A Reproducible Research Methodology for Designing and Conducting Faithful Simulations of Dynamic Task-based Scientific Application

Luka Stanisic

Inria, Bordeaux Sud-Ouest, France

MPCDF seminar
Garching
February 24, 2017

Background

Phd
(supervisors A. Legrand & J.F. Mehaut)
Grenoble, France
Modeling and simulation of dynamic
task-based applications
Methodology for reproducible research
Statistical analysis, trace visualizations

Bachelor
(CS specialty)
EE faculty
Belgrade, Serbia

2011 2012 2013 2014 2015 2016 2017

Research Master
(parallelism specialty)
Grenoble, France
Benchmarking
CPU cache modeling
ARM vs Intel

PostDoc
Bordeaux, France
Performance optimization
Large scale simulations
Modeling complex kernels
Simulating openQCD

Background

Bachelor
(CS specialty)
EE faculty
Belgrade, Serbia

Phd
(supervisors A. Legrand & J.F. Mehaut)
Grenoble, France
Modeling and simulation of dynamic task-based applications
Methodology for reproducible research
Statistical analysis, trace visualizations

2011 2012 2013 2014 2015 2016 2017

Research Master
(parallelism specialty)
Grenoble, France
Benchmarking
CPU cache modeling
ARM vs Intel

PostDoc
Bordeaux, France
Performance optimization
Large scale simulations
Modeling complex kernels
Simulating openQCD

Parallel Programming Challenges

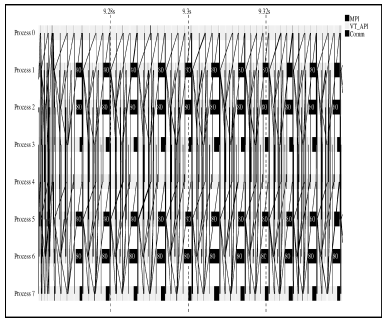
- Communications and data placement
- Synchronization of the workers

- Computation duration variability
  $\rightsquigarrow$ scalability

- Exploiting hybrid machines
- Choosing granularity
  $\rightsquigarrow$ portability of code and performance

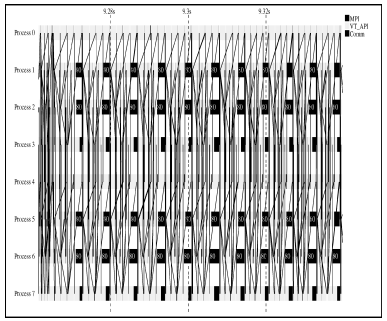
Different Programming Approaches

- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, . . .)
  - ☺ Perfect control ⇝ maximal achievable performance
  - ☺ Efficient granularity ⇝ advanced numerical features



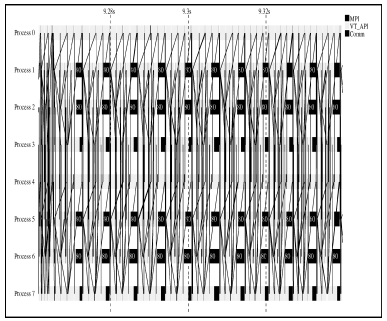
Different Programming Approaches

- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, . . .)
  - ☺ Perfect control ⇝ maximal achievable performance
  - ☺ Efficient granularity ⇝ advanced numerical features
  - ☹ Monolithic codes ⇝ hard to develop and maintain
  - ☹ Fixed scheduling ⇝ sensitive to variability
  - ☹ Hard and long to optimize ⇝ performance portability

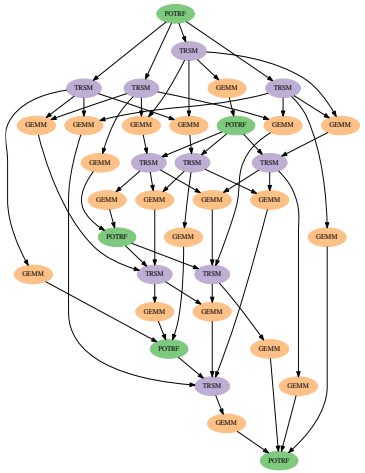


Different Programming Approaches

- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, . . .)
  - ☺ Perfect control ⇝ maximal achievable performance
  - ☺ Efficient granularity ⇝ advanced numerical features
  - ☹ Monolithic codes ⇝ hard to develop and maintain
  - ☹ Fixed scheduling ⇝ sensitive to variability
  - ☹ Hard and long to optimize ⇝ performance portability

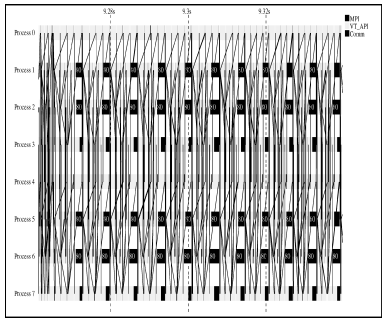


- Recent task-based programming models (PaRSEC, OmpSs, Charm++, StarPU, . . .)
  - ☺ Single, abstract programming model based on DAG
  - ☺ Runtime system responsible for dynamic scheduling
  - ☺ Portability of code and performance

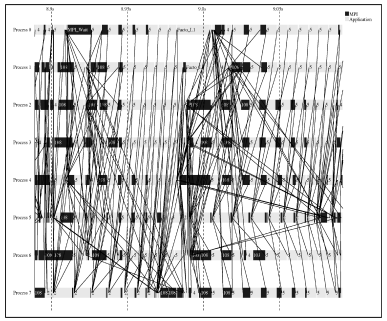


Different Programming Approaches

- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, . . .)
  - ☺ Perfect control ⇝ maximal achievable performance
  - ☺ Efficient granularity ⇝ advanced numerical features
  - ☹ Monolithic codes ⇝ hard to develop and maintain
  - ☹ Fixed scheduling ⇝ sensitive to variability
  - ☹ Hard and long to optimize ⇝ performance portability

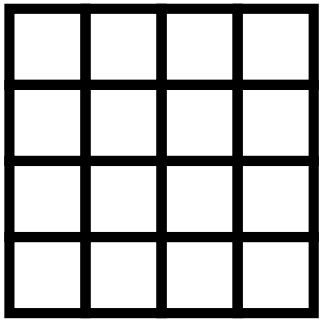


- Recent task-based programming models (PaRSEC, OmpSs, Charm++, StarPU, . . .)
  - ☺ Single, abstract programming model based on DAG
  - ☺ Runtime system responsible for dynamic scheduling
  - ☺ Portability of code and performance
  - ☹ Introducing runtime system overhead
  - ☹ Developing omnipotent runtime ⇝ new challenges

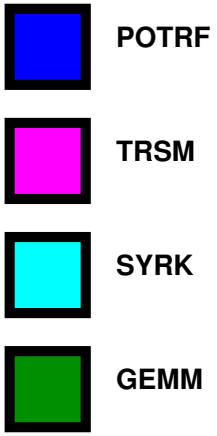


Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)

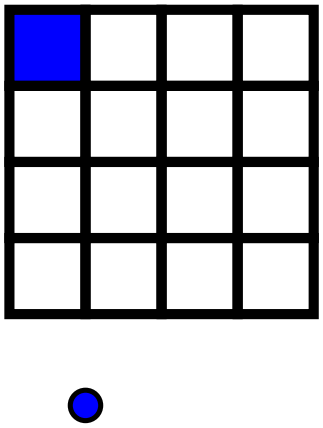


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```

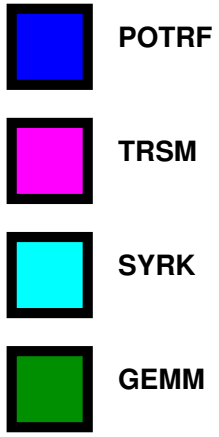


Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)

