Modeling and Simulation of Dynamic Task-Based Applications

Luka STANISIC
luka.stanisic@inria.fr

Inria, Bordeaux Sud-Ouest, France

SOLHAR meeting
Toulouse
December 2, 2016

Parallel Programming Challenges

- Communications and data placement
- Synchronization
- Scalability
- Portability of code and performance
- Multiplicity of computer architectures (CPU, GPU, MIC, ...)
- Multitude of technologies (MPI, OpenMP, CUDA, ...)

$\rightsquigarrow$ need for performance evaluation on a regular basis

Evaluating Performance of Parallel Applications

Native experiments

- Complex systems
- Wide variety of setups
- Faithful but expensive

Model, equations, theory

- PRAM, BSP, DAG
- Scheduling bounds
- Quick trends but simplistic

Evaluating Performance of Parallel Applications

Native experiments

- Complex systems
- Wide variety of setups
- Faithful but expensive

Model, equations, theory

- PRAM, BSP, DAG
- Scheduling bounds
- Quick trends but simplistic

Simulation: running real code with machine abstraction

Advantages:

- Reproducible executions (performance, bugs)
- Predictions on unavailable architectures (extrapolation)
- Richer experimental design possible

Difficulties:

- Implementing more than a simple prototype
- Hard to validate its reliability

Outline

1. Simulating Task-based Applications
   - Coupling StarPU and SimGrid
   - Evaluating Linear Algebra Applications on a Single Node
   - Evaluating Linear Algebra Applications on Multiple Nodes (ongoing)
2. Modeling Duration of Complex Computation Kernels
   - Ad hoc Modeling of qr_mumps Kernels
   - Automatizing Modeling Process (new)
3. Perspectives

StarPU and SimGrid

StarPU (Inria Bordeaux)

- Dynamic runtime for hybrid architectures (CPU, GPU, MPI)
- Opportunistic scheduling of a task graph guided by performance models
- Features dense, sparse and FMM applications

SimGrid (Inria Grenoble, Lyon, Nancy, ...)

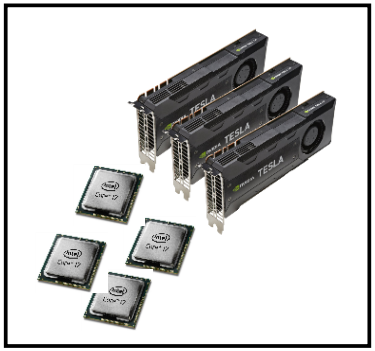
- Scalable simulation framework for distributed systems
- Sound fluid network models accounting for heterogeneity and contention
- Modeling with threads rather than only trace replay
  $\rightsquigarrow$ ability to simulate dynamic applications
- Portable, open source and easily extendable

StarPU was ported on top of SimGrid by Samuel Thibault

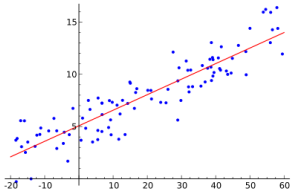
Same approach could be applicable to any task-based runtime

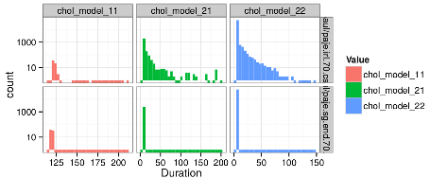
Devised Workflow: StarPU+SimGrid

Calibration



StarPU



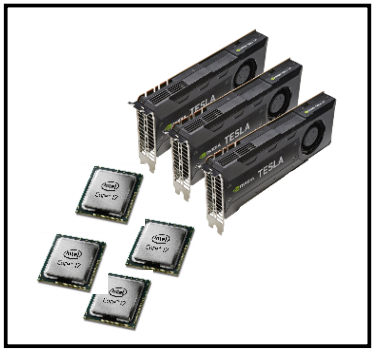


Performance Profile

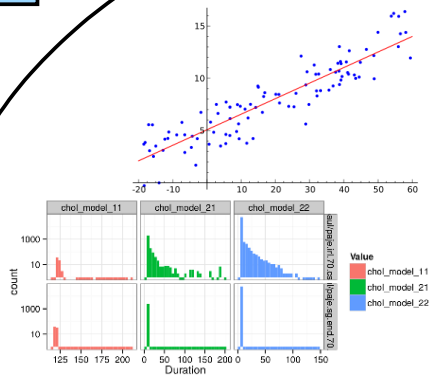
Run once!

Devised Workflow: StarPU+SimGrid

Calibration



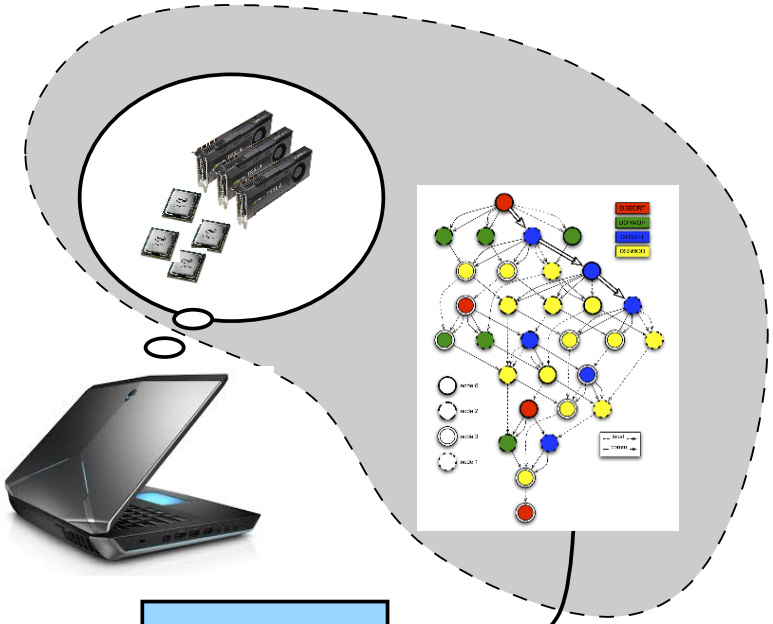
StarPU



Performance Profile

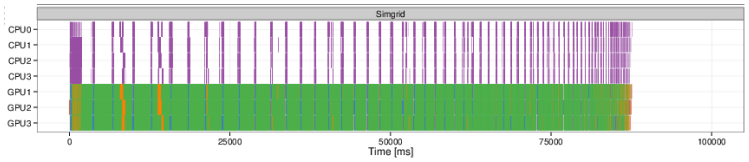
Run once!

Simulation



StarPU

SimGrid



Quickly Simulate Many Times

Implementation Principles

Emulation executing real applications in a synthetic environment

Simulation replace process execution by delays using performance models

- StarPU applications and runtime are *emulated*
  $\rightsquigarrow$ similar scheduling
- Thread synchronization, actual computations, memory allocation and data transfer are *simulated*
  $\rightsquigarrow$ need for a good computational kernel and communication models
- Control part of StarPU is modified to inject into SimGrid runtime, communication and computation delays

Modeling Runtime

Simulation delays (increasing simulated time)

- Process synchronizations
- Memory allocations of CPU or GPU
- Submission of data transfer requests

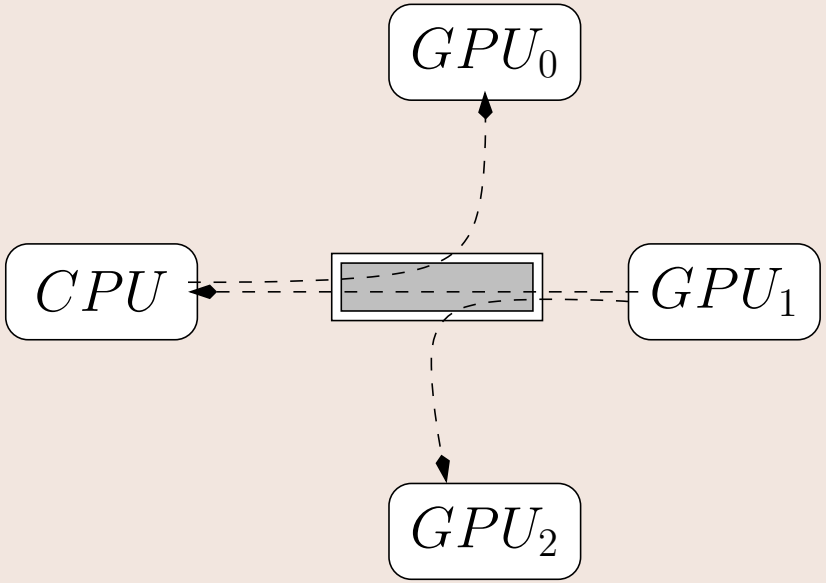
Example from *malloc.c* in StarPU

```
...
#ifdef STARPU_SIMGRID
   MSG_process_sleep((float) dim * alloc_cost_per_byte);
#else
   if (_starpu_can_submit_cuda_task()) {
     cudaError_t cures;
     cures = cudaHostAlloc(A, dim, cudaHostAllocPortable);
...
```

