StarPU scheduler vs. the analyst: who is right?

Vinícius Garcia Pinto, Lucas Mello Schnorr, Arnaud Legrand

September 5, 2018

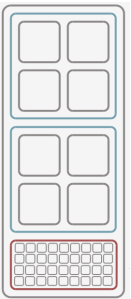
Motivation

State-of-the-art High-Performance Computing **platforms**

- **Hybrid** architectures
- multilevel parallelism
  - vector instructions
  - multithreaded cores,
  - multicore processors
  - manycore accelerators
- complex memory hierarchy

Paradigm shift in the hardware

- limitations of traditional tools
  - programming and analyzing



Xeon E5-2670

Xeon Phi 5110P

Cascade
#53 at Top500
128GB RAM per node

Source: http://bit.ly/cascadefig

Motivation

Traditional tools for parallel *programming* and performance *analysis*

- homogeneous platforms
- HPC/scientific domains

Common **programming** strategy on hybrid platforms

- use of explicit models
  - MPI + *pthreads*, OpenMP + CUDA, MPI + CUDA
- unfeasible and potentially problematic
  - tightly coupled to the hardware

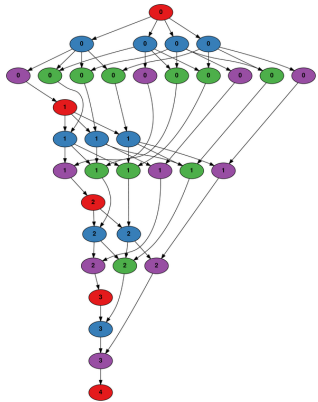
Promising approach

- abstraction layer to manage the heterogeneity of the hardware
  - task-based programming
  - dynamic runtime systems

Motivation

Task-based programming

- reduced programming complexity
- improved performance portability
- better management of multi-level parallelism
  - automatic data-transfers
  - multi-implementation tasks (CPU, GPU, …)
- **dynamic runtime systems**
  - scheduling
  - load balancing
  - communications (intra-node, inter-node)
  - synchronizations



Motivation

*Application **performance** depends heavily on **runtime system** decisions*

- Performance **analysis**
  - highly-summarized metrics
    - makespan, speedup, efficiency
  - several well-established tools
    - structured applications
    - homogeneous platforms
  - few task-oriented tools
    - runtime system
    - non-structured applications
    - hybrid platforms

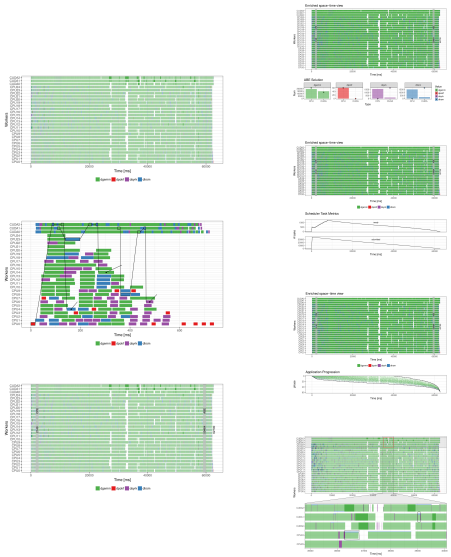
Section 2

Contribution

Contribution

Task-based **analysis strategies**

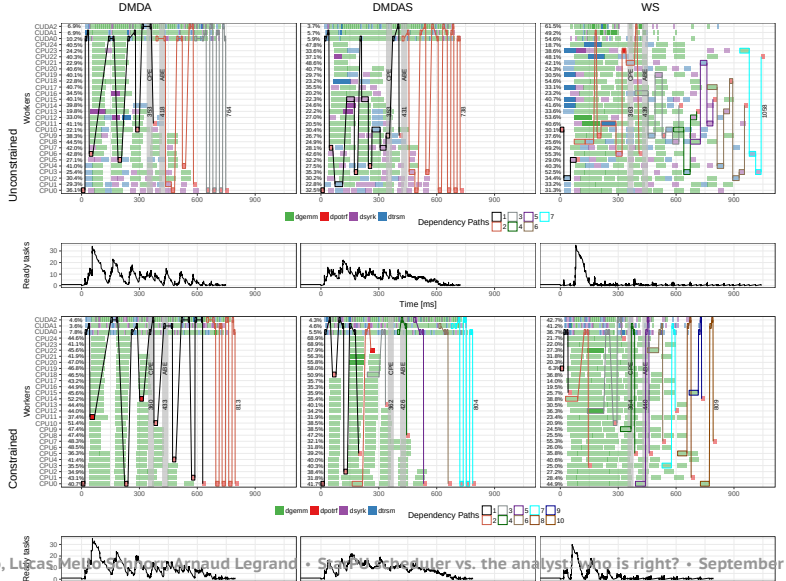
- Implemented on top of modern analysis tools
  - R programming language
  - paje (*pj-dump*)
  - *ggplot2*
  - *tidyverse* data manipulation packages
  - *plotly*
- **Enriched Space-time view**
- **Additional Panels**



