Static vs Dynamic Scheduling Strategies

Olivier Beaumont (Inria Bordeaux)

Joint work with Emmanuel Agullo, Lionel Eyraud-Dubois, Abdou Guermouche, Julien Hermann, Suraj Kumar, Thomas Lambert, Loris Marchal, Samuel Thibault,...

Scheduling for Large Scale Systems
Nashville, May 19

Outline

Outline

Background: Evolution of parallel hardware

- Multicore chips are commonplace, with complex memory hierarchies
- Heterogeneity seems to be a solid trend
- Many crucial features that are hard to model
  - Many heterogeneous Processing Units
  - With non-symmetric access to the distributed memory
  - Shared communication resources
  - Shared caches
  - Shared access to storage resources
  - Complicated co-scheduling effects
  - + failures, DVFS
- Scheduling is hard

Background – Scheduling Actual Applications

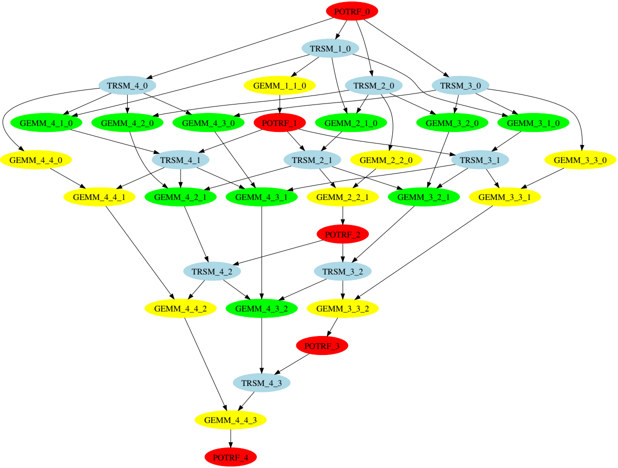
- Start with a code

Algorithm 1 Pseudocode of the tiled Cholesky factorization

```
for k = 0 to n − 1 do
    A[k][k] ← POTRF(A[k][k] RW)
    for i = k + 1 to n − 1 do
        A[i][k] ← TRSM(A[k][k] R, A[i][k] RW)
    end for
    for j = k + 1 to n − 1 do
        A[j][j] ← SYRK(A[j][k] R, A[j][j] RW)
        for i = j + 1 to n − 1 do
            A[i][j] ← GEMM(A[i][k] R, A[j][k] R, A[i][j] RW)
        end for
    end for
end for
```

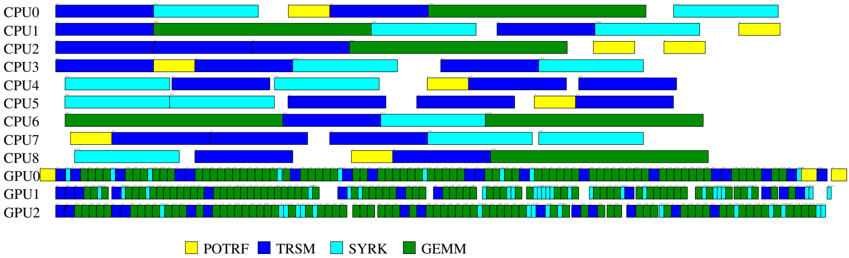
Background – Scheduling Actual Applications

- Start with a code
- Build the task graph



Background – Scheduling Actual Applications

- Start with a code
- Build the task graph
- Find the resource allocation and the schedule



Background – Scheduling Actual Applications

- Start with a code
- Build the task graph
- Find the resource allocation and the schedule
- Limitations (for Cholesky)
  - Heterogeneity makes things harder
  - Simplifying models (co-scheduling effects, bus sharing,...)
  - Needs to be done again for any new architecture !

Background: Can we rely on Static Schedules ?

- Scheduling and resource allocation are hard in general
- and there exist many uncertainties.
- Thus, deciding in advance
  - where to place tasks
  - when/in which order executing them

  can make the system very slow (due to idle times).
- Makes the use of classical static strategies questionable

Solutions to Cope with Uncertainties

- On the theoretical side: robust scheduling
  - Given probability distribution of execution/transfer durations
  - Find (allocation/schedule) that minimizes expected makespan
  - Bad point: makes optimization problems even extremely harder (see Erik's talk) !
- On the practical side: runtime systems
  - Fault Tolerance: checkpointing, replication
  - Scheduling and Load Balancing:
    Mostly dynamic greedy strategies
    (e.g. Hadoop, PaRSEC, StarSs, KAAPI, StarPU)
  - where runtime decisions are based on
    - state of the resources
    - estimations of processing & transfer times
    - add static priorities to choose between ready tasks

Goal in this talk

- Dynamic Strategies:
  - What is the impact of deciding at runtime (myopic vision) ?
- Static Strategies :
  - What is the impact of bad estimations (astigmatic vision) ?
- Combining both ?
  - How to model the behavior of dynamic schedulers ?
    - Balls into Bins games, Power of Two choices, Ordinary Differential Equations
  - Can inject static knowledge into dynamic schedulers ?
    - Affinities between tasks and resources, Specific Dynamic Policies
  - Can inject dynamic adaptations into static schedulers ?
    - Change local ordering, Work Stealing

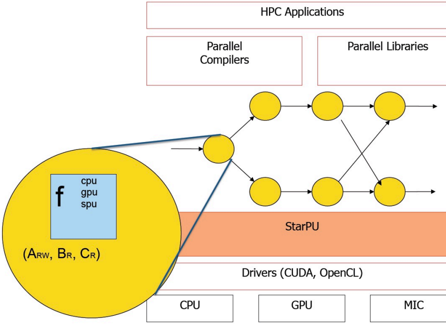
Focus on three basic kernels only

- Map(Reduce) Tasks (with LM and SI)
  - No dependencies, input data already replicated (HDFS, GFS),
  - Focus on the impact of replication.
- Matrix Multiplication (with Thomas Lambert, LED, AG)
  - Tasks are independent but depend on input files
  - Focus on data transfers
- Cholesky Factorization (with Suraj Kumar, EA, LED, AG, JH, ST)
  - 4 different kernels, dependencies, unrelated execution times,
  - But it is easy to overlap communications
  - Focus on the impact of non-uniformity.