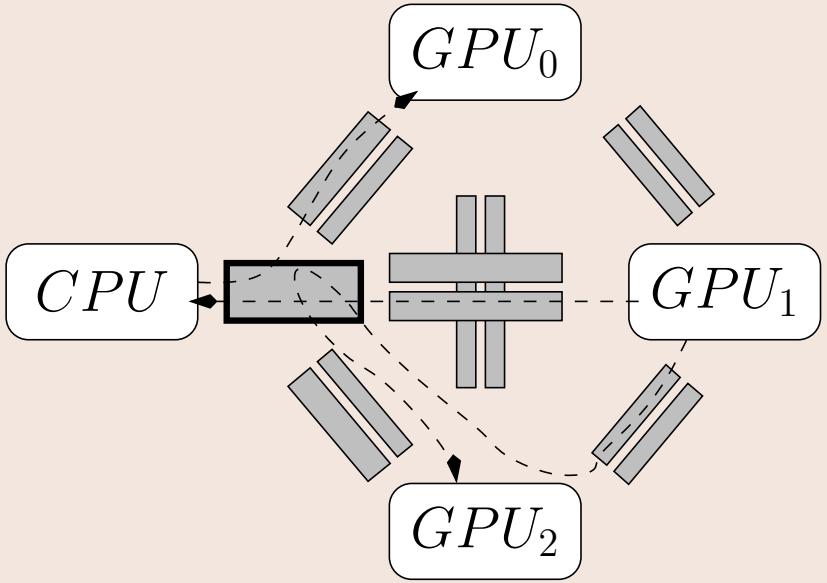
Modeling Communication

- Components of hybrid platforms have differing characteristics
- Correctly modeling their communication is of primary importance

Flexible flow-based contention model



(a) Fatpipe (crude) model

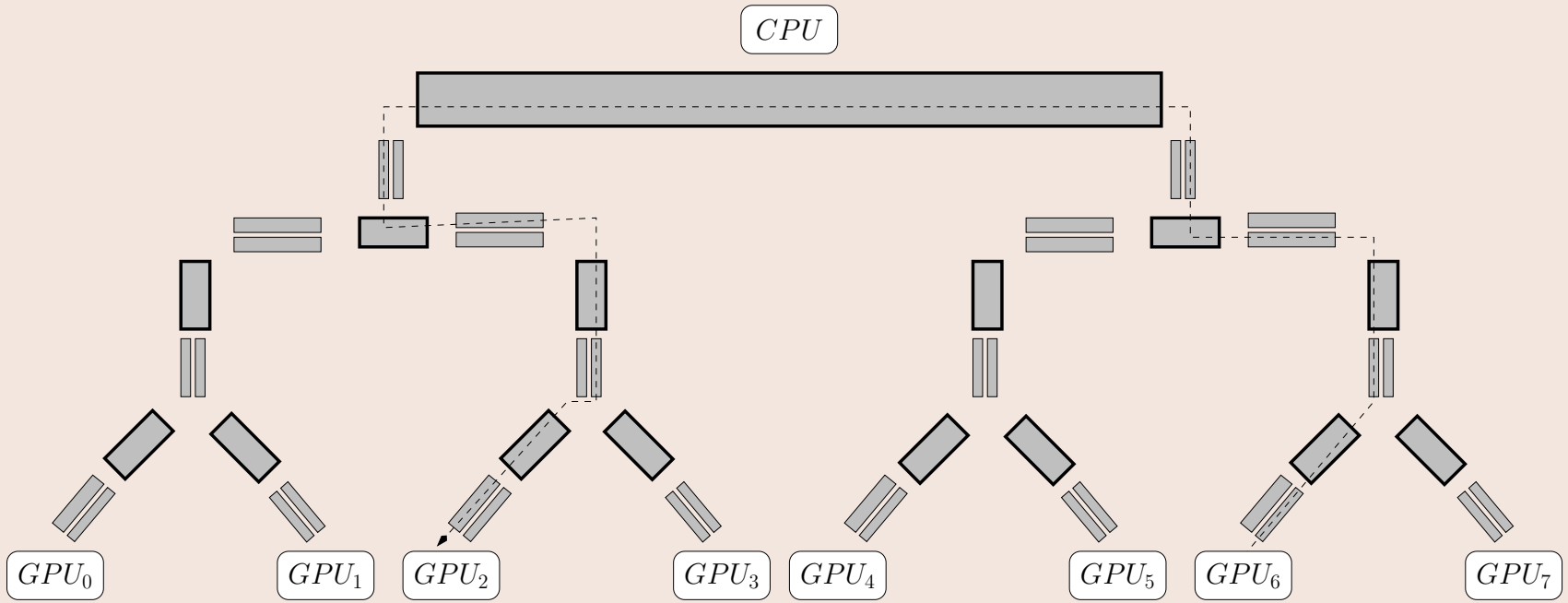


(b) Complete graph (pragmatic) model

Modeling Communication

- Components of hybrid platforms have differing characteristics
- Correctly modeling their communication is of primary importance

Flexible flow-based contention model

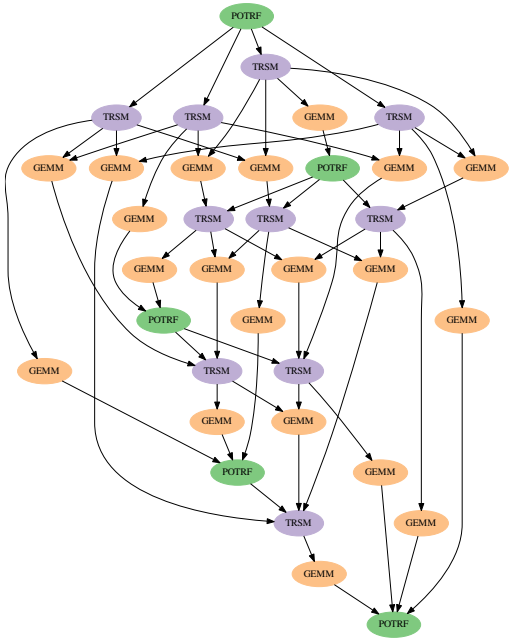


(c) Treelike (elaborated) model

Modeling Computation

Actual computation results irrelevant

- Application structure and execution do not depend on intermediate results
  $\rightsquigarrow$ only computation time matters



Execution of kernels replaced by simulation delays

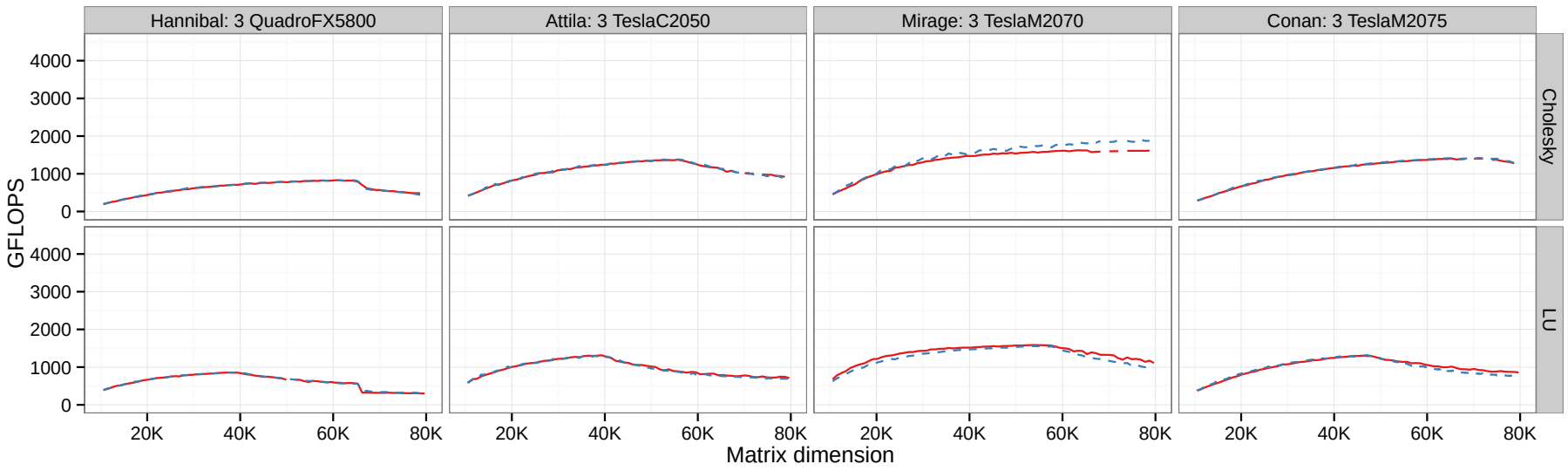
- Average duration works just fine for dense linear algebra kernels
- Additional variability can be introduced (through histograms, . . .) to remove potential bias