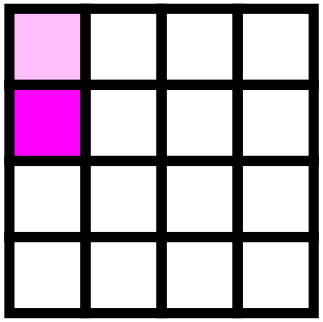


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```

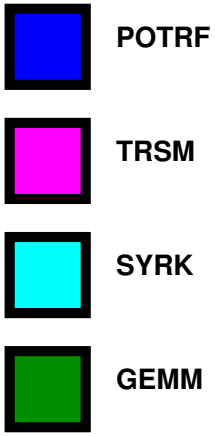


Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)

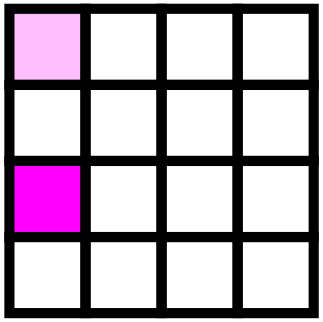


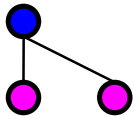
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```



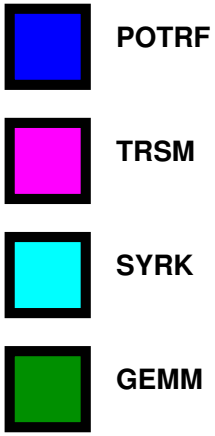
Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)



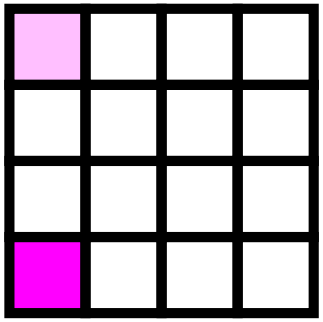


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```

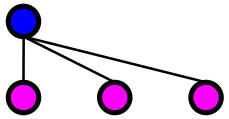


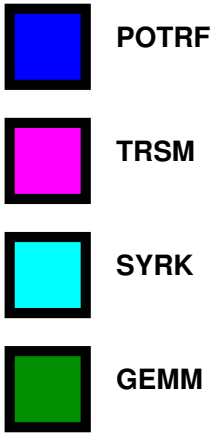
Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)



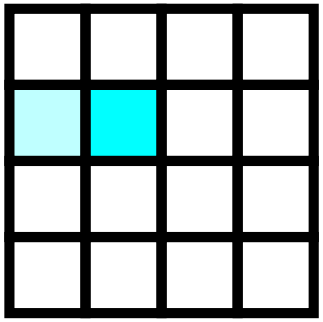
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```



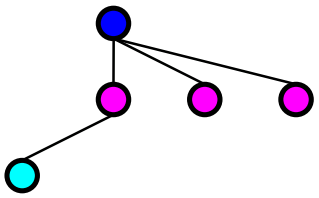


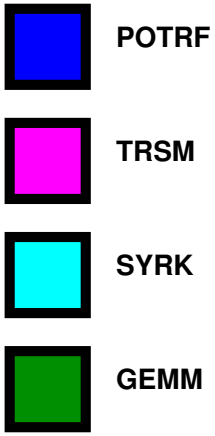
Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)

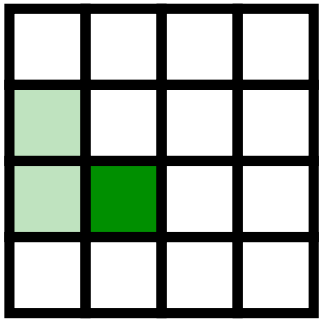


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```

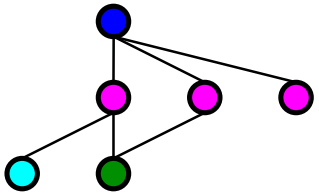


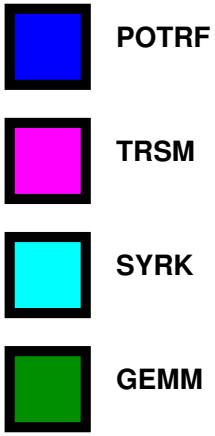


Tiled Cholesky Algorithm
using Sequential Task Flow (STF)



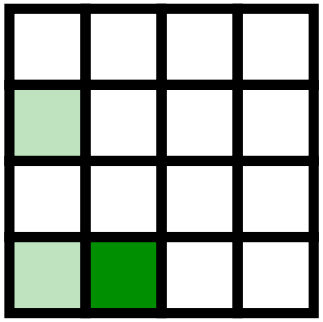
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```



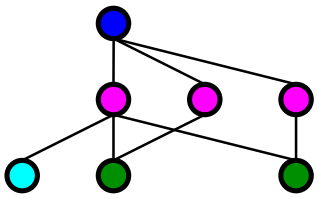


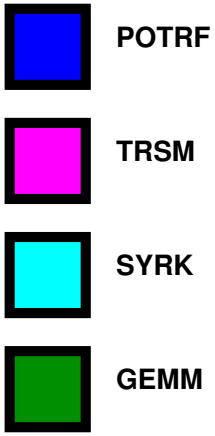
Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)



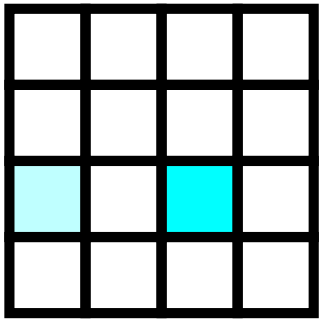
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```



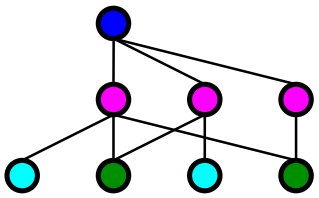


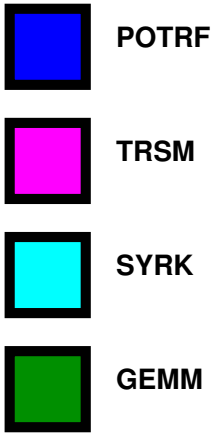
Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)



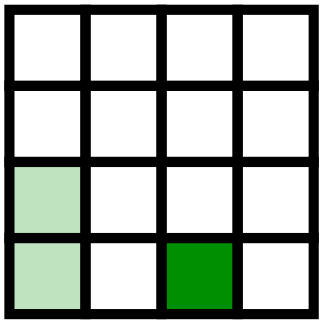
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```



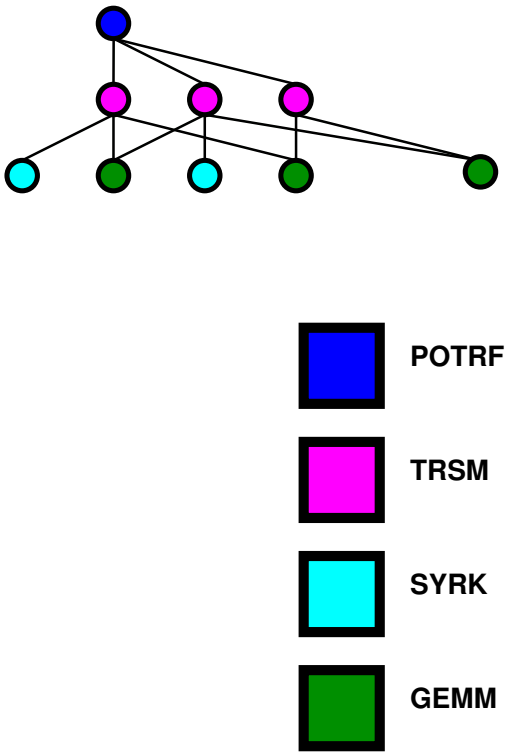


Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)