Outline

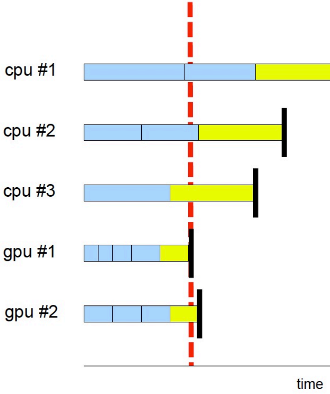
Runtime Systems (StarPU)

- Dynamically
  - Perform Load Balancing
  - React to Hardware Feedback
- Tasks (may) have multiple implementations
  - Upper layers: how data are accessed

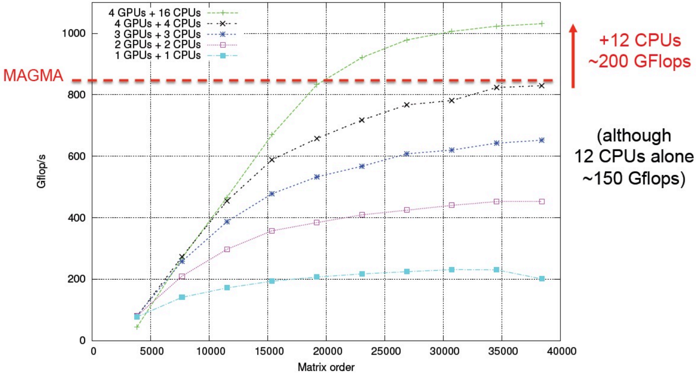


StarPU policy: Basic Idea

- Allocate tasks to resources in advance
  - Overlap Comms with Computations
- When a task becomes ready
  - ie its dependencies are released
- Try all possible resources
- Estimate ending time based on
  - expected running time based on history, parametric cost-model,...
  - expected release time of the resource
  - expected transfer time
- choose the resource with smallest expected completion time



StarPU po



- MAGMA with use of GPUs only (+1 CPU per GPU)
- some kind of super-linear speedup
  - sgeqrt: CPU 9Gflops, GPU 30 Gflops, speedup x3
  - somqr: CPU 9Gflops, GPU 230 Gflops, speedup x27
  - StarPU: 20% of sgeqrt on CPUs, 93% of somqr of GPUs
- **Heterogeneous architectures are cool !**

Outline

Outline

MapReduce basics

- Well known framework for data-processing on parallel clusters
- Popularized by Google, open source implementation: *Apache Hadoop*
- Large data files split into chunks that are
  - scattered on the platform
  - replicated using HDFS for Hadoop
  - there is a time for replication and a time for execution
- Basic Dynamic Scheduling Strategy
  - When a Map slot is available on a processor
  - Choose a local chunk if any
  - Otherwise choose any unprocessed chunk and transfer data

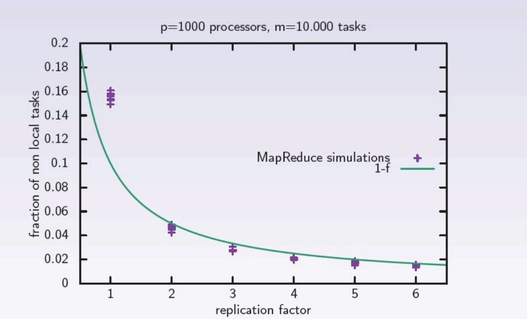
A few remarks

- If all execution times were known (and rather homogeneous)
- and if the application was the only one to execute at runtime
  - the problem would be relatively easy to solve
  - (Ok, still NP Complete but easy to efficiently approximate under realistic assumptions)
- Replication is used
  - for fault tolerance (a little bit)
  - to improve task localilty at runtime (mostly)

Outline

Impact of Replication

- Rationale:
  - the higher is replication ratio (typical value is 3)
  - the higher is the fraction of local tasks (without data transfers)
  - More chances to get executed locally



- Replication is efficient
  - 16% of non local communications without replication
  - $\leq 5\%$ when $r = 2$

Better model: Makespan without replication

- Without replication: each chunk is on a single processor
  - Processor execution time = sum of chunk sizes
  - Similar to the maximum load of a bin in balls-in-bins:
- With homogeneous tasks, when p is close to n,

$$M \sim \frac{\log p}{\log\left(\frac{p \log p}{n}\right)} \text{ whp, } \log p \text{ times the expected value !}$$

- With heterogeneous tasks, see [Raab & Steeger'13], [Berenbrick'08]

Better model: Makespan WITH replication

- Closely related to Balls-In-Bins distribution with r choices:
  - For each ball, select r bins at random
  - Allocate ball to the least loaded bin among them
  - Also known as the power of 2 ($r = 2$) choices
- Balls-In-Bins with multiple choices:
  - For ball B_i, place B_i in least loaded bin with indexes in RC_i

  Rem: load-balancing during the allocation, known weights
- Modified MapReduce:
  - For task T_i, place a copy of T_i on procs with indexes in RC_i
  - When processor P_k becomes idle, execute a local task (if any)

  Rem: load-balancing at runtime, no need to know the weights !

Theorem.

The makespan of Modified MapReduce is equal to the maximum load of Balls-In-Bins with multiple choice

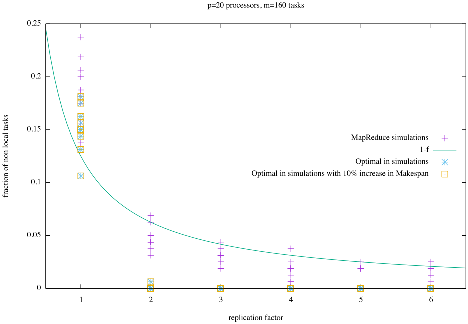
Outline

Impact of Dynamic Scheduling

- Randomized replication
  - Randomization is necessary due to (possibly huge) delay between data allocation and execution
  - is "cheap" (since it is done offline)
  - dramatically reduces the number of communications
- Can we do better at runtime ?
  - once the allocation is known
  - can we pre-compute affinities between tasks and resources
  - for each processor, set of preferred local tasks
  - *ie* tasks it would have run if everything was stable, homogeneous,...
  - run non preferred local tasks if all preferred tasks have been executed

Injecting Static Knowledge in Dynamic Schedulers

- How to compute preferred tasks ?
  - for homogeneous tasks
  - $b-$ matching problem amenable to a flow problem
  - can even be solved in polynomial time.



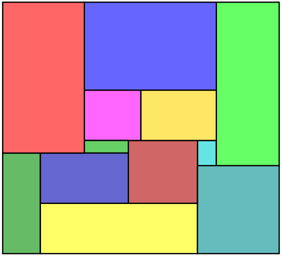
- Conclusion:
  - as soon as $r = 2$, almost all tasks can be executed locally
  - without changing the makespan !

