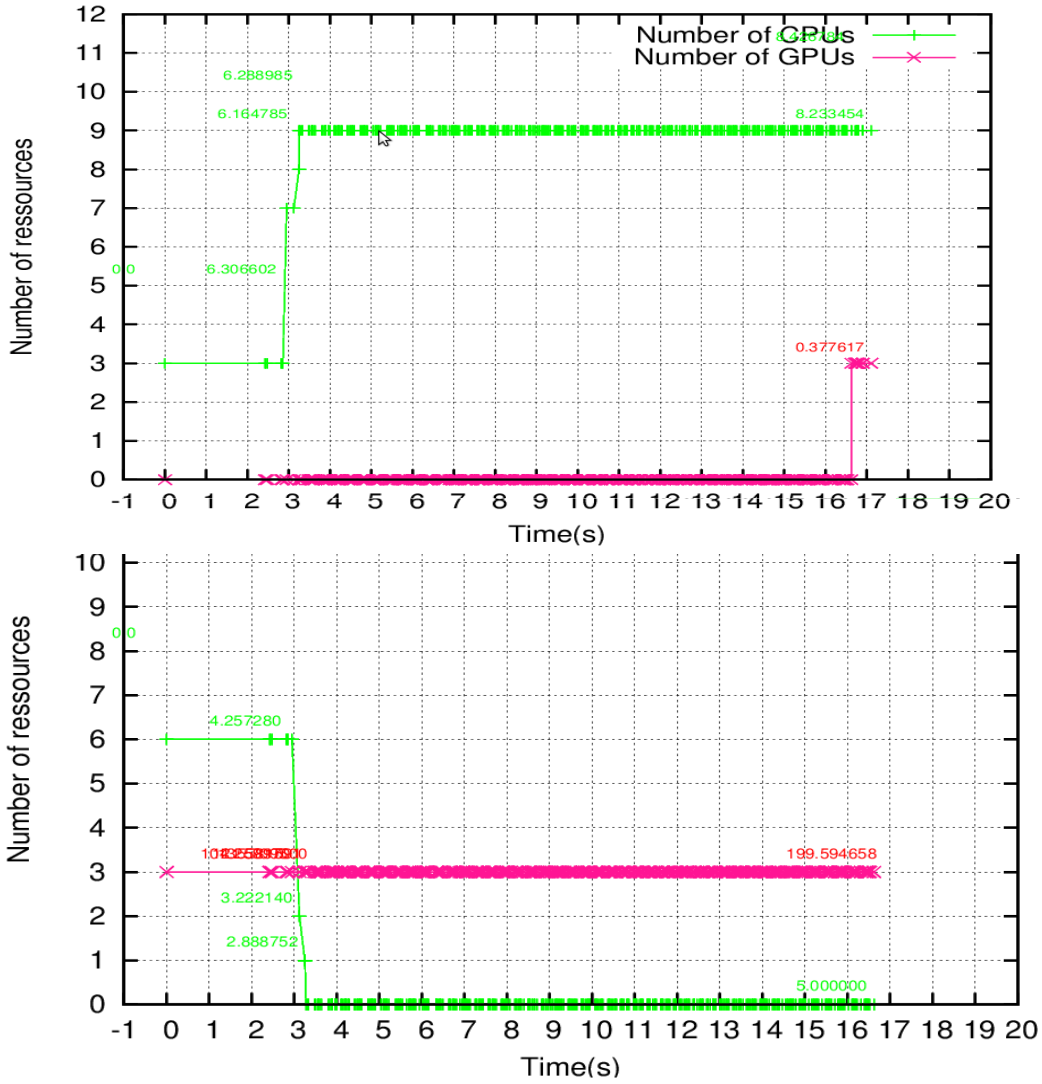
- ***Input***: the workload of the application (number of flops)
- Compute the number of resources of each type of architecture needed by each context
  - How many GPUs/CPUs ?
  - To execute in a minimal amount of time



Tracing the decision process

- Objective:
  - Same termination time
- Monitored speed reinserted in the system
- Resources attracted by the computations where they perform best

Forecast the future



When the programmer is a great wizard

History based performance models

- Knowledge of the execution flow of the application
  - At least a part of if
  - Types of tasks
  - Number of tasks of each type

- Previous calibration of the application
  - Prediction of the execution time of the tasks
  - StarPU system of calibration
  - Scheduling policies based on:
    - Task completion time estimation
    - Data transfer time estimation

Minimize the execution time

Forecast the future in detail

- ***Input***: the workload of the application(number of tasks of each type)
- Calibration information => execution time of each type of kernel
- Scheduling policy – independent decisions

$$\min\left(t_{min}\right) \text{ subject to } \left(\begin{array}{l}\left(\forall w \in W, \forall c \in C, \sum_{t \in T_c} n_{t,w} \cdot d_{t,w} \leq t_{min} \cdot x_{w,c}\right) \\ \wedge \left(\forall c \in C, \forall t \in T_c, \sum_{w \in W} n_{t,w} = n_t\right) \\ \wedge \left(\forall w \in W, \forall c \in C, x_{w,c} \in \{0,1\}\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} x_{w,c} = 1\right)\end{array}\right)$$