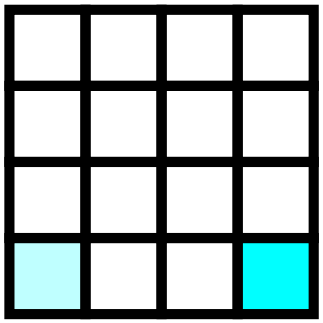


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```

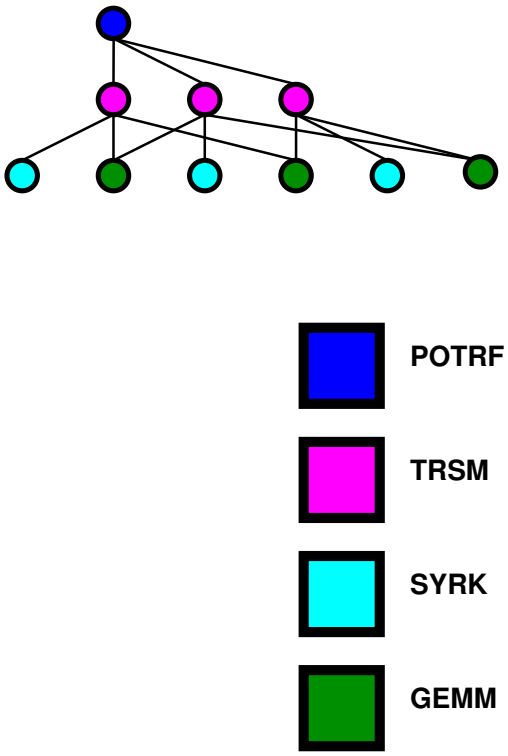


Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)

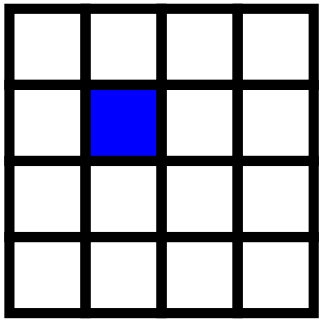


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```

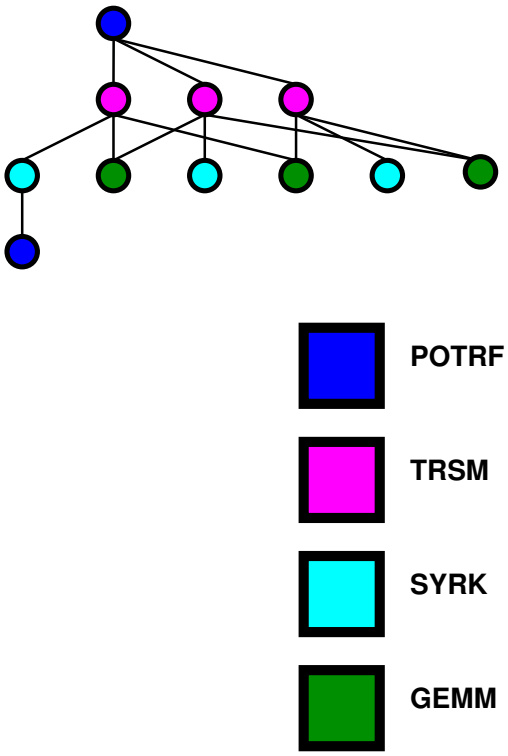


Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)

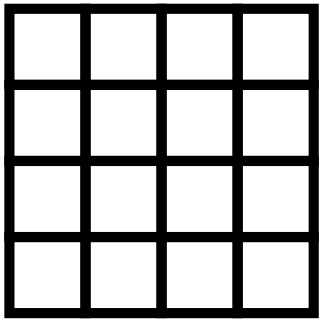


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```

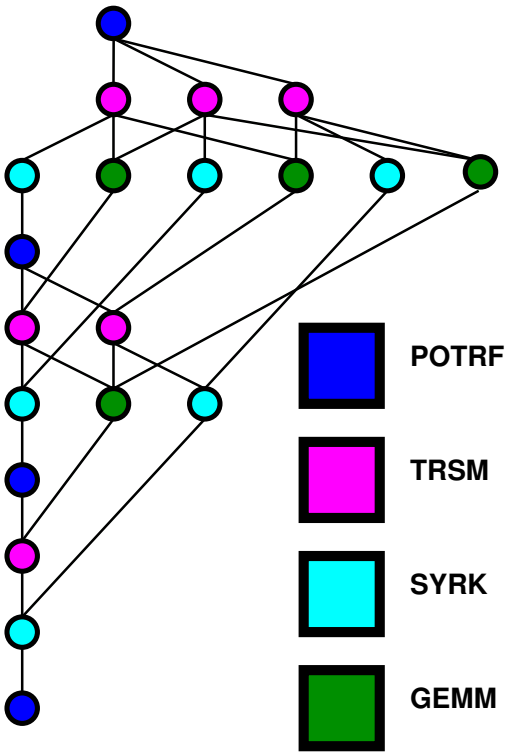


Task-based Programming Paradigm

Tiled Cholesky Algorithm
using Sequential Task Flow (STF)



```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
wait();
```



Need Regular Performance Evaluation

Native experiments

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

Model, equations, theory

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

Need Regular Performance Evaluation

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

Research Statement

Is it possible to perform a clean, coherent, reproducible study of HPC applications executed on top of dynamic task-based runtime systems, using simulation?

Research Statement

Is it possible to perform a clean, coherent, reproducible study of HPC applications executed on top of dynamic task-based runtime systems, using simulation?

Research Statement

Is it possible to perform a clean, coherent, reproducible study of HPC applications executed on top of dynamic task-based runtime systems, using simulation?

Outline

StarPU and SimGrid

StarPU (Inria Bordeaux)

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

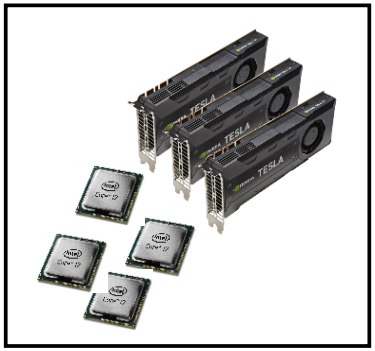
SimGrid (Inria Grenoble, Lyon, Rennes, ...)

- 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

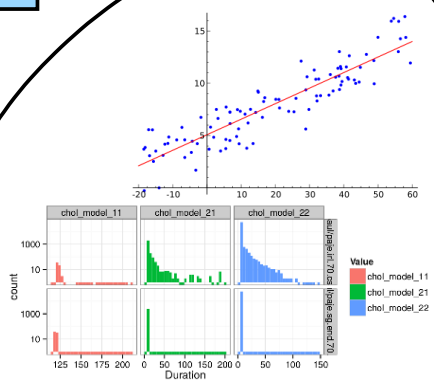
Same approach could be applicable to any task-based runtime system

Devised Workflow: StarPU + SimGrid

Calibration



StarPU

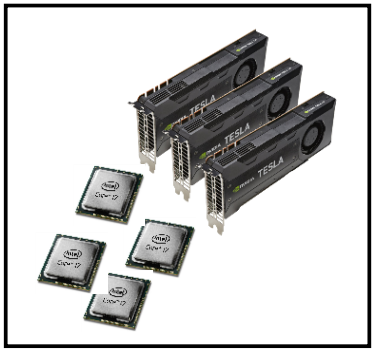


Performance Profile

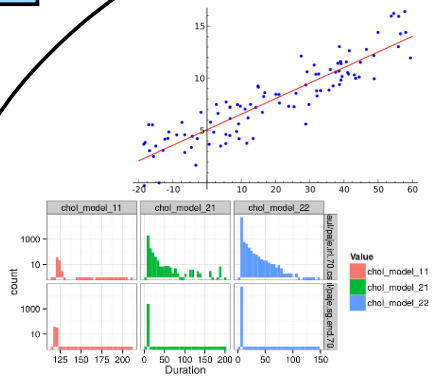
Run once!