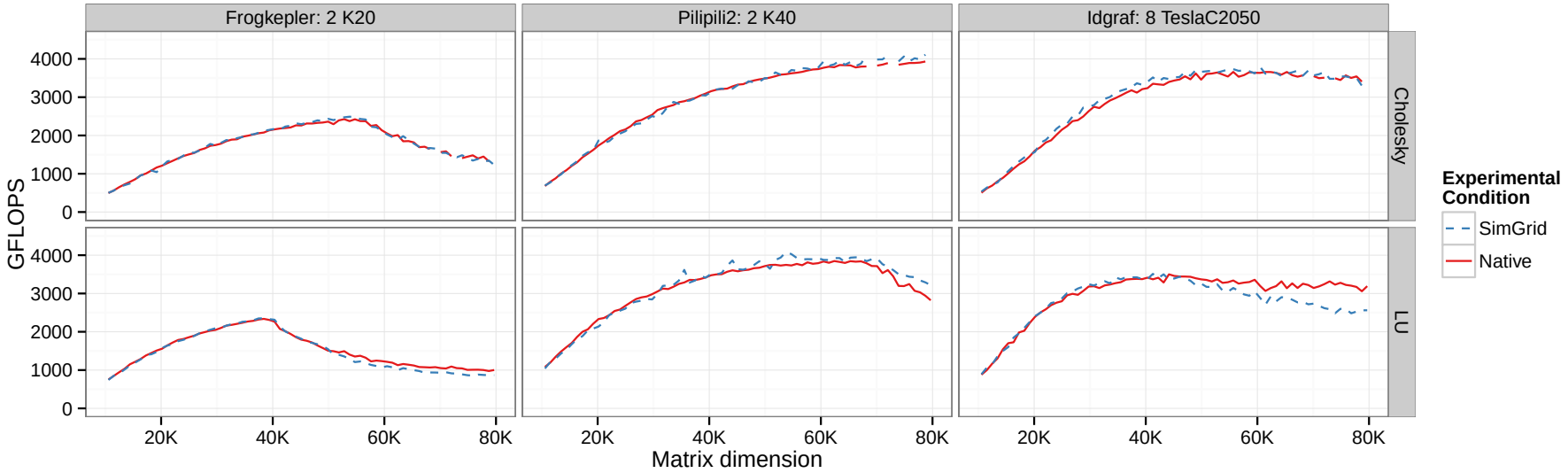
Outline

Evaluating Linear Algebra Applications

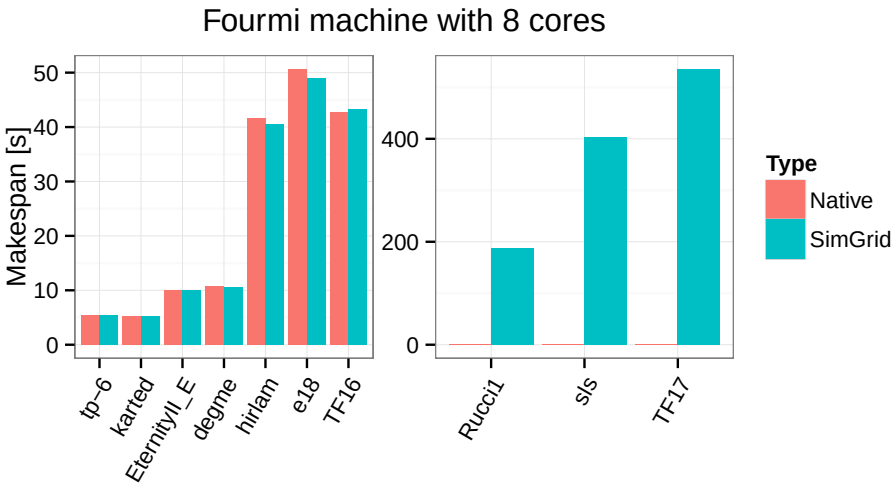
- Started with regular dense kernels and a fixed tile size
  - Cholesky
  - LU
- Moved to irregular codes
  - qr_mumps: MUMPS multi-frontal factorization on top of StarPU
- Validated approach on a wide diversity of machines

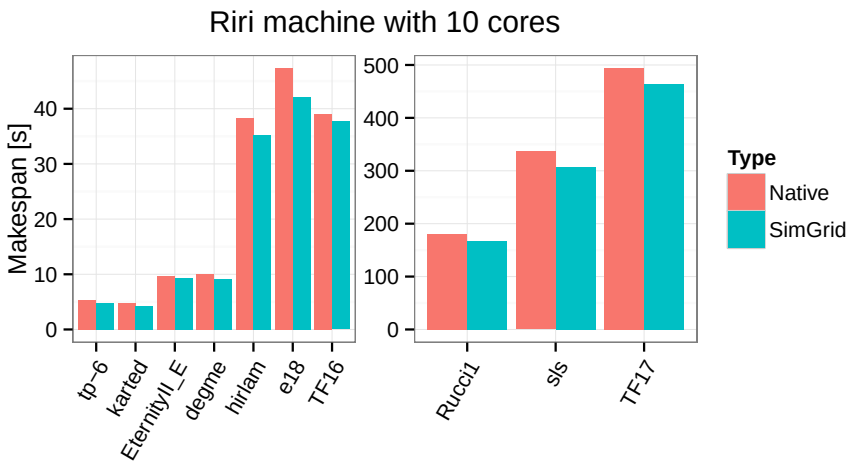
Overview of Simulation Accuracy (Dense)



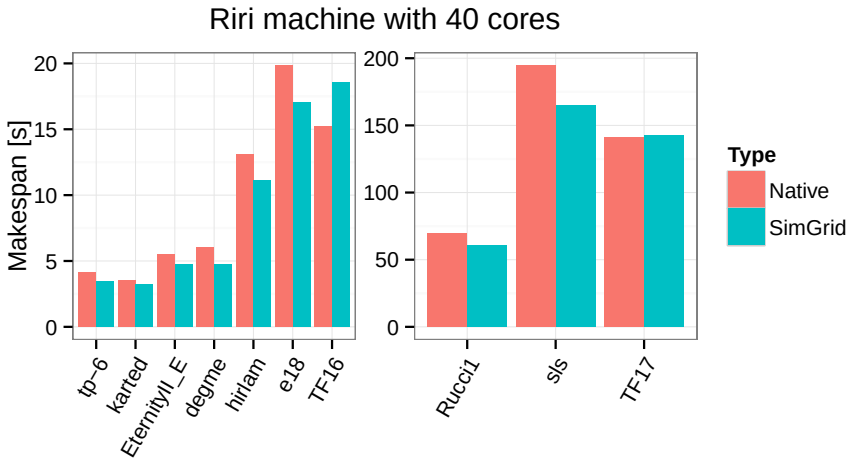


Overview of Simulation Accuracy (Sparse)



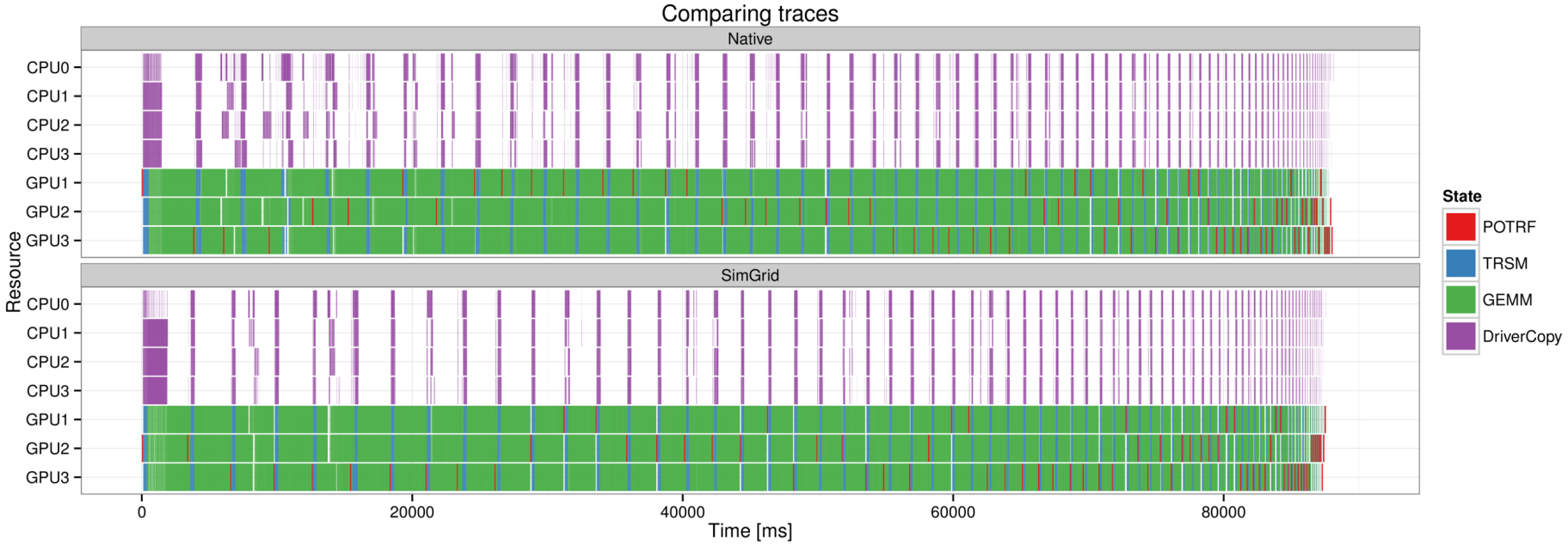


- Most of the time, simulation is slightly optimistic
- With bigger and architecturally more complex machines, error increases



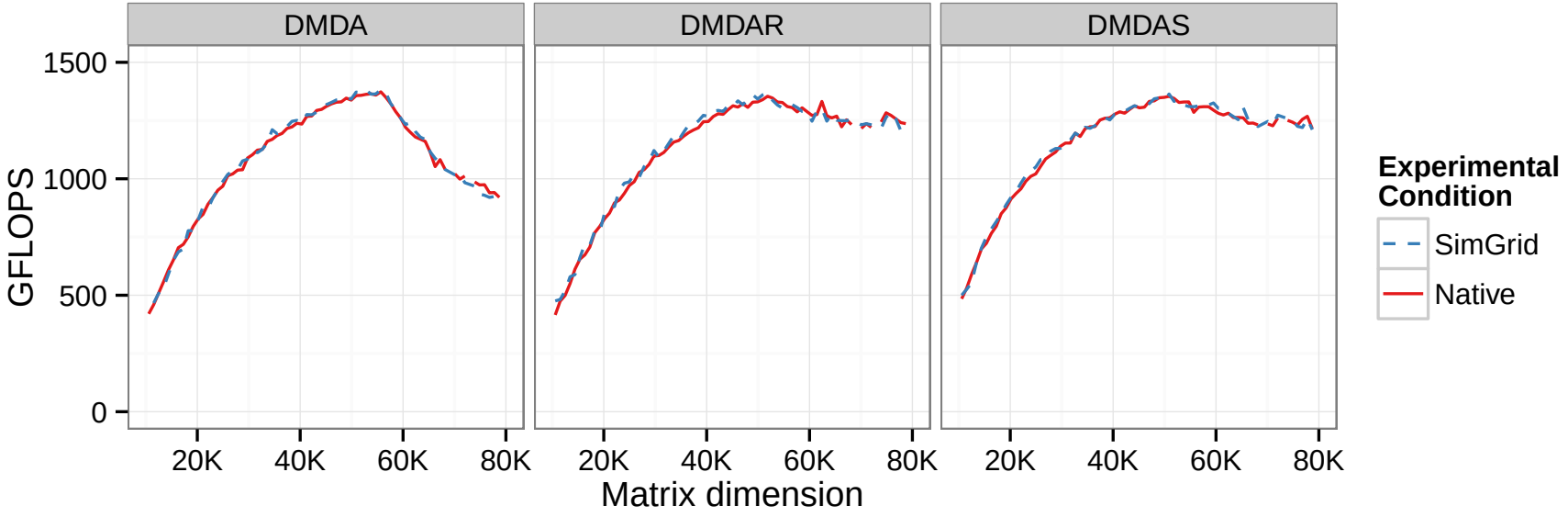
Visual Trace Comparing

- GFLOPS are a limited metric
- Verifying that simulation traces are representative