Minimize the execution time

Forecast the future in detail

- ***Input***: the workload of the application(number of tasks of each type)
- Calibration information => execution time of each type of kernel
- Scheduling policy – independent decisions

Execution time of task t on worker w

$$\min\left(t_{min}\right) \text{ subject to } \left(\begin{array}{l}\left(\forall w \in W, \forall c \in C, \sum_{t \in T_c} n_{t,w} \cdot d_{t,w} \leq t_{min} \cdot x_{w,c}\right) \\ \wedge \left(\forall c \in C, \forall t \in T_c, \sum_{w \in W} n_{t,w} = n_t\right) \\ \wedge \left(\forall w \in W, \forall c \in C, x_{w,c} \in \{0,1\}\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} x_{w,c} = 1\right)\end{array}\right)$$

Minimize the execution time

Forecast the future in detail

- ***Input***: the workload of the application(number of tasks of each type)
- Calibration information => execution time of each type of kernel
- Scheduling policy – independent decisions

Execution time of task t on worker w

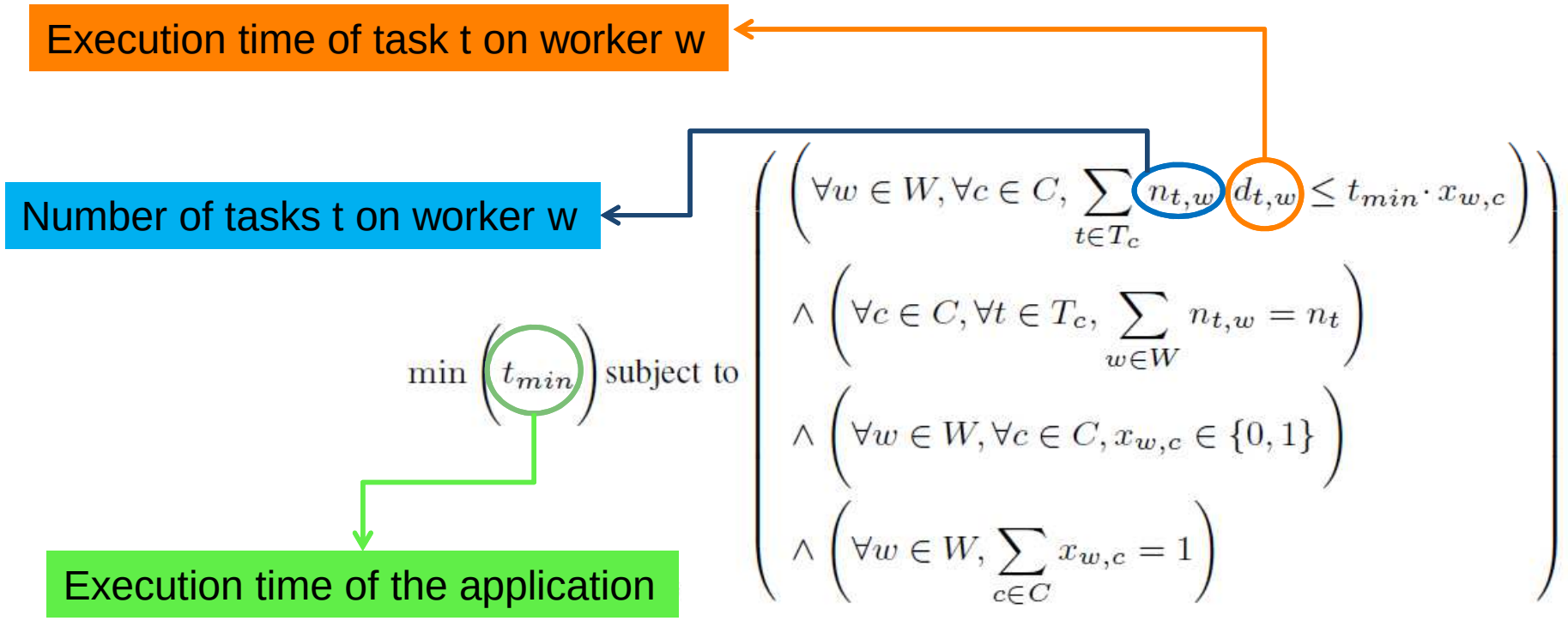
Number of tasks t on worker w

$$\min\left(t_{min}\right) \text{ subject to } \left(\begin{array}{l} \left(\forall w \in W, \forall c \in C, \sum_{t \in T_c} n_{t,w}\, d_{t,w} \leq t_{min} \cdot x_{w,c}\right) \\ \wedge \left(\forall c \in C, \forall t \in T_c, \sum_{w \in W} n_{t,w} = n_t\right) \\ \wedge \left(\forall w \in W, \forall c \in C, x_{w,c} \in \{0,1\}\right) \\ \wedge \left(\forall w \in W, \sum_{c \in C} x_{w,c} = 1\right) \end{array}\right)$$