Devised Workflow: StarPU + SimGrid

Calibration



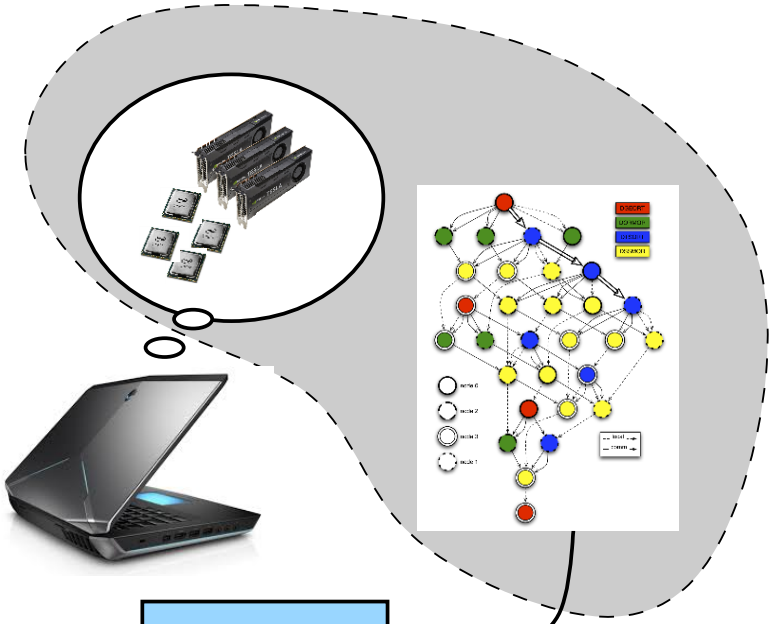
StarPU



Performance Profile

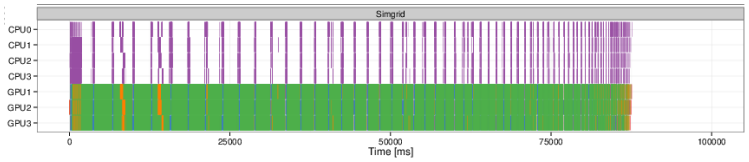
Run once!

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

Modeling Runtime System

Simulation delays (increasing simulated time)

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

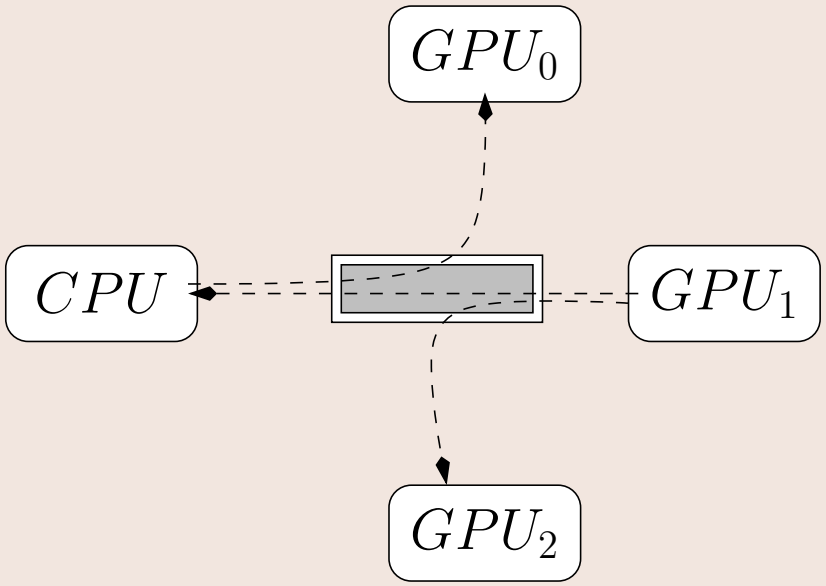
Example for CUDA memory allocation in StarPU

```
...
#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);
...
```

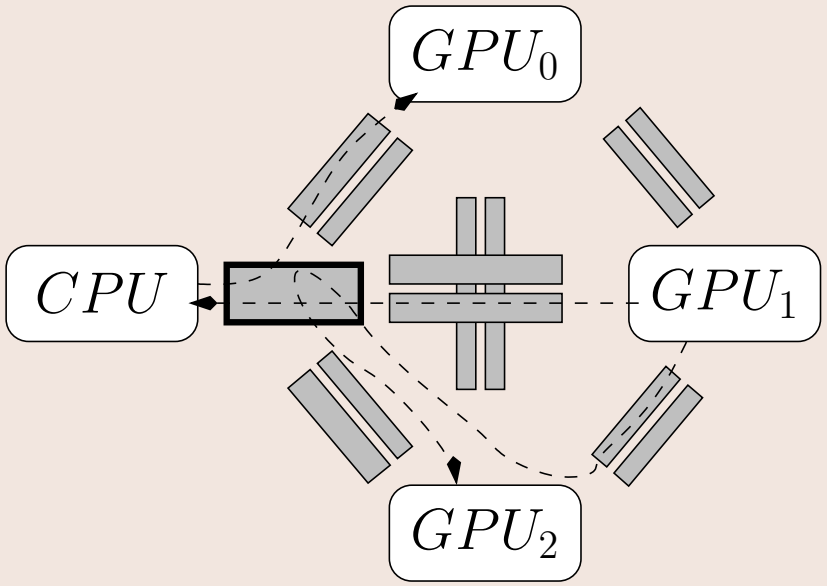
Modeling Communication

- Components of hybrid platforms have differing characteristics
- Correctly modeling their communication is of primary importance
- Built on exhaustively validated existing SimGrid network models

Flexible flow-based contention model



(a) Fatpipe (crude) model

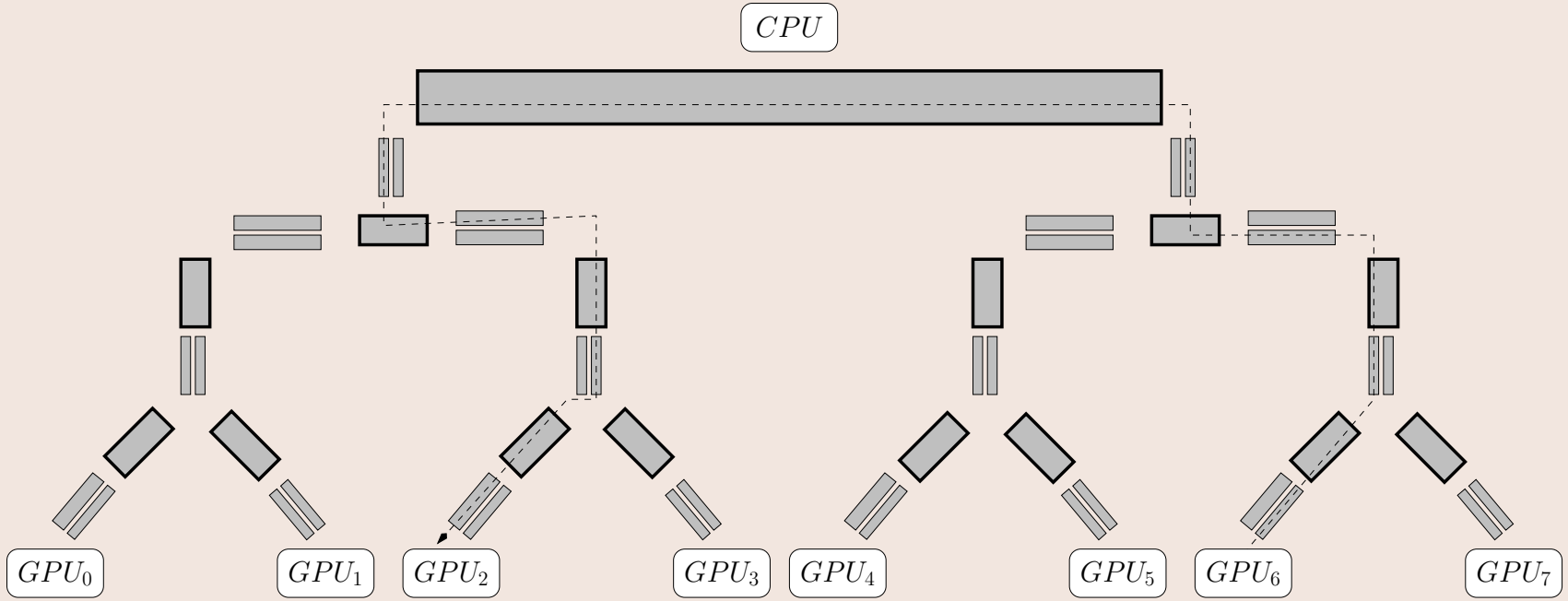


(b) Complete graph (pragmatic) model

Modeling Communication

- Components of hybrid platforms have differing characteristics
- Correctly modeling their communication is of primary importance
- Built on exhaustively validated existing SimGrid network models