Outline

Outline

Optimization Problem [IPDPS'16] T. Lambert+ [EuroPar'16]

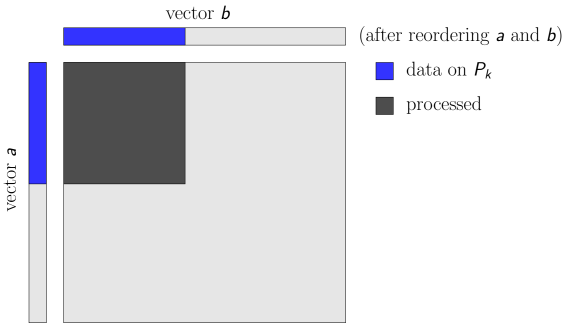
- Problem: partition the square $[0;1] \times [0;1]$ so that
  - $x_k \times y_k = s_k$
  - $\sum_k (x_k + y_k)$ is minimized
  - Lower bound: $2 \sum \sqrt{s_k}$



- NP-Complete, introduced in 2001
- Approximation Ratio $\frac{7}{4}$, then $\frac{5}{4}$ (Nagamochi et al.)
- then $\frac{2}{\sqrt{3}} = 1.15$ (Armin Fügenschuh) under strong conditions (that do not hold true for CPU-GPU platforms)
- We built a $\frac{2}{\sqrt{3}} = 1.15$ approx algorithm without conditions (based on ugly, long and technical proof), can be generalized to cube partitioning.

Outline

StarPU Dynamic Strategy [HPDC'14]

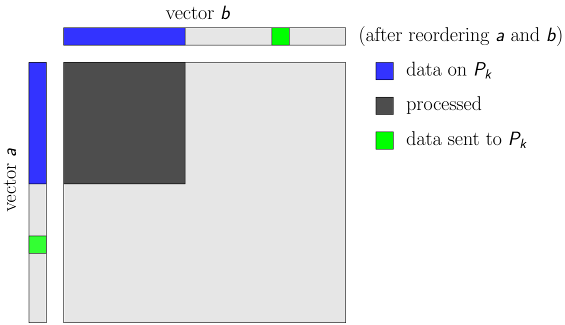


StarPU data-aware strategy DYNAMIC:

Idea : favor tasks for which processors already hold some data.

1. When P_k requests a task, send a new couple (a_i, b_j) to P_k
2. Allocate all available tasks $a_i \times b_{j'}$ (for $b_{j'}$ already on P_k) Allocate all available tasks $a_{i'} \times b_j$ (for $a_{i'}$ already on P_k)

StarPU Dynamic Strategy [HPDC'14]

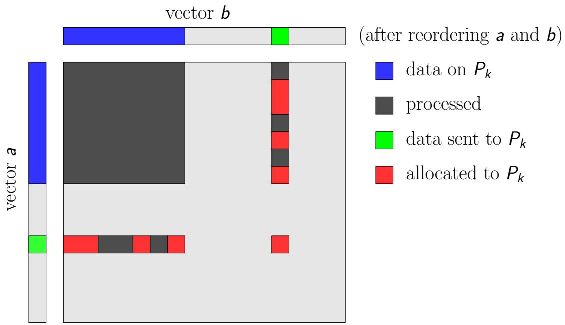


StarPU data-aware strategy DYNAMIC:

Idea : favor tasks for which processors already hold some data.

1. When P_k requests a task, send a new couple (a_i, b_j) to P_k
2. Allocate all available tasks $a_i \times b_{j'}$ (for $b_{j'}$ already on P_k) Allocate all available tasks $a_{i'} \times b_j$ (for $a_{i'}$ already on P_k)

StarPU Dynamic Strategy [HPDC'14]



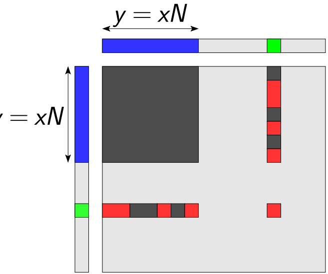
StarPU data-aware strategy DYNAMIC:

Idea : favor tasks for which processors already hold some data.

1. When P_k requests a task, send a new couple (a_i, b_j) to P_k
2. Allocate all available tasks $a_i \times b_{j'}$ (for $b_{j'}$ already on P_k) Allocate all available tasks $a_{i'} \times b_j$ (for $a_{i'}$ already on P_k)

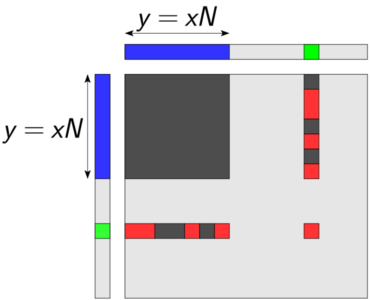
Dynamic2Phases: Analysis (I)

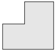
- Assume that the size N of both vectors is large
- Consider a fluid relaxation
- Describe the continuous system using Ordinary Differential Equations



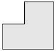
- Ratio $x = y/N$ of elements of a and b on P_k at $t_k(x)$
- Basic step: when this ratio goes from x to $x + \delta x = y/N + \ell/N$
- In ■ : all tasks processed (by P_k or other processors)

Dynamic2Phases: Analysis (II)



- In  : $g_k(x)$ is the fraction of unprocessed tasks (assumed uniformly distributed)
- Time for P_k to compute the red tasks:

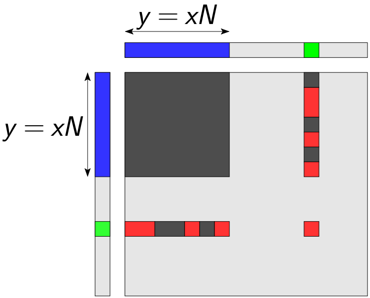
$$\frac{2x\ \delta x\ g_k(x) N^2}{s_k} = t_k(x+\delta x) - t_k(x)$$

- Number of tasks from  computed by other processors during this step: $(t_k(x+\delta x) - t_k(x)) \sum_{i \neq k} s_i$
- Evolution of $g_k(x)$:

$$g_k(x+\delta x) - g_k(x) = g_k(x)\delta x \frac{-2x\alpha_k}{1-x^2} \quad \text{where } \alpha_k = \frac{\sum_{i\neq k} s_i}{s_k}$$

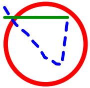
$$\Rightarrow \quad g_k(x) = (1-x^2)^{\alpha_k}$$

Dynamic2Phases: Analysis (III)

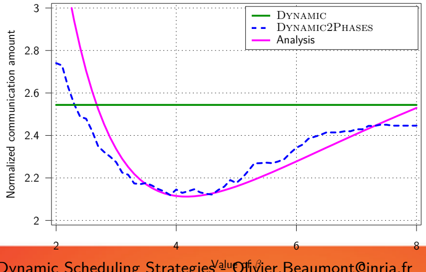


- determine $h_k(x)$, the number of the tasks
  - in the grey area
  - but not processed by P_k
- h_k is solution of a simple ODE.
- that can be used to determine when to switch between strategies
- It works very well in practice !