Contribution

- This work presents a **simulation and debugging** approach:
  - a workflow to investigate *potential* scheduling mistakes in the StarPU runtime system
    - simulation/emulation
    - StarPU + SimGrid
    - debugging
    - GDB
    - custom scripts

Contribution - A motivating example



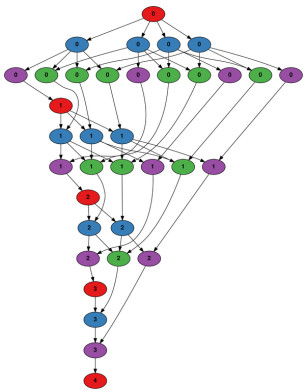
Section 3

How it works

A task-based application

- Application
  - **Cholesky** decomposition provided by the Chameleon Solver
    - built on top of the **StarPU** runtime system

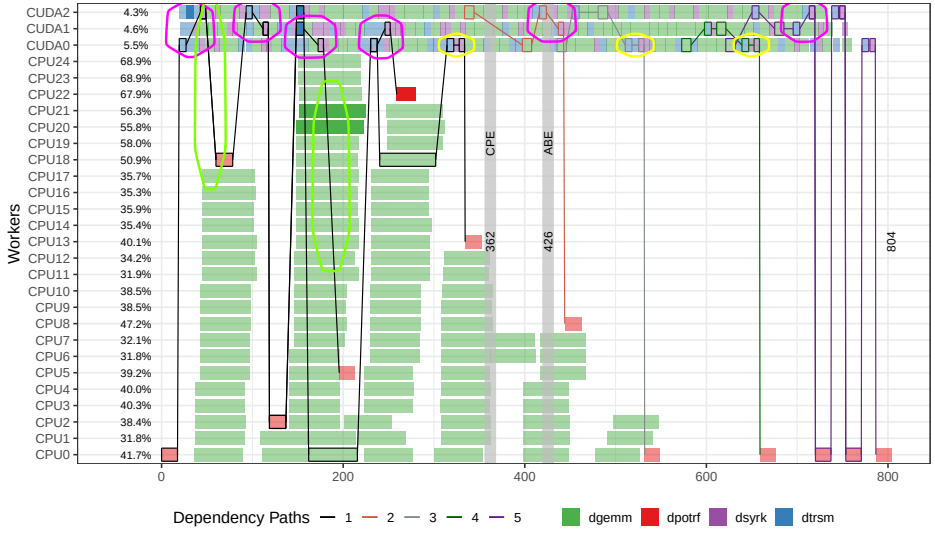
```
for (k = 0; k < N; k++) {
   DPOTRF(RW, A[k][k]);
   for (i = k+1; i < N; i++){
      DTRSM(RW, A[i][k],
               R, A[k][k]);
   }
   for (i = k+1; i < N; i++) {
      DSYRK(RW, A[i][i],
               R, A[i][k]);
      for (j = k+1; j < i; j++){
         DGEMM(RW, A[i][j],
                  R, A[i][k],
                  R, A[j][k]);
      }
   }
}
```



An experimental platform

	idcin2
Nodes	1
Processors per Node	2
Processor	Xeon E5-2697v3 2.60GHz
Cores per Processor	14
Core count	28
Memory	256GB
GPUs per Node	3
GPU	GeForce GTX TITAN X
GPU Memory	12GB
Interconnection	-
Linux kernel	3.2.0-4-amd64
Chameleon	0.9.1 (master)
StarPU	1.3 (trunk)
OpenMPI	-
BLAS	Eigen BLAS 3.3
CUDA version	7.5
CUDA Driver	352.39
GCC	4.7

A small and already optimized execution



A small and already optimized execution

Some issues

- **Delayed tasks** in the critical path
  - highlighted by *oblique dependency lines*
- Each of these tasks could be executed sooner
  - just after the end of its last dependency
- Scheduling mistakes?
  - StarPU runtime system
- Wrong priorities?
  - Application programmer
  - Runtime system
- Platform disturbances?