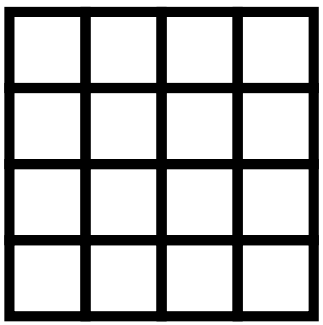
Flexible flow-based contention model

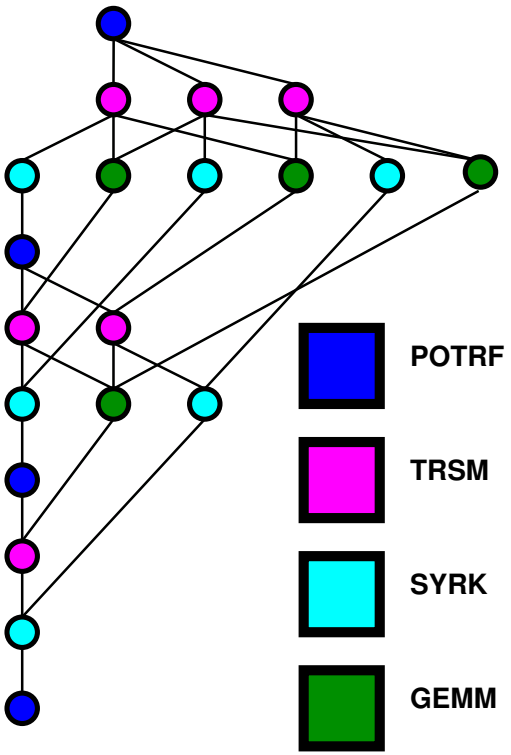


(c) Treelike (elaborated) model

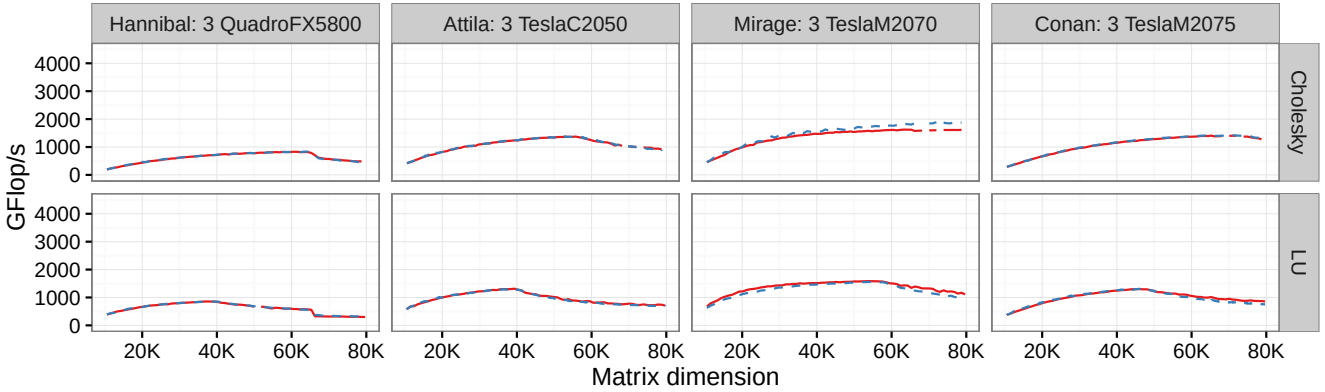
Modeling Computation

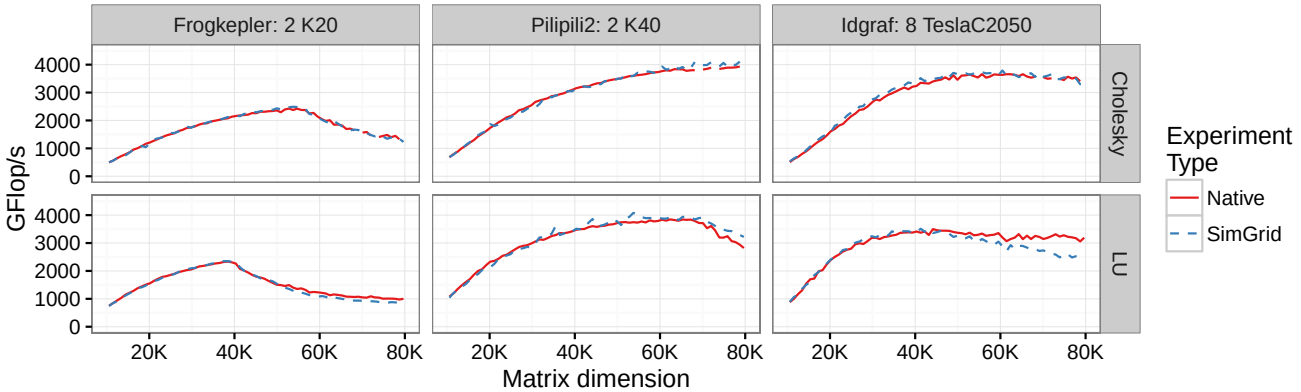
- Actual computation results irrelevant
  $\rightsquigarrow$ only computation time matters
- Task = Kernel for task-based paradigm
- Execution of tasks replaced by simulation delays
- Average duration for regular kernels
- Additional techniques to optionally account for variability





Overview of Simulation Accuracy



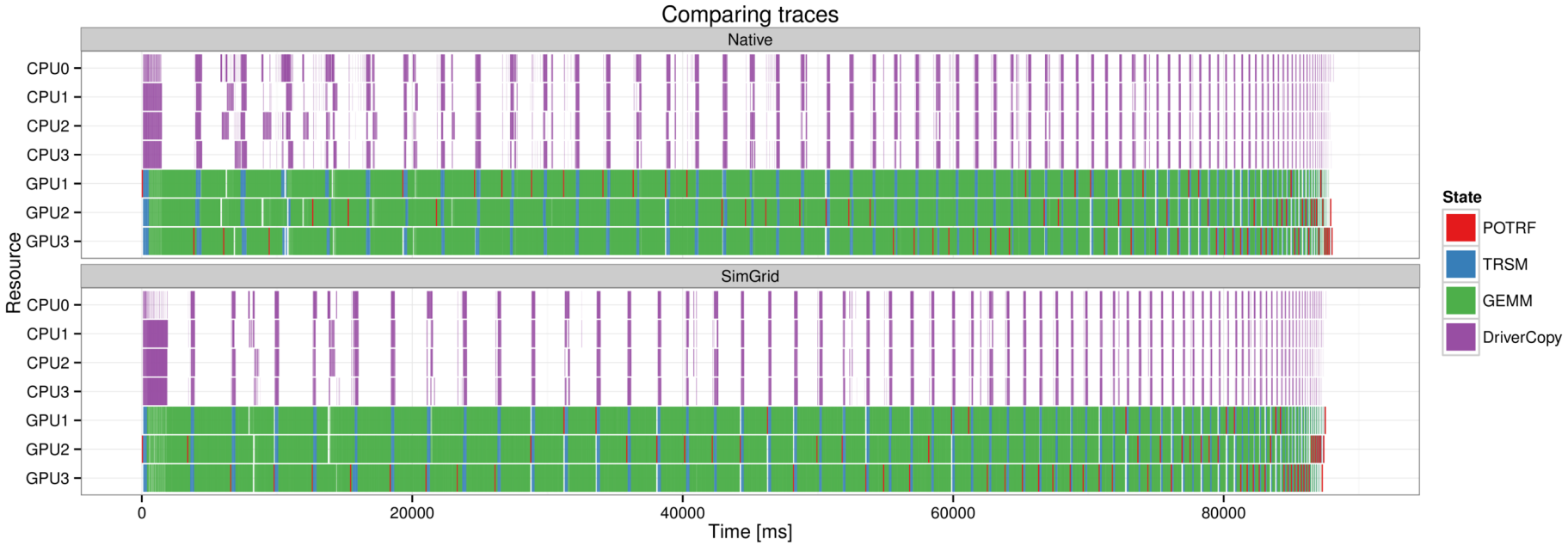


Publications

[1] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Méhaut. Faithful Performance Prediction of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. *Concurrency and Computation: Practice and Experience*, page 16, May 2015.