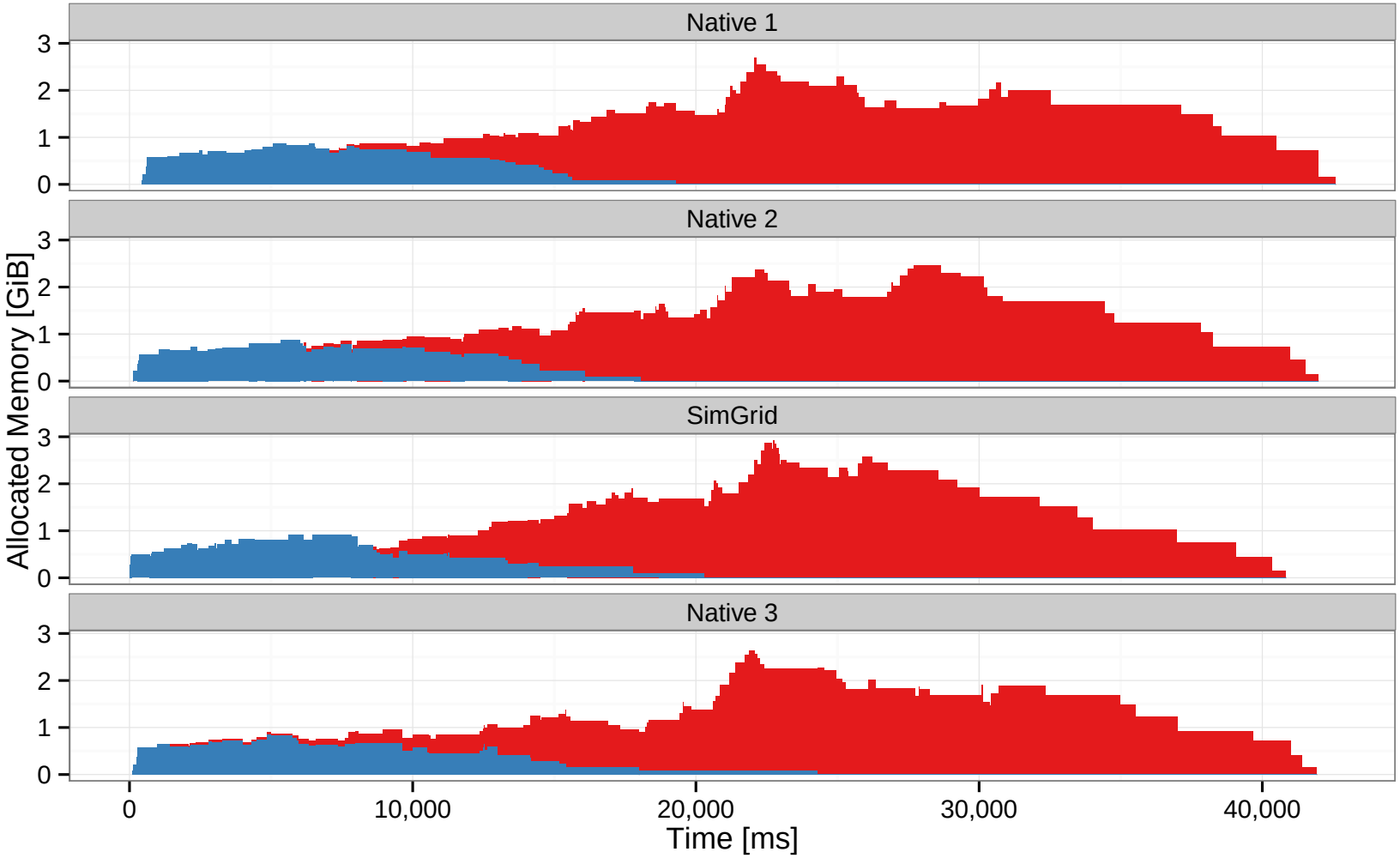
Comparing Different Schedulers

- Differences between schedulers performances faithfully predicted
- DMDAR and DMDAS locality aware schedulers
  $\rightsquigarrow$ less transfers between GPUs



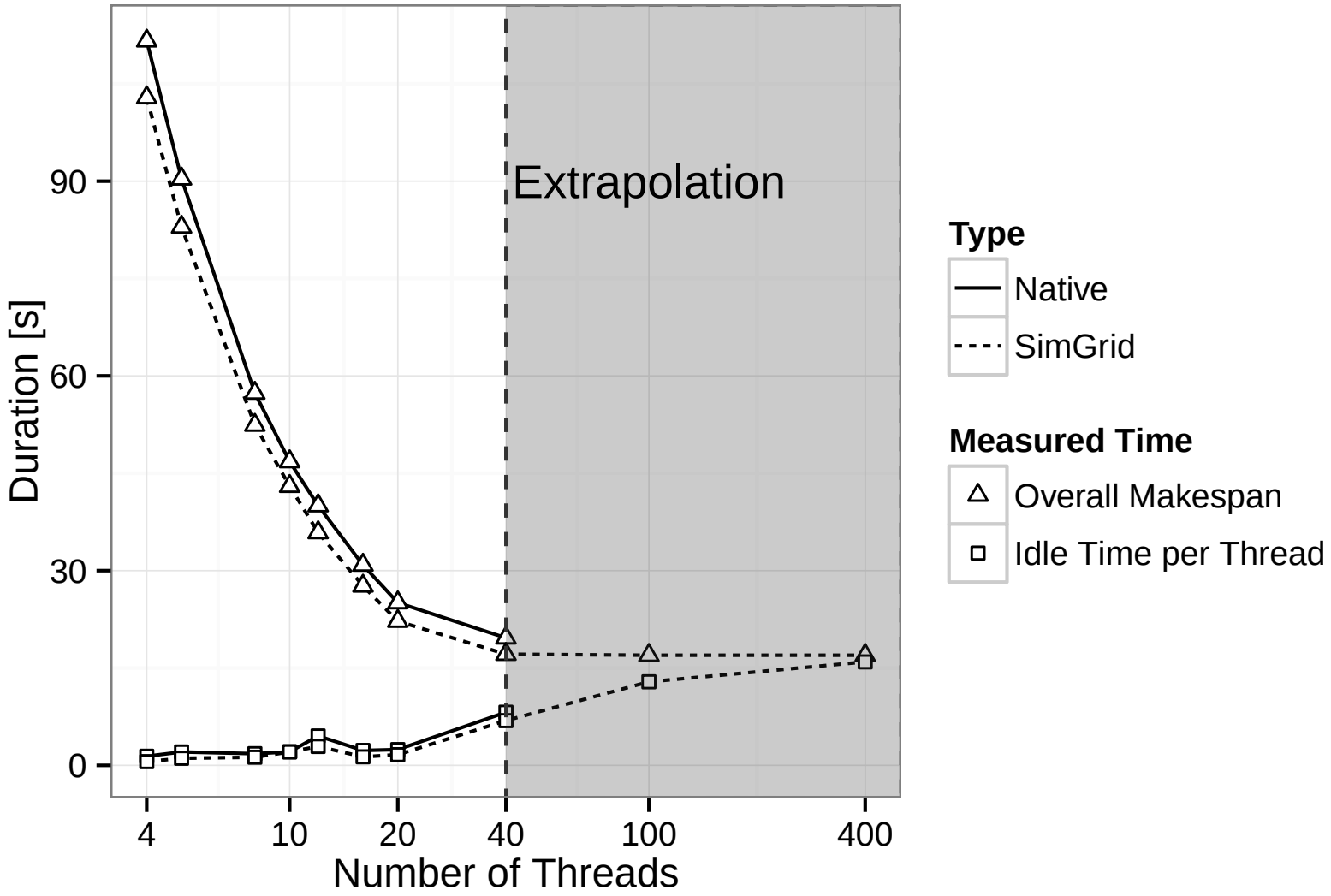
Studying Memory Consumption

- Minimizing memory footprint is very important for such applications
- Remember scheduling is dynamic so consecutive Native experiments have different output



Extrapolating to Larger Machines

- Predicting performance in idealized context
- Studying the parallelization limits of the problem