- Comparison discrete execution vs. continuous analysis:

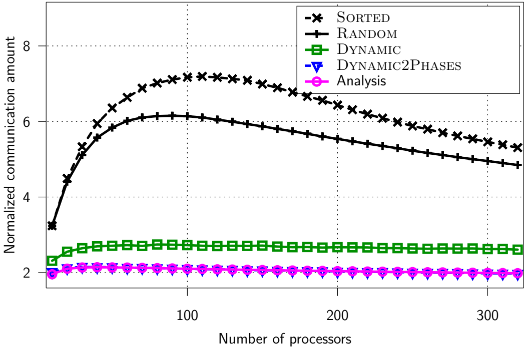


$\Rightarrow$



Add static knowledge in Dynamic Strategies

Comparison with previous heuristics:



- When to switch can easily be computed at runtime
  - and be injected in StarPU scheduling policy
  - $\rightarrow$ add application-dependent knowledge to dynamic policy
- About $2\times$ lower bound on the communication amount
  - Is it good or bad ?

Outline

Comparison of Static and Dynamic [SBACPAD'15]

- Static Strategies are bad...
  - to cope with unexpectedly slow resources
- but they are clever !
  - Worst Case 1.15
  - Average Case < 1.05

- Dynamic Strategies are good
  - to cope with uncertainties
- but they are not so clever...
  - Basic Dynamic: 2.5
  - Dynamic2Phases: 2

Setting for Comparison

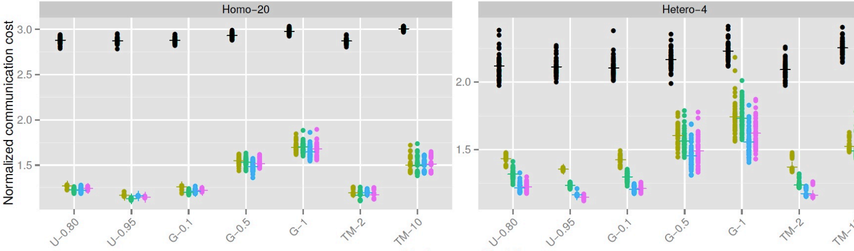
- Wrong processing estimation times given to Static
- Force makespan through Work Stealing (keep resources busy)
- Bad Estimations + Work Stealing induce extra data transfers

Comparison of Static and Dynamic Strategies

- Static but wrong processing estimation times
  - Uniform .8 or .95,
  - Gaussian with 0.1 to 1 variance
  - Makespan (either 1 or 2) or (either 1 or 10)
- Node
  - Quasi homogeneous platform with 20 processors
  - heterogeneous with 10 CPUs + 4 GPUs
- And the winner is...

Comparison of Static and Dynamic Strategies

- Static but wrong processing estimation times
  - Uniform .8 or .95,
  - Gaussian with 0.1 to 1 variance
  - Makespan (either 1 or 2) or (either 1 or 10)
- Node
  - Quasi homogeneous platform with 20 processors
  - heterogeneous with 10 CPUs + 4 GPUs
- And the winner is... **Static !**
- black is dynamic, colored are hybrid
  (static distribution + WS to cope with uncertainties)



Outline

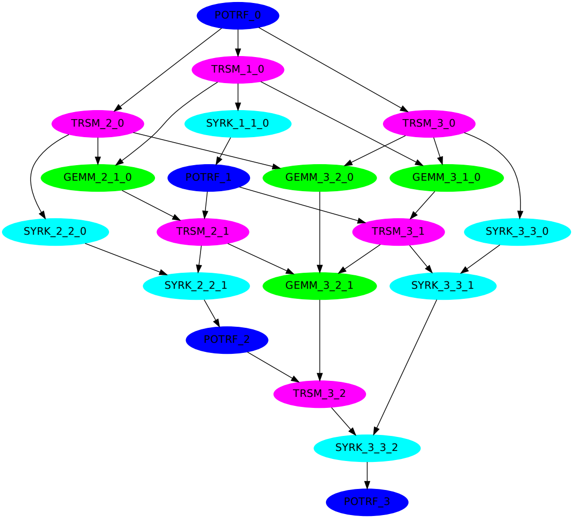
Outline

Task Based Cholesky Factorization

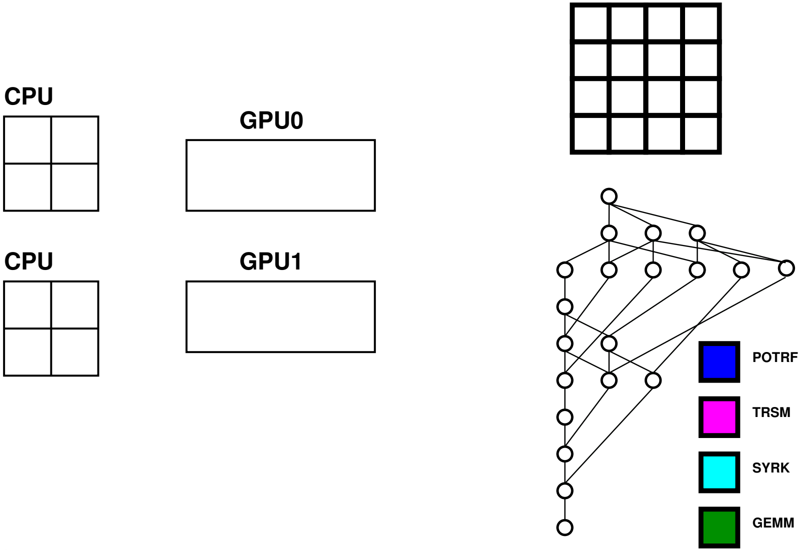
Algorithm 2 Pseudocode of the tiled Cholesky factorization

```
for k = 0 to n − 1 do
    A[k][k] ← POTRF(A[k][k] RW)
    for i = k + 1 to n − 1 do
        A[i][k] ← TRSM(A[k][k] R, A[i][k] RW)
    end for
    for j = k + 1 to n − 1 do
        A[j][j] ← SYRK(A[j][k] R, A[j][j] RW)
        for i = j + 1 to n − 1 do
            A[i][j] ← GEMM(A[i][k] R, A[j][k] R, A[i][j] RW)
        end for
    end for
end for
```