Minimize the execution time

Forecast the future in detail

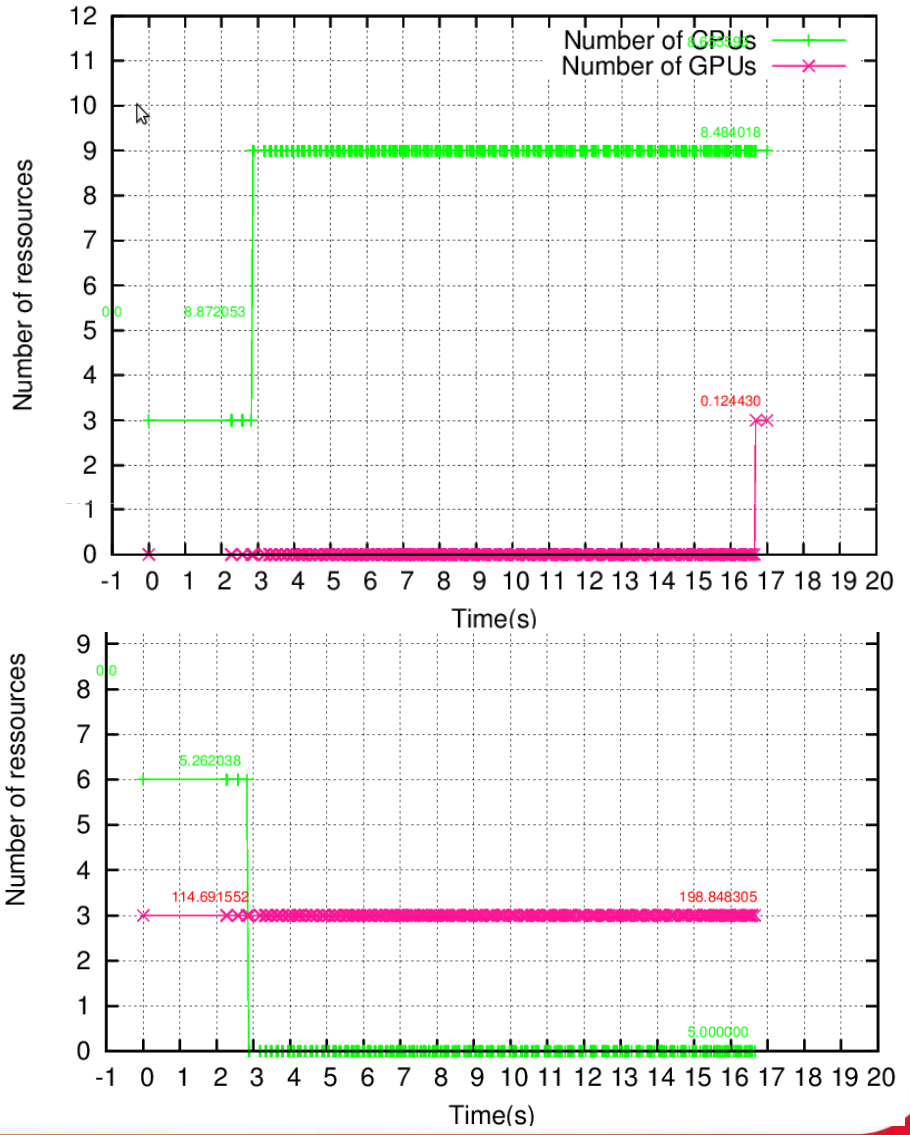
- ***Input***: the workload of the application(number of tasks of each type)
- Calibration information => execution time of each type of kernel
- Scheduling policy – independent decisions



Tracing the decision process

- Objective:
  - Same termination time
- Resources attracted by the type of tasks they execute best
  - CPUs – better executing small Cholesky
  - GPUs – better executing big Cholesky

Forecast the future



Conclusion

- Scheduling Contexts allow using multiple parallel libraries simultaneously
  - Currently implemented in StarPU
  - A Hypervisor dynamically shrinks / extends contexts
- Contexts may be resized whenever we have:
  - Idle resources
  - “Significant” differences of velocity between contexts
    - Estimated speed vs computed speed
    - Acts sooner than idleness based criteria
- Different algorithms to improve the execution of the application
  - Maximize its throughput
  - Minimize its execution time

Future Work

- New metrics to trigger the resizing
  - Burden on the application vs precision of the decision?
- New policies to improve the resizing decision
- More intelligent sharing of resources (GPUs)
- Experiment on real life applications
- Extend scheduling contexts to other parallel environments
- …
- And much more!