Achievements

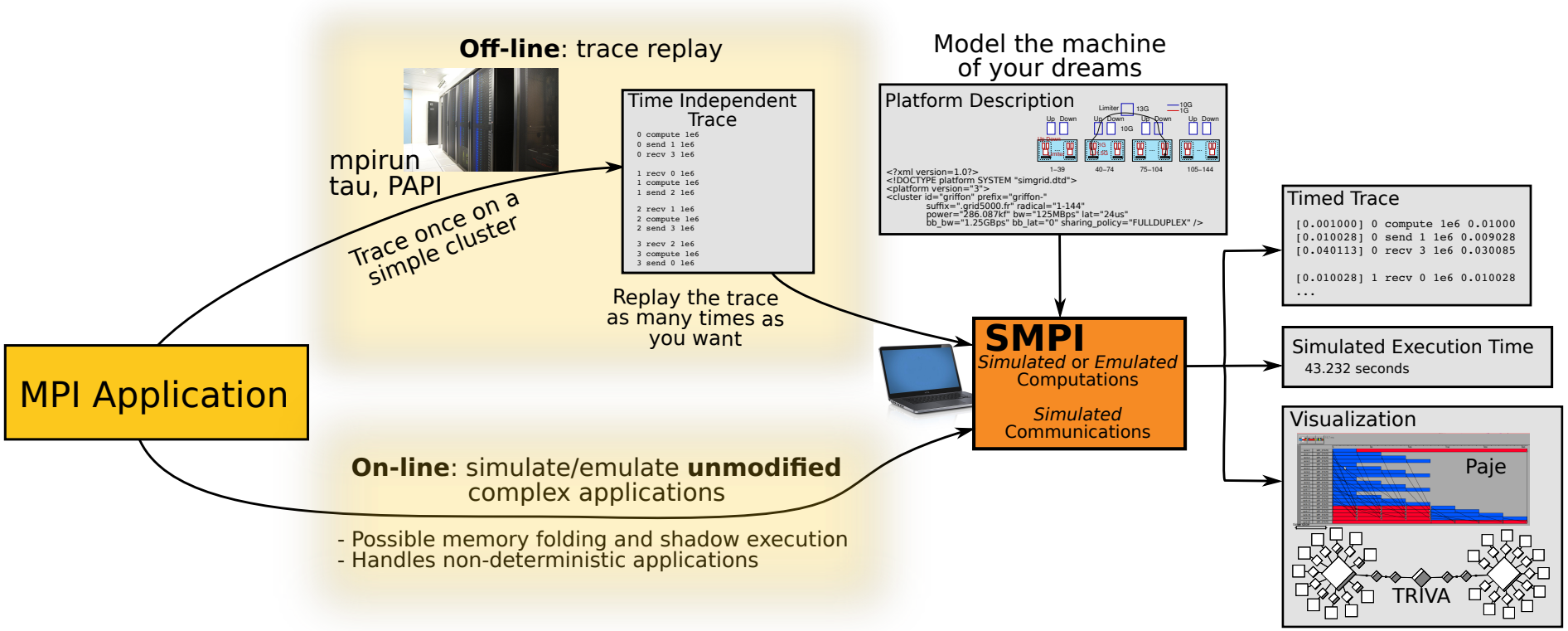
- Works great for small hybrid setups with both dense and sparse linear algebra StarPU applications
  - Additionally ScalFMM and QDWH on homogeneous nodes
- Not only a prototype, already used by other researchers
- Our solution allows to:
  - Debug applications on a commodity laptop in a reproducible way
  - Detect problems with real experiments using reliable comparison
  - Test different scheduling alternatives
  - Evaluate memory footprint
  - Quickly and accurately evaluate the impact of various scheduling/application parameters:

qr_mumps	Cores	RAM	Evaluation	Makespan
Native	40	58.0GiB	157s	141s
SimGrid	1	1.5GiB	57s	142s

Outline

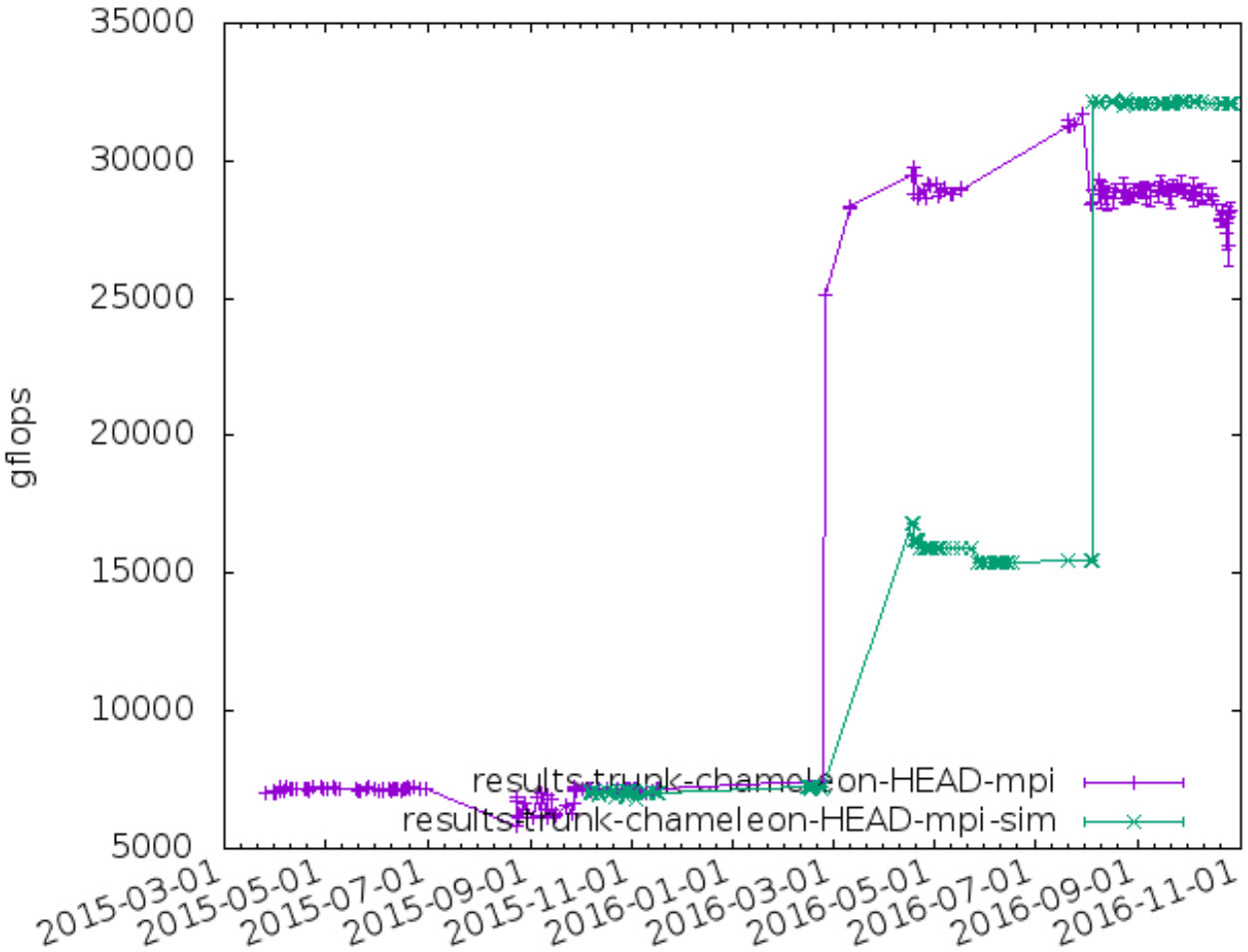
Simulating MPI

- SimGrid SMPI module
- Many successful studies
- Constantly evolving



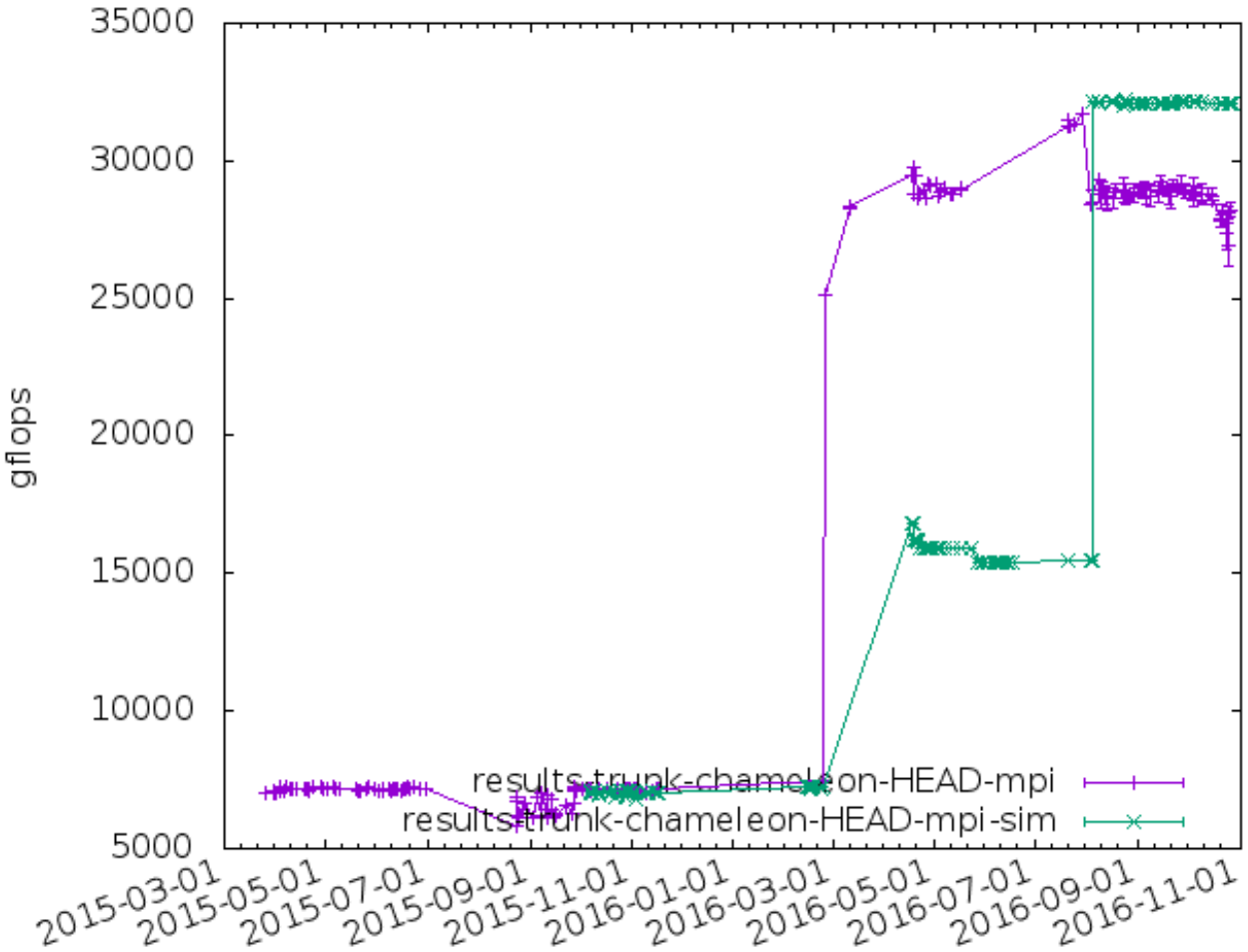
Simulating MPI+Task

StarPU-MPI+SimGrid works...