The workflow

Online steps

1. Re-execute the application using simulation mode (StarPU-SimGrid)
   - with the same application and platform performance models from the real (non-simulated) execution
2. Pause the simulation/emulation at the exact moment when a new task is scheduled
   - with custom GDB scripts, collect the state of internal scheduling variables
3. Resume the simulation and collect the traces

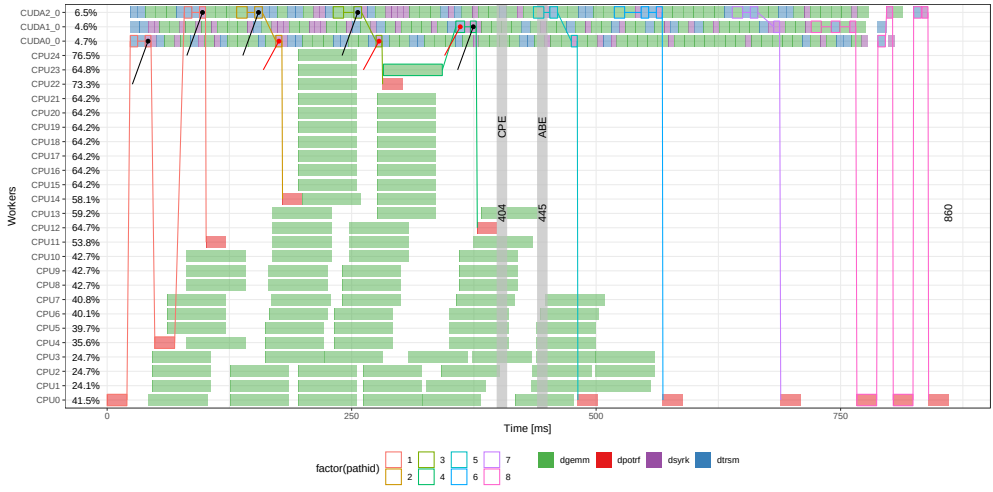
Offline steps

1. Rebuild the scheduler state when scheduling the new task
   - visual representation of the scheduler estimations
2. Compare the estimations with the final execution

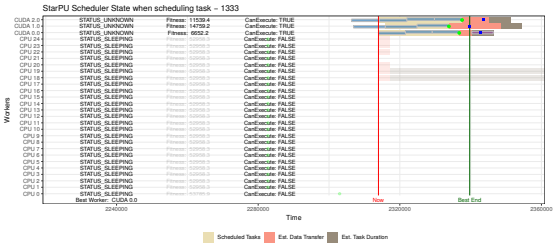
Section 4

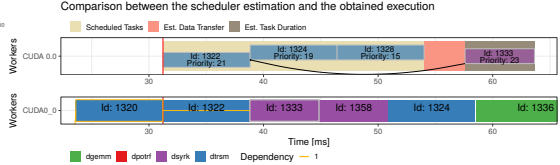
The analysis or *who is right?*

Some interesting tasks

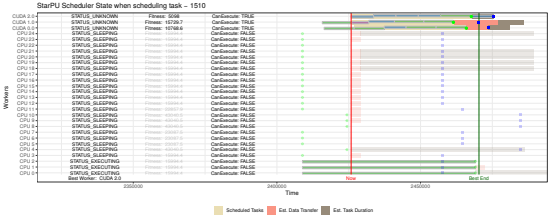


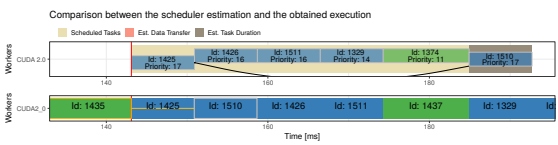
Some interesting tasks





Some interesting tasks





Some interesting tasks

Investigating Scheduling Decisions

- Other tasks
  - 9 other tasks are delayed due to the same reasons
    - pipeline and/or data transfers
- Experiments with Modified Pipeline Size (2 versus 1)
  - critical tasks are executed sooner
  - in the end, the lack of overlap increases GPU idleness
    - worst overall performance

Section 5

Final Remarks

StarPU scheduler vs. the analyst: who is right?

- There is no scheduling mistake
  - StarPU is choosing the best worker and the task priorities are respected
    - So, the StarPU scheduler is right ☹ ☹ ☹
- Delays can be explained by two other reasons
  - data transfers (prefetch)
  - pipeline mechanism
- However, we can propose some modifications ☺ ☺ ☺
  - a speculative prefetch strategy
  - a preemptive strategy in the pipeline mechanism
    - swap lower-priority task with the new one