[2] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Méhaut. Modeling and Simulation of a Dynamic Task-Based Runtime System for Heterogeneous Multi-core Architectures. In *Euro-par - 20th International Conference on Parallel Processing*, Euro-Par 2014, LNCS 8632, pages 50–62, Porto, Portugal, Aug. 2014.

Overview of Simulation Accuracy



Publications

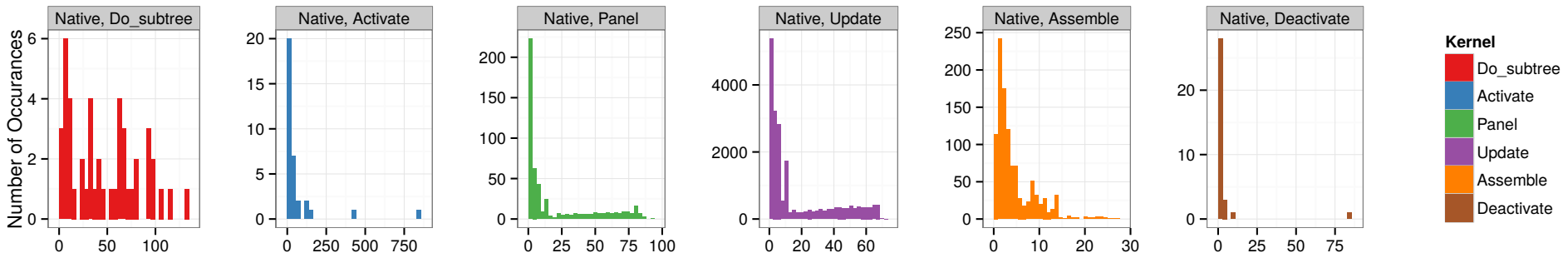
[1] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Méhaut. Faithful Performance Prediction of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. *Concurrency and Computation: Practice and Experience*, page 16, May 2015.

[2] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Méhaut. Modeling and Simulation of a Dynamic Task-Based Runtime System for Heterogeneous Multi-core Architectures. In *Euro-par - 20th International Conference on Parallel Processing*, Euro-Par 2014, LNCS 8632, pages 50–62, Porto, Portugal, Aug. 2014.

Outline

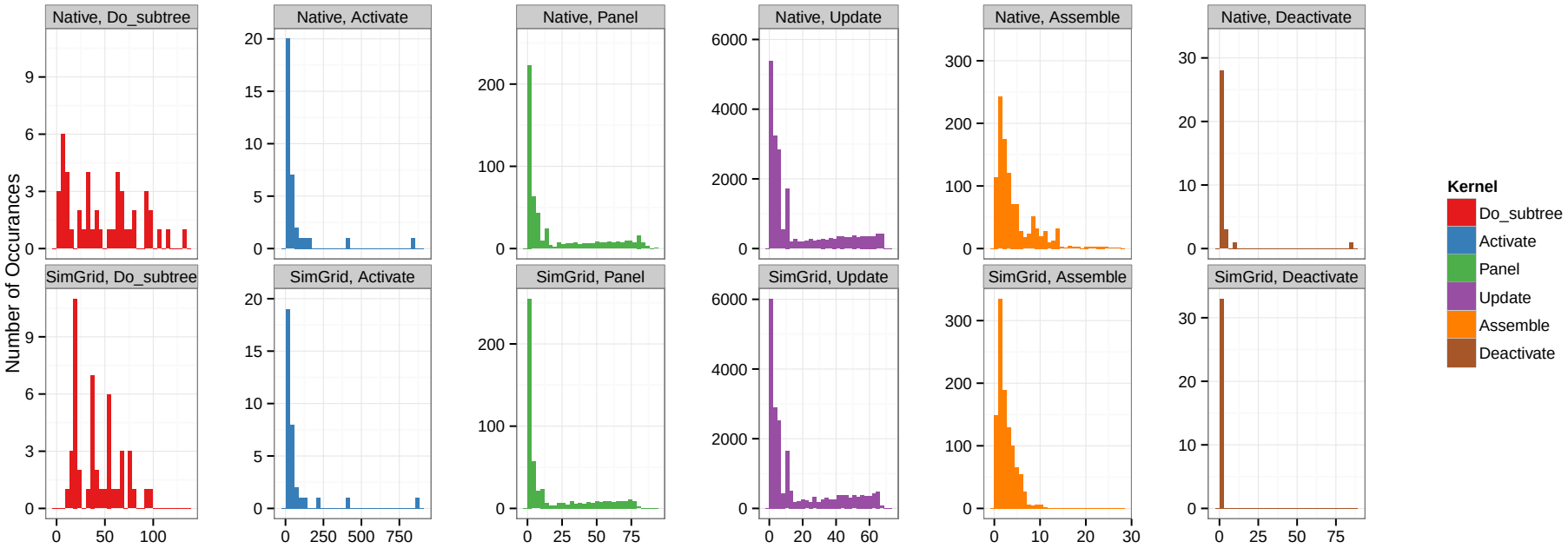
Difference Between Regular and Irregular Kernels

- Regular kernels use always the same block size
  $\rightsquigarrow$ duration is (relatively) stable
- Irregular kernel durations depend on their input parameters
  $\rightsquigarrow$ need more than simple average values



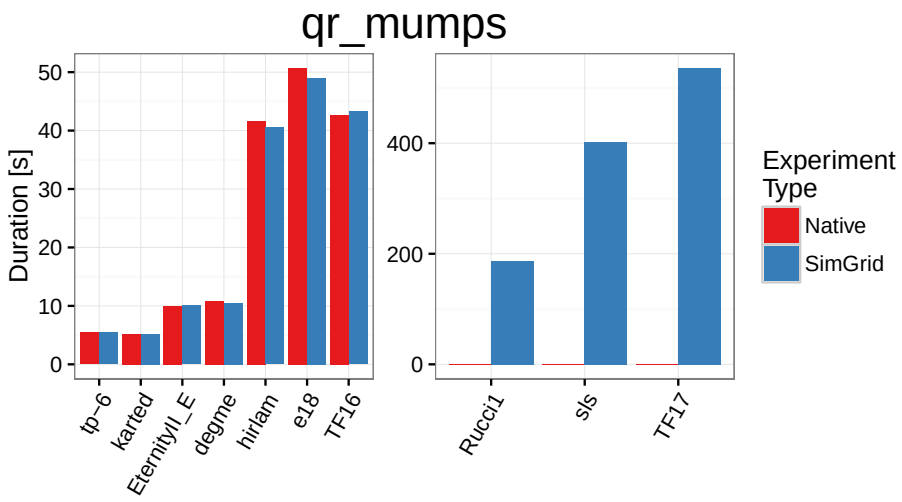
Modeling Duration of Complex Computation Kernels

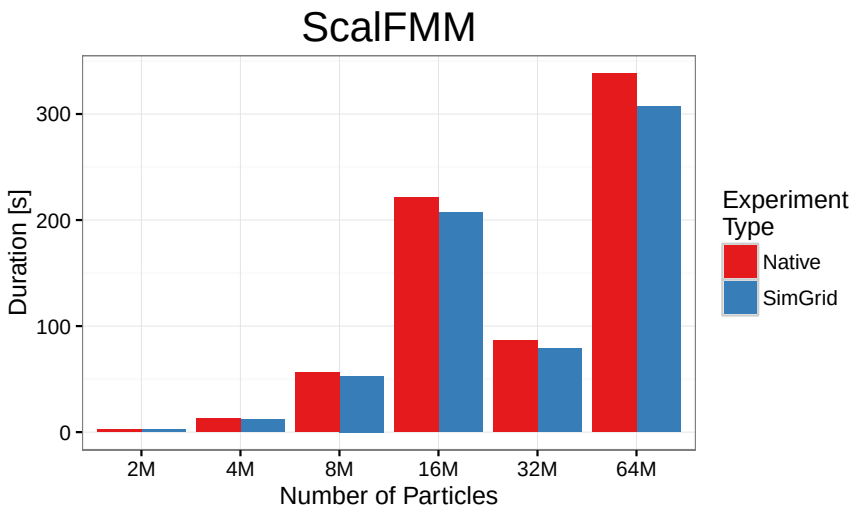
- Using statistical analysis and multiple linear regression
- Extended StarPU to automatically support such models
- Kernel duration estimations useful for both simulation and native executions (scheduling)