Simulating MPI+Task

StarPU-MPI+SimGrid works...



...but slow and with limited accuracy

Why is It More Difficult?

- Technically challenging
- Contention harder to model

Why is It More Difficult?

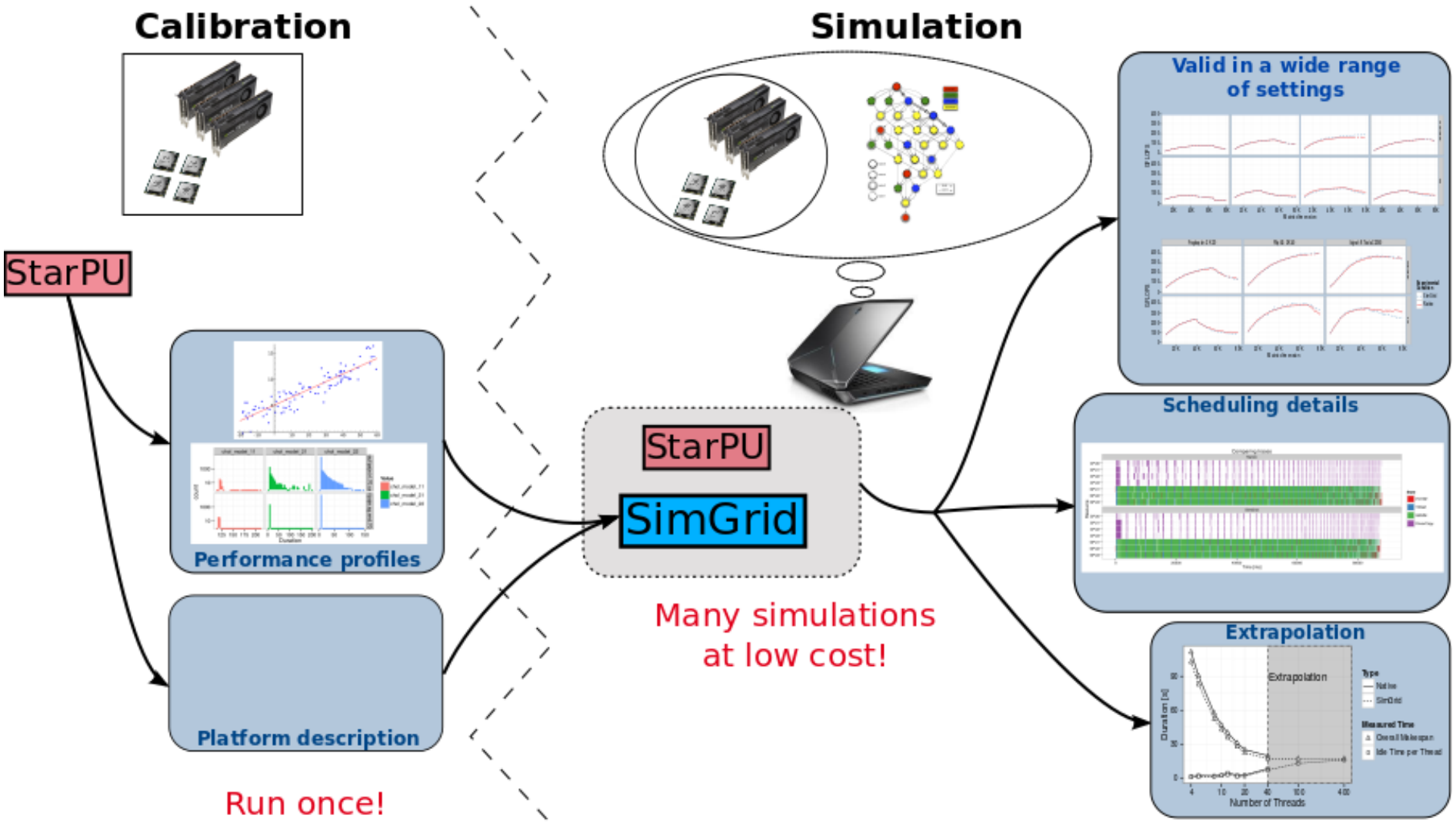
- Technically challenging
- Contention harder to model

Upcoming SimGrid improvements

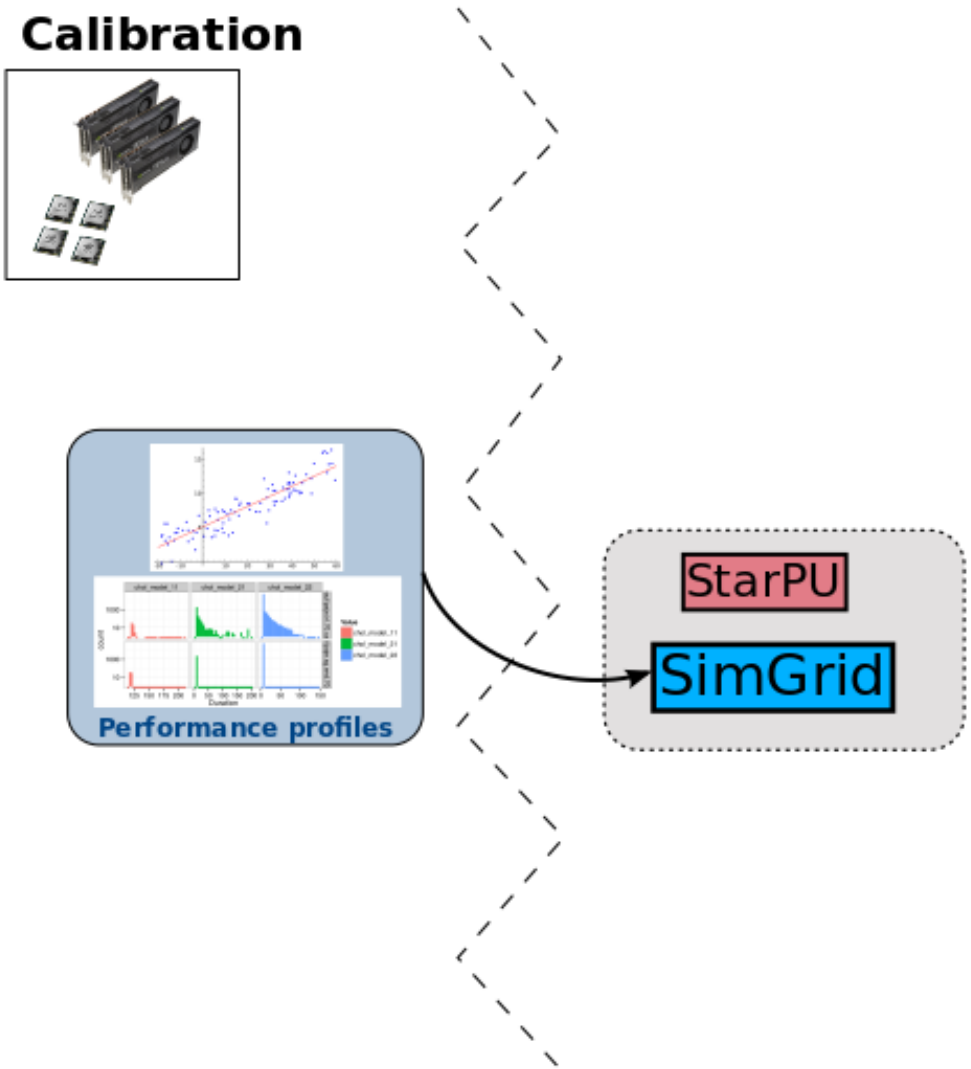
- Rewritten internals
- Automatized platform calibration (hwloc/netloc)
- More accurate multi-core models
- Combine multiple network models