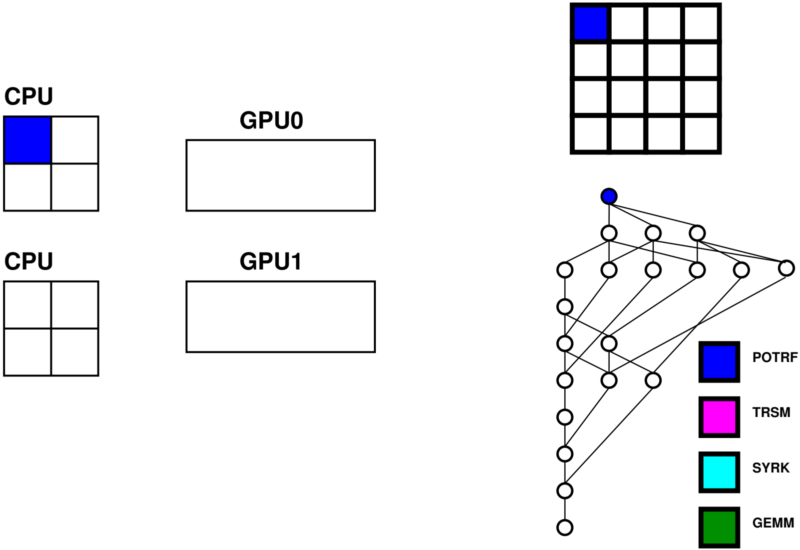
Cholesky task graph



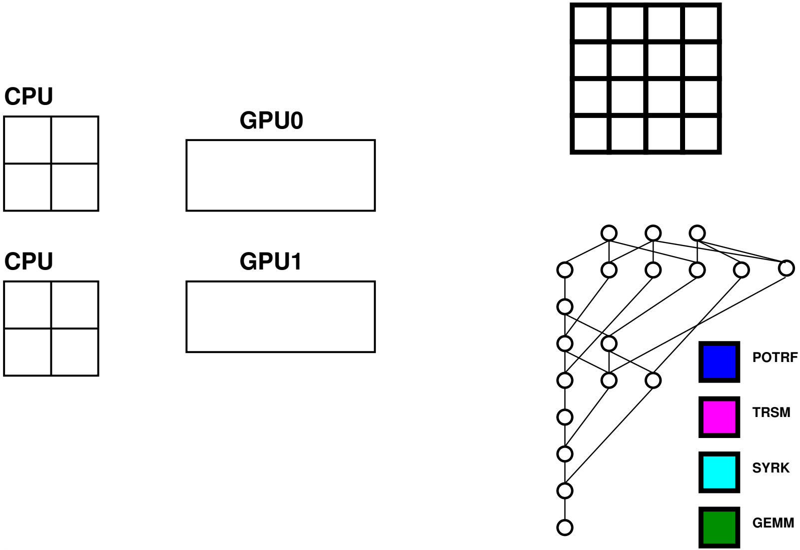
Scheduling of Cholesky tasks with StarPU



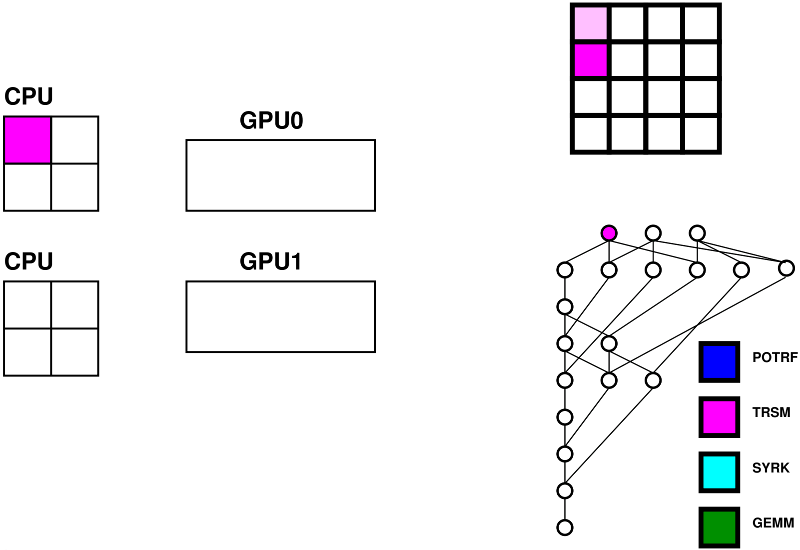
Scheduling of Cholesky tasks with StarPU



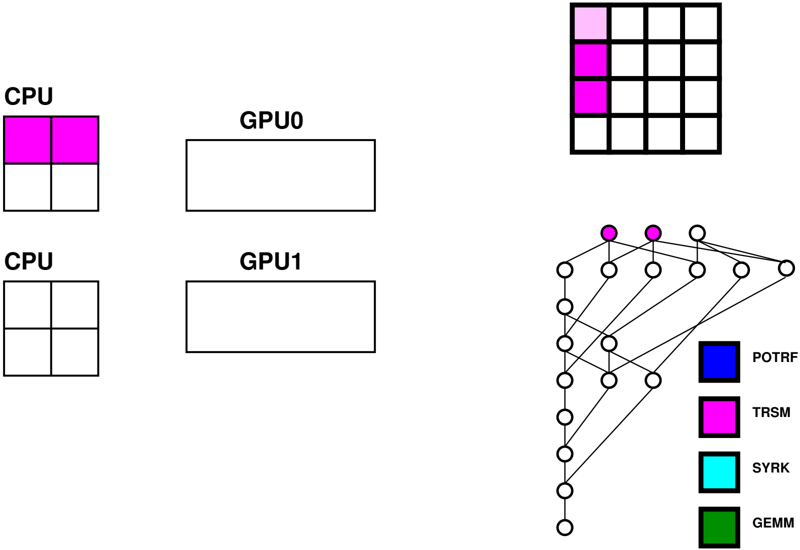
Scheduling of Cholesky tasks with StarPU