Simulating Irregular Numerical Libraries

- Chameleon solver: dense linear algebra library
- qr_mumps solver: MUMPS multi-frontal factorization
- ScalFMM library: simulating N-body interactions using the FMM
- QDWH solver: QR-based Dynamically Weighted Halley (ongoing)





Publication

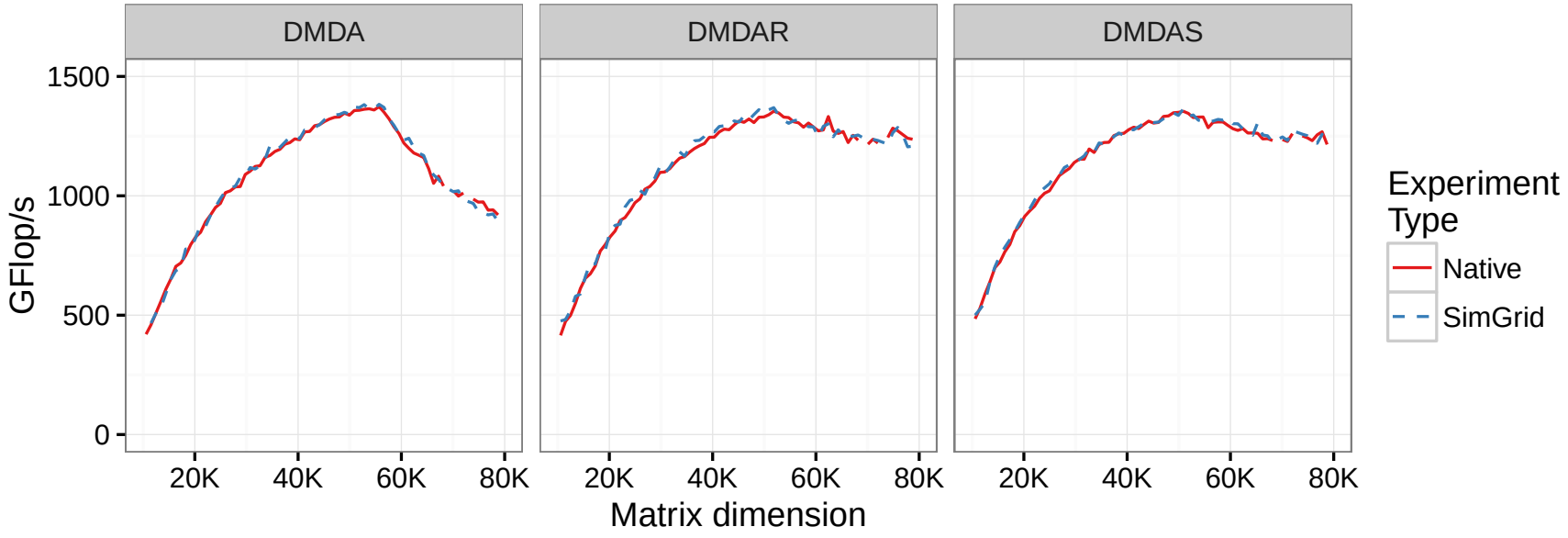
[3] L. Stanisic, E. Agullo, A. Buttari, A. Guermouche, A. Legrand, F. Lopez, and B. Videau. Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers. *Parallel and Distributed Systems (ICPADS)*, Dec. 2015.

[4] E. Agullo, B. Bramas, O. Coulaud, L. Stanisic, and S. Thibault. Modeling Irregular Kernels of Task-based codes: Illustration with the Fast Multipole Method. *submitted to Transaction on Mathematical Software (TOMS)*, [Research Report] 9036, INRIA Bordeaux. pp.35. Feb. 2017.