Outline

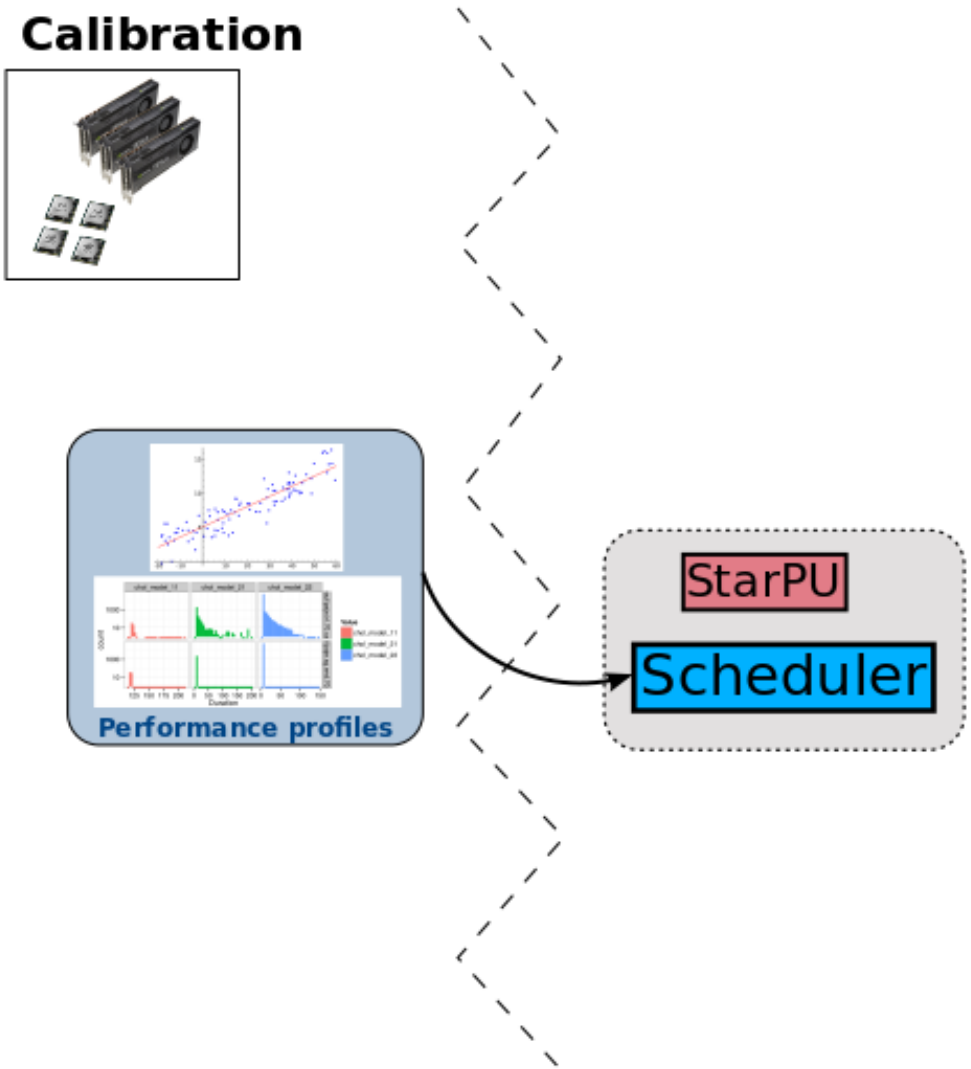
Focusing on Kernel Performance Models



Focusing on Kernel Performance Models

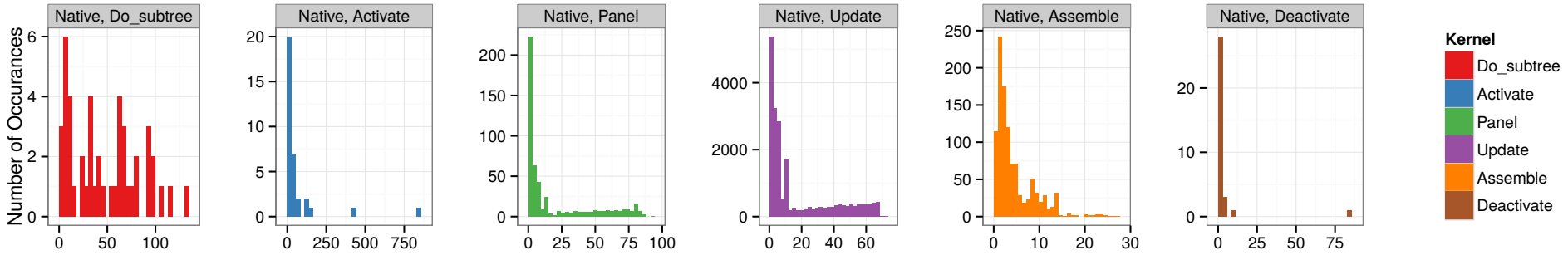


Focusing on Kernel Performance Models



Difference between Dense and Sparse Kernels

- Dense kernels (`POTRF`, `GEMM`, ...) during single experiment are always executed with the same block size
  $\rightsquigarrow$ duration is (relatively) stable
- Sparse kernel durations depend on their input parameters
  $\rightsquigarrow$ more variability
- Cannot model sparse kernels with simple mean values



Example for Modeling Kernels: `Panel`

- Theoretical `Panel` complexity:

$$T_{\texttt{Panel}} = a + 2b(NB^2 \times MB) - 2c(NB^3 \times BK) + \frac{4d}{3}NB^3$$

Example for Modeling Kernels: Panel

- Theoretical Panel complexity:

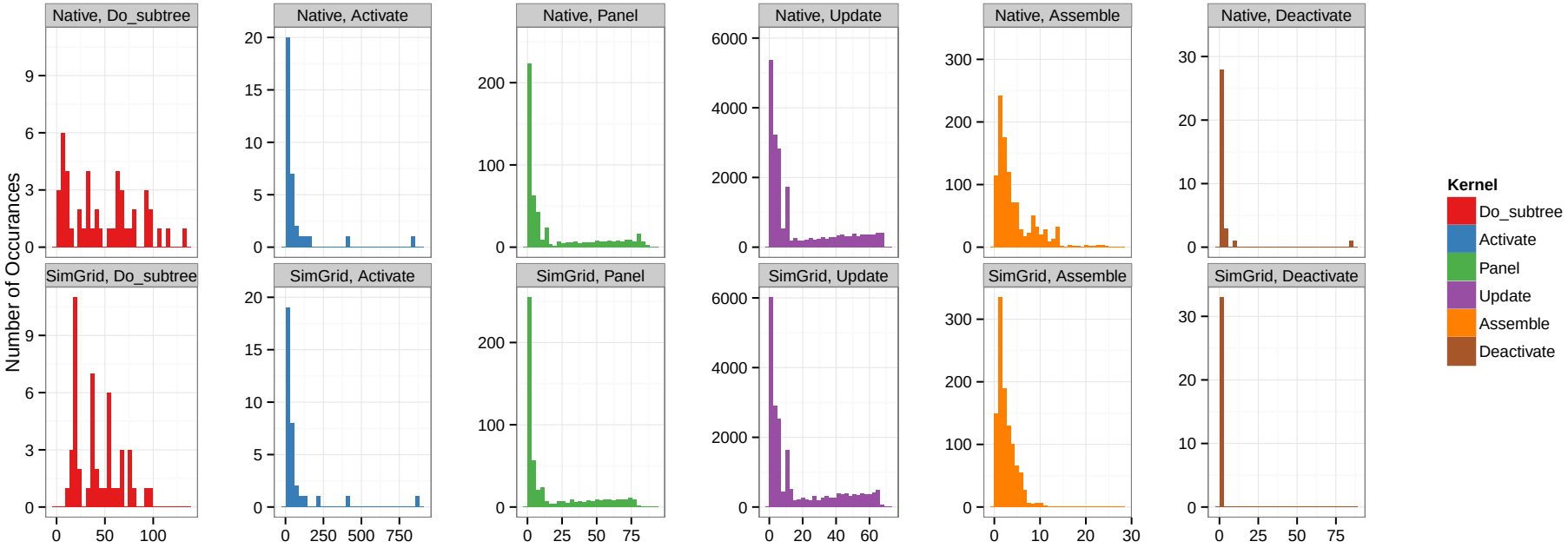
$$T_{\texttt{Panel}} = a + 2b(NB^2 \times MB) - 2c(NB^3 \times BK) + \frac{4d}{3}NB^3$$

- We can do a multiple linear regression based on ad hoc calibration

	Panel Duration	
Constant	-2.49×10^1 $(-2.83 \times 10^1, -2.14 \times 10^1)$	***
$NB^2 \times MB$	5.49×10^{-7} $(5.46 \times 10^{-7}, 5.51 \times 10^{-7})$	***
$NB^3 \times BK$	-5.52×10^{-7} $(-5.57 \times 10^{-7}, -5.48 \times 10^{-7})$	***
NB^3	1.50×10^{-5} $(1.30 \times 10^{-5}, 1.70 \times 10^{-5})$	***
Observations	493	
R^2	0.999	
Note:	*p<0.1; **p<0.05; ***p<0.01	

Comparing Kernel Duration Distributions

	Do_subtree	Activate	Panel	Update	Assemble
1.	#Flops	#Zeros	*NB*	*NB*	#Coeff
2.	#Nodes	#Assemble	*MB*	*MB*	/
3.	/	/	*BK*	*BK*	/
R^2	0.99	0.99	0.99	0.99	0.86



Limits of the Approach

Shortcomings

- Requires expertise in statistical analysis (R language)
- Manual analysis of parameter combinations and coefficients
- External code for computing kernel duration (with hardcoded values)
- Several small ad hoc modifications to the application and StarPU

Limits of the Approach

Shortcomings

- Requires expertise in statistical analysis (R language)
- Manual analysis of parameter combinations and coefficients
- External code for computing kernel duration (with hardcoded values)
- Several small ad hoc modifications to the application and StarPU

Envisioned solution

- Extended StarPU to support multiple linear regression (mlr) models
- Automatized the process as much as possible
- Provide good example of how to perform such study

Outline

How to Use New Perfmodels?

- Computing kernel duration:

$$T_{kernel} = a + b(M^{\alpha_1} \times N^{\beta_1} \times K^{\gamma_1}) + c(M^{\alpha_2} \times N^{\beta_2} \times K^{\gamma_2})$$

- If StarPU knows the formula, it can estimate kernel duration

How to Use New Perfmodels?

- Computing kernel duration:

$$T_{kernel} = a + b(M^{\alpha_1} \times N^{\beta_1} \times K^{\gamma_1}) + c(M^{\alpha_2} \times N^{\beta_2} \times K^{\gamma_2})$$

- If StarPU knows the formula, it can estimate kernel duration

1. Add all parameters to the task structure (sometimes new estimators)
   - Application developer - once per application kernel

How to Use New Perfmodels?

- Computing kernel duration:

$$T_{kernel} = a + b(M^{\alpha_1} \times N^{\beta_1} \times K^{\gamma_1}) + c(M^{\alpha_2} \times N^{\beta_2} \times K^{\gamma_2})$$

- If StarPU knows the formula, it can estimate kernel duration

1. Add all parameters to the task structure (sometimes new estimators)
   - Application developer - once per application kernel
2. Analyze experiment traces to find the right parameter combinations using R and multiple linear regression, add to the task structure
   - Application analyst - once per application kernel

How to Use New Perfmodels?