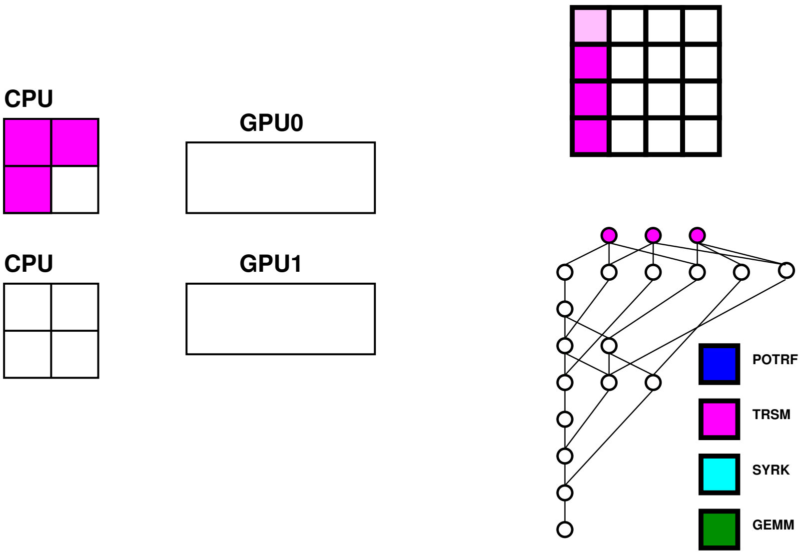
Scheduling of Cholesky tasks with StarPU



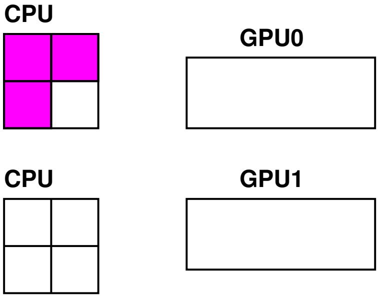
Scheduling of Cholesky tasks with StarPU



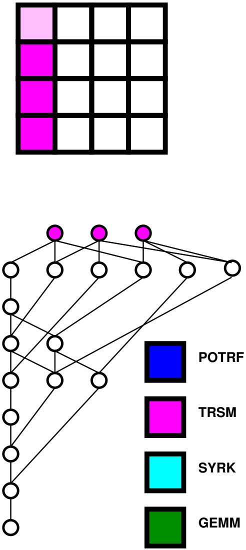
Scheduling of Cholesky tasks with StarPU



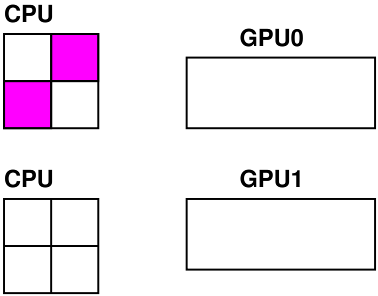
Scheduling of Cholesky tasks with StarPU



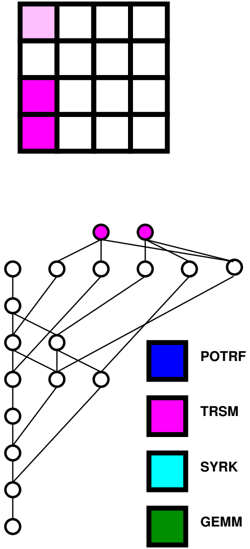
- Handles dependencies



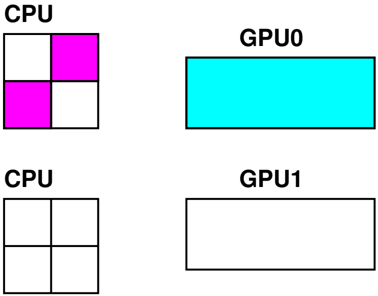
Scheduling of Cholesky tasks with StarPU

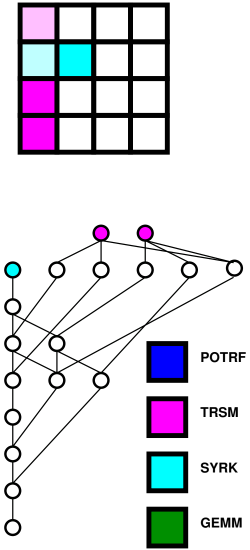


- Handles dependencies



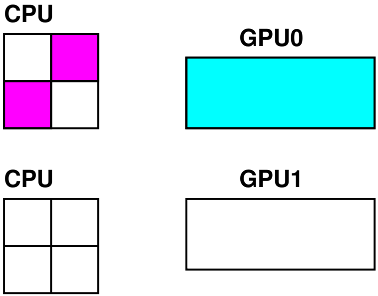
Scheduling of Cholesky tasks with StarPU

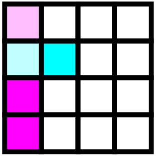


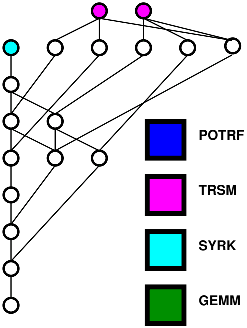


- Handles dependencies

Scheduling of Cholesky tasks with StarPU

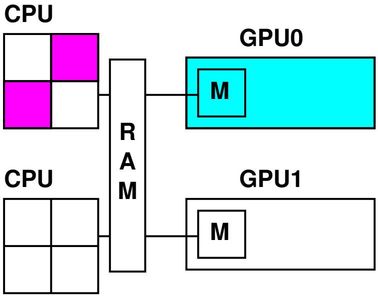


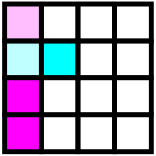


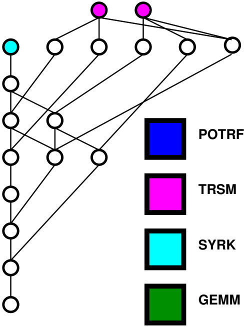


- Handles dependencies
- Handles scheduling (e.g. HEFT)

Scheduling of Cholesky tasks with StarPU

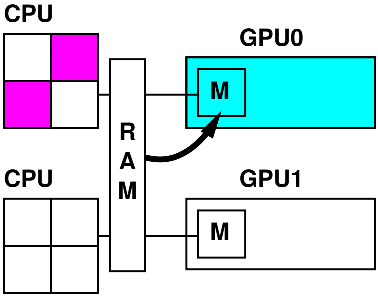


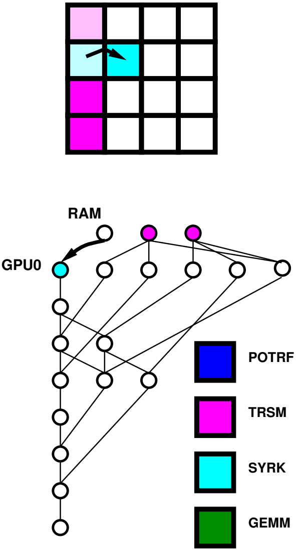




- Handles dependencies
- Handles scheduling (e.g. HEFT)

Scheduling of Cholesky tasks with StarPU





- Handles dependencies
- Handles scheduling (e.g. HEFT)

Machine Information

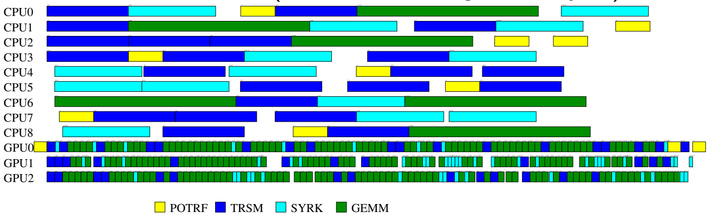
- Heterogeneous settings
  - Hexacore Westmere Intel Xeon X5650 processors (12 CPU cores per node): **9 CPU cores** since 3 are dedicated to data transfers
  - **3** Nvidia Tesla M2070 **GPUs**

POTRF	TRSM	SYRK	GEMM
$\simeq 2.3\times$	$\simeq 11\times$	$\simeq 26\times$	$\simeq 29\times$

Table : GPUs relative performance

Achievable Solutions, $n = 12$

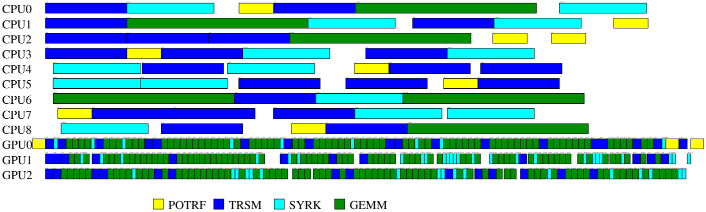
- Best Known Solution (Constraint Prog., 15 days !): 785 GFlops



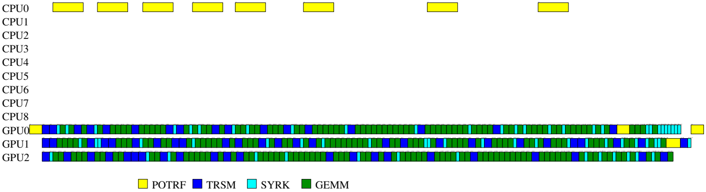
- StarPU schedule: 680 GFlops, not that bad but

Achievable Solutions, $n = 12$

- Best Known Solution (Constraint Prog., 15 days !): 785 GFlops



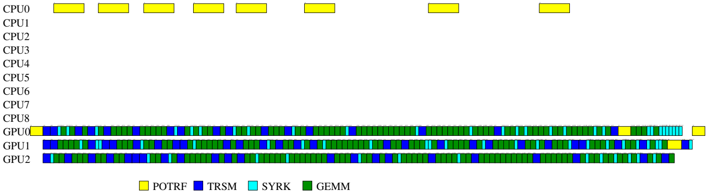
- StarPU schedule: 680 GFlops, not that bad but



- very disappointing...
  but enough to achieve pseudo-super-linear speedup !

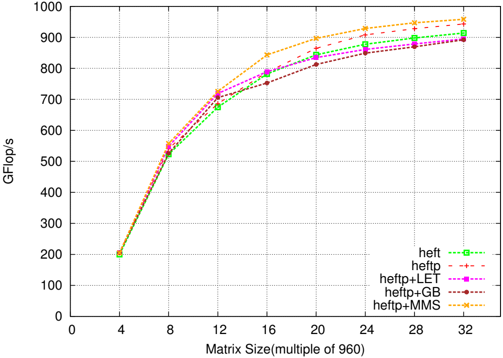
Outline

Improving StarPU scheduler (I) [IPDPS'16, S. Kumar]



- StarPU is too cautious
  - It fails to allocate enough tasks on a CPU
  - even though it would not have induced idle time on a GPU
- Improvement over default policy
  - Pseudo-Allocate next task on CPU
    - Simulate StarPU until the ending time of the task
    - using SimGRID simulator
  - Reallocate on a GPU iff it induces idle time on GPU
- and several variants.

Improving StarPU scheduler (II)



- Variants: how far is it worth to simulate ?
- for $n = 12$, $680 \rightarrow 725$ (best 785)
- much more expensive than default policy, but
    - Dedicate one CPU to compute the schedule !

Acceleration Ratio-based Policies (I)

POTRF	TRSM	SYRK	GEMM
$\simeq 2.3\times$	$\simeq 11\times$	$\simeq 26\times$	$\simeq 29\times$

Table : GPUs relative performance

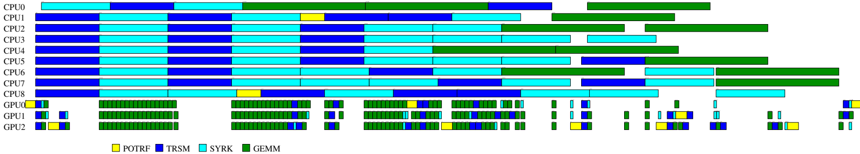
- GPUs should prefer GEMMs to SYRKs to TRSMs to POTRFs
- CPUs should prefer POTRFs to TRSMs to SYRKs to GEMMs
- implement this through priority queues

Acceleration Ratio-based Policies (I)

POTRF	TRSM	SYRK	GEMM
$\simeq 2.3\times$	$\simeq 11\times$	$\simeq 26\times$	$\simeq 29\times$

Table : GPUs relative performance

- GPUs should prefer GEMMs to SYRKs to TRSMs to POTRFs
- CPUs should prefer POTRFs to TRSMs to SYRKs to GEMMs
- implement this through priority queues



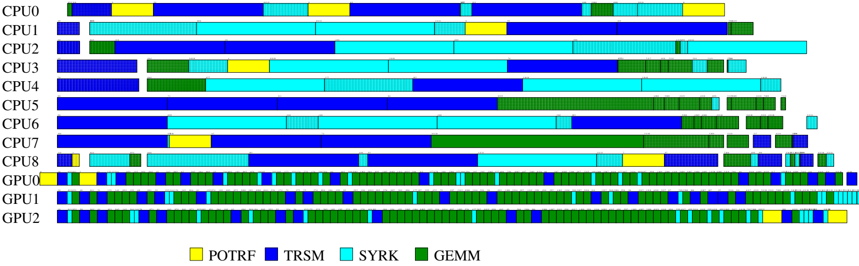
- works poorly :440 Gflops vs Best 785 GFlops:
  GPUs wait for tasks performed by CPUs