- Computing kernel duration:

$$T_{kernel} = a + b(M^{\alpha_1} \times N^{\beta_1} \times K^{\gamma_1}) + c(M^{\alpha_2} \times N^{\beta_2} \times K^{\gamma_2})$$

- If StarPU knows the formula, it can estimate kernel duration

1. Add all parameters to the task structure (sometimes new estimators)
   - Application developer - once per application kernel
2. Analyze experiment traces to find the right parameter combinations using R and multiple linear regression, add to the task structure
   - Application analyst - once per application kernel
3. StarPU automatically computes coefficients for a specific machine using `dgels_` from LAPACK/MKL/min-dgels
   - Application user - once per platform

How to Use New Perfmodels?

- Computing kernel duration:

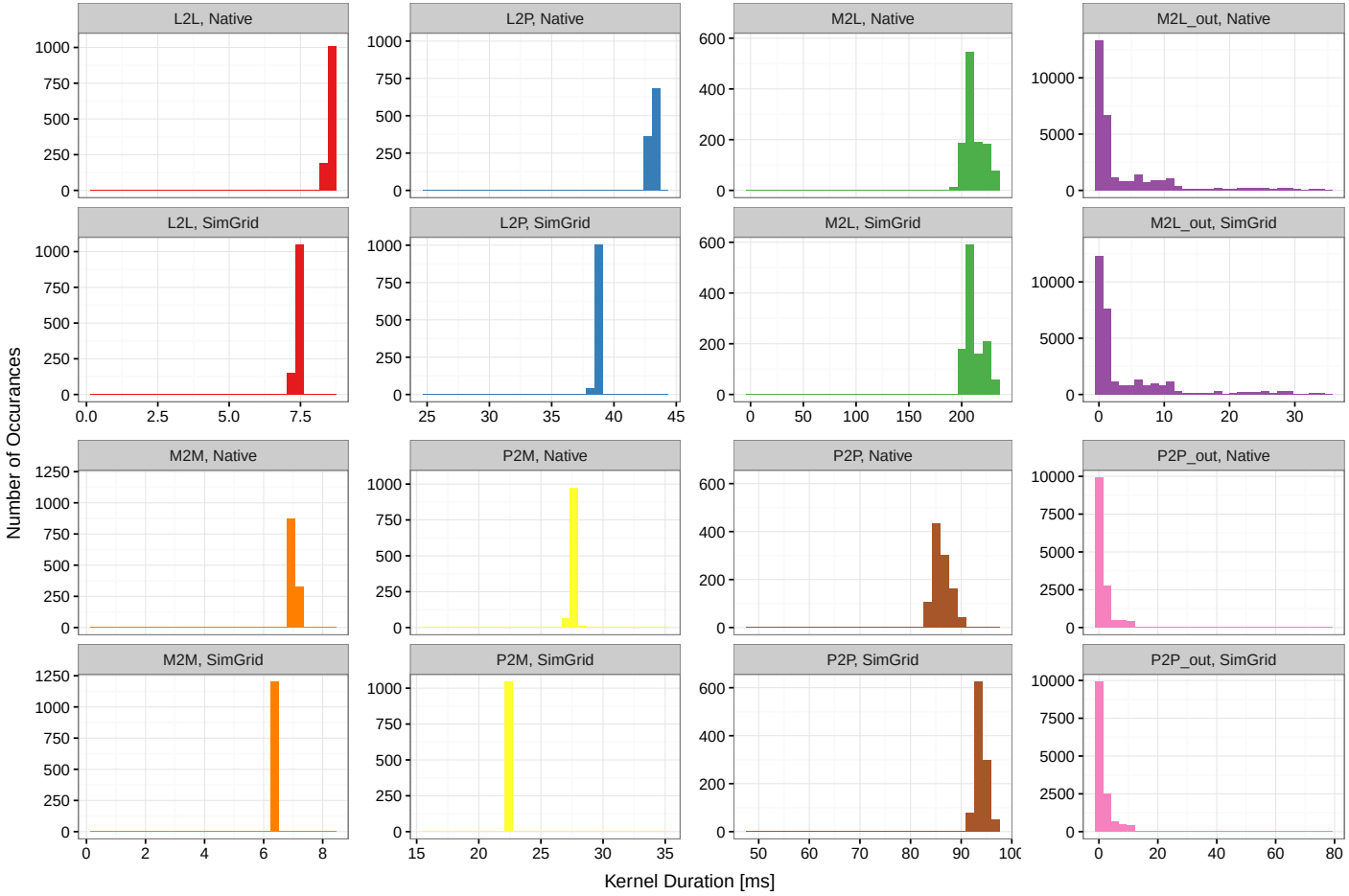
$$T_{kernel} = a + b(M^{\alpha_1} \times N^{\beta_1} \times K^{\gamma_1}) + c(M^{\alpha_2} \times N^{\beta_2} \times K^{\gamma_2})$$

- If StarPU knows the formula, it can estimate kernel duration

1. Add all parameters to the task structure (sometimes new estimators)
   - Application developer - once per application kernel
2. Analyze experiment traces to find the right parameter combinations using R and multiple linear regression, add to the task structure
   - Application analyst - once per application kernel
3. StarPU automatically computes coefficients for a specific machine using `dgels_` from LAPACK/MKL/min-dgels
   - Application user - once per platform
4. Now StarPU can automatically compute task duration estimation
   - Everyone can use - many times

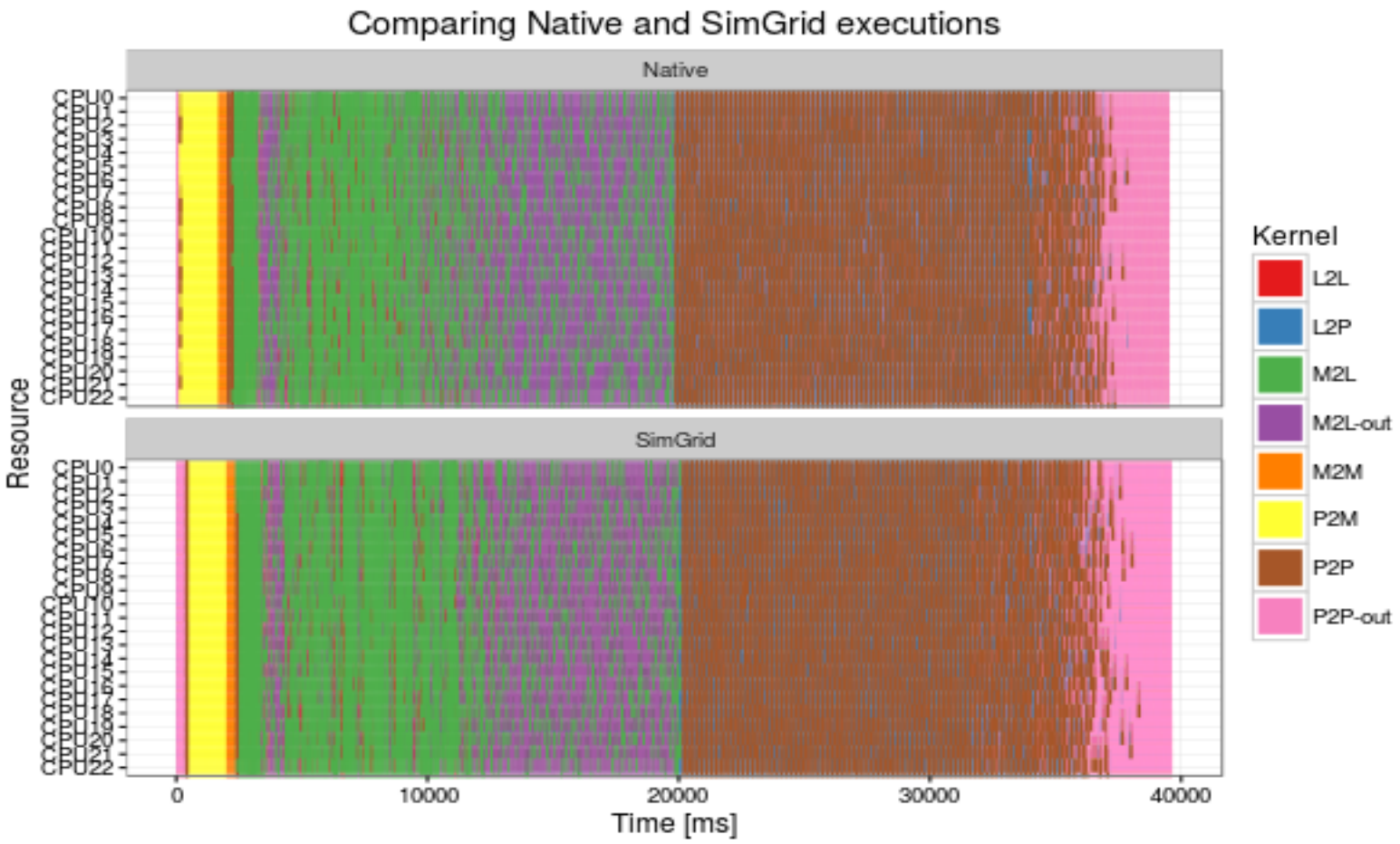
Evaluating Approach with ScalFMM

- ScalFMM simulates N-body interactions using Fast Multipole Method
- 8 complex kernels with very variable workload



Evaluating Approach with ScalFMM

- ScalFMM simulates N-body interactions using Fast Multipole Method
- 8 complex kernels with very variable workload



Outline

Future Work

- Perform an exhaustive study of StarPU-MPI+SimGrid
- Continue improving internals and (in)validating models

- Integrate new multiple linear regression models in qr_mumps
- Take into account cache contention and NUMA factor

- Model and simulate parallel workers or any other StarPU application