Acceleration Ratio-based Policies (II)

- GPUs should prefer GEMMs to SYRKs to TRSMs to POTRFs
- CPUs should prefer POTRFs to TRSMs to SYRKs to GEMMs
- merge some queues + allow task spoliation (if GPU gets idle)

Acceleration Ratio-based Policies (II)

- GPUs should prefer GEMMs to SYRKs to TRSMs to POTRFs
- CPUs should prefer POTRFs to TRSMs to SYRKs to GEMMs
- merge some queues + allow task spoliation (if GPU gets idle)



- works fine: 750 (best known solution is 785)

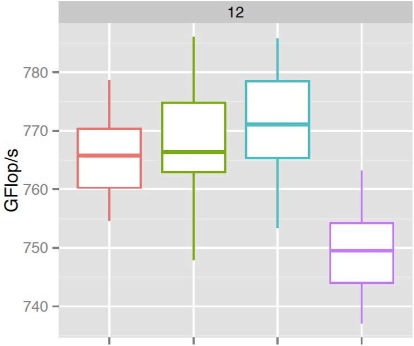
Outline

Static vs Dynamic Strategies

- To do a fair comparison, let us add uncertainties
- task completion times belong to $[90\%, 110\%] \times$ expected time
- In Static Schedules, allow GPUs to perform spoliation
- And the winner is:

Static vs Dynamic Strategies

- To do a fair comparison, let us add uncertainties
- task completion times belong to $[90\%, 110\%] \times$ expected time
- In Static Schedules, allow GPUs to perform spoliation
- And the winner is: **Static Again !**



- red: Pure Static
- green: Hybrid (Static + GEMM Spoliation)
- cyan: Hybrid (Static + GEMM-SYRK Spoliation)
- purple: Best Dynamic

Outline

Static or Dynamic ?

- First Conclusions
  - Dynamic Schedulers achieve good results
  - They are able to use both fast and slow resources
- Static Schedules are (expected to be) bad since they are
  - Hard to compute
  - Likely to perform badly under uncertainties
- But...

Dynamic Schedulers are far from the optimal

- MapReduce and Non Local Map Tasks
  - Hadoop: replicates to improve locality
    - greedily allocate tasks to resources
    - induces comms (highly depends on settings, was 10%)
  - Static policy:
    - solution of a $b-$matching problem, almost no comms
- Matrix-Multiplication and data transfers
  - StarPU: puts tasks resources that already hold input data
    - $2.5\times$ the lower bound
  - Static policy: based on an initial partitioning of the matrices
    - NP-Complete but 1.15 worst case and on av. 1.05 the LB
- Cholesky and Heterogeneous Unrelated Resources
  - StarPU: Allocate on a CPU if it ends faster than on GPUs
    - with $n = 12$, 680 GFlops (still better GPUs only)
  - Static: Schedule based on Brute Force Constraint Prog.
    - Hard Optimization Problem but 785 GFlops (even with noise)

Injecting Static Knowledge into Dynamic Schedulers

- MapReduce and Non Local Map Tasks
  - Use Static Optimal Solution to find local preferred tasks
  - $\rightarrow$ works well even with non homogeneous tasks
- Matrix-Multiplication and data transfers
  - StarPU policy: $\rightarrow 2.5\times$ the lower bound
  - Analyze the fluid relaxation of the system $\rightarrow 2\times$ LB
  - Best Static $\rightarrow 1.05\times$ LB
- Cholesky Factorization: Heterogeneous Unrelated Resources
  - StarPU policy $\rightarrow$ 680 GFlops
  - Use Simulation to have a less myopic policy $\rightarrow$ 725 GFlops
  - Use Knowledge of Cholesky to define affinities $\rightarrow$ 760 GFlops
  - Hard Combinatorial Optimization Problem $\rightarrow$ 785 GFlops

Injecting Dynamism into Static Policies

- MapReduce and Non Local Map Tasks
  - Replication enables to cope with uncertainties in execution times
- Matrix-Multiplication and data transfers
  - If processing times fluctuate
    - Start with Static then Use Work Stealing
    - $1.05 \rightarrow 1.5\times$ (for extreme & unrealistic variations)
  - still better than the 2.5 or even 2
- Cholesky Factorization: Heterogeneous Unrelated Resources
  - If processing time Fluctuate
    - Keep the same allocation + ordering of tasks
    - except if GPUs get idle and CPUs perform GEMMs $785 \rightarrow 770$ GFlops
    - still better than 760 GFlops

Static or Dynamic ?

- Static Schedules are bad since they are
  - Are hard to compute
  - Are likely to perform badly under uncertainties

Static or Dynamic ?

- Static Schedules are bad since they are
  - Are hard to compute
    - **True** not so much for MR and MM but Cholesky ($n = 12$) took 15 days)-;
  - Are likely to perform badly under uncertainties
    - **Not True** for MR, MM and Cholesky:
    - Adding some Work Stealing works well in practice

Static or Dynamic ?

- Static Schedules are bad since they are
  - Are hard to compute
    - **True** not so much for MR and MM but Cholesky ($n = 12$) took 15 days)-;
  - Are likely to perform badly under uncertainties
    - **Not True** for MR, MM and Cholesky:
    - Adding some Work Stealing works well in practice
- Still many things to do
  - Three simple kernels only
  - but we tried our best in all directions !
- There is plenty of room
  - Work on Optimization Problems for specific applications
  - Inject Knowledge of the Application into Dynamic Schedulers

Static or Dynamic ?

- Static Schedules are bad since they are
  - Are hard to compute
    - **True** not so much for MR and MM but Cholesky ($n = 12$) took 15 days)-;
  - Are likely to perform badly under uncertainties
    - **Not True** for MR, MM and Cholesky:
    - Adding some Work Stealing works well in practice
- Still many things to do
  - Three simple kernels only
  - but we tried our best in all directions !
- There is plenty of room
  - Work on Optimization Problems for specific applications
  - Inject Knowledge of the Application into Dynamic Schedulers

Thank You ! Mail to {Olivier.Beaumont@inria.fr}

2×1 year postdoc positions starting anytime before Dec 2016