Outline

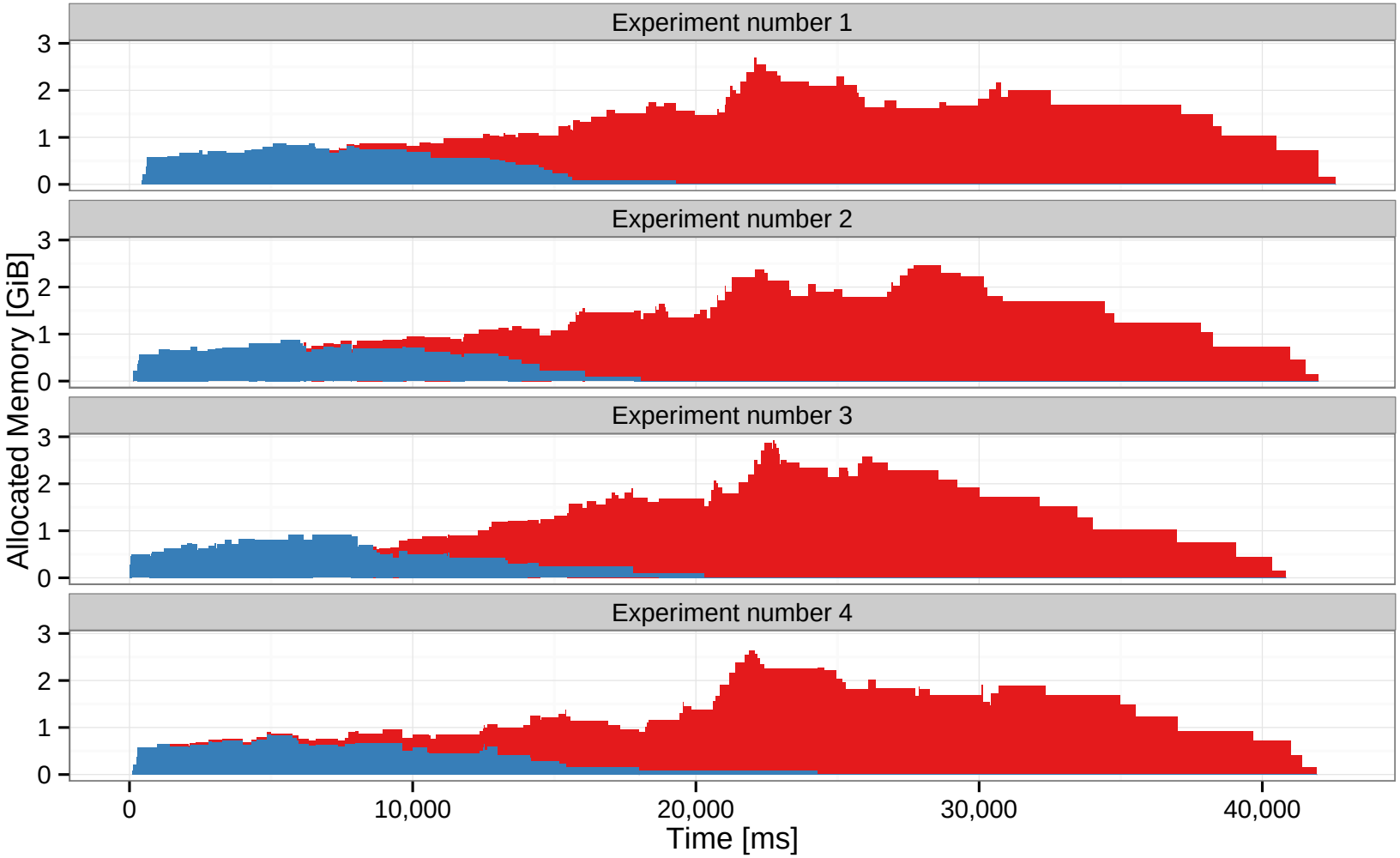
Comparing Different Schedulers

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



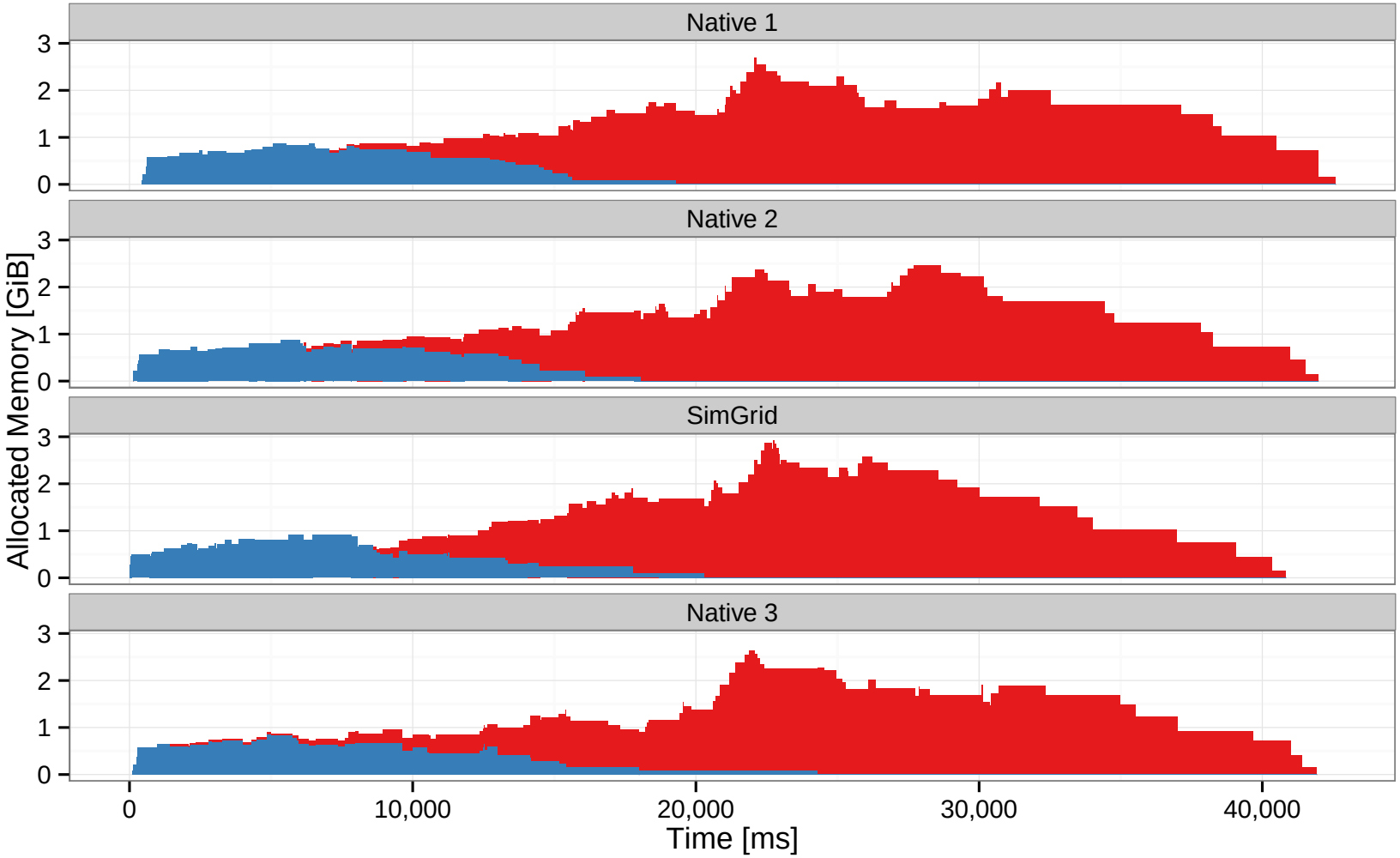
Studying Memory Consumption

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



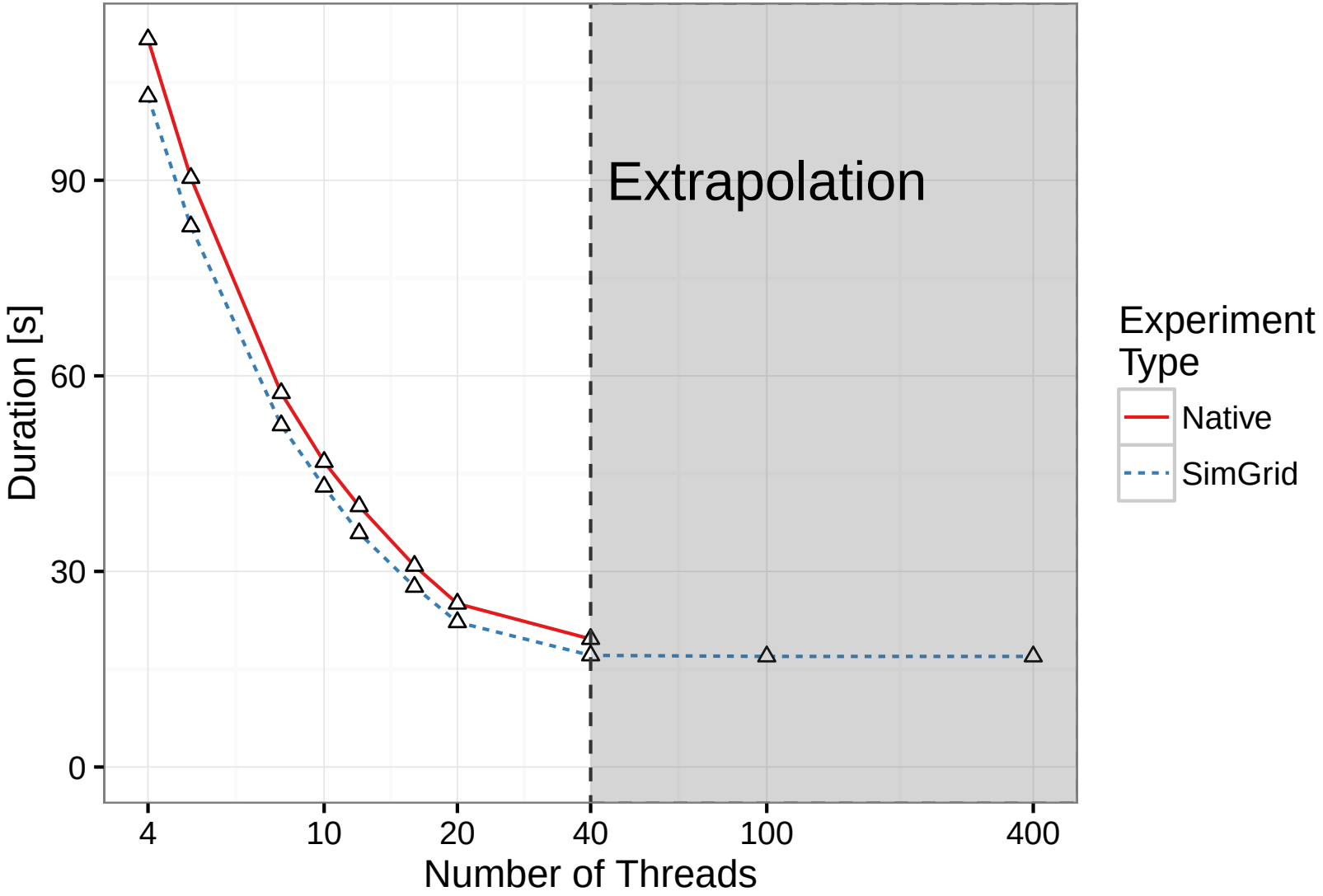
Studying Memory Consumption

- 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



Outline

Achievements

- Works great for small hybrid setups with dense, sparse and FMM StarPU applications
- 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

Challenges of Experimental Studies in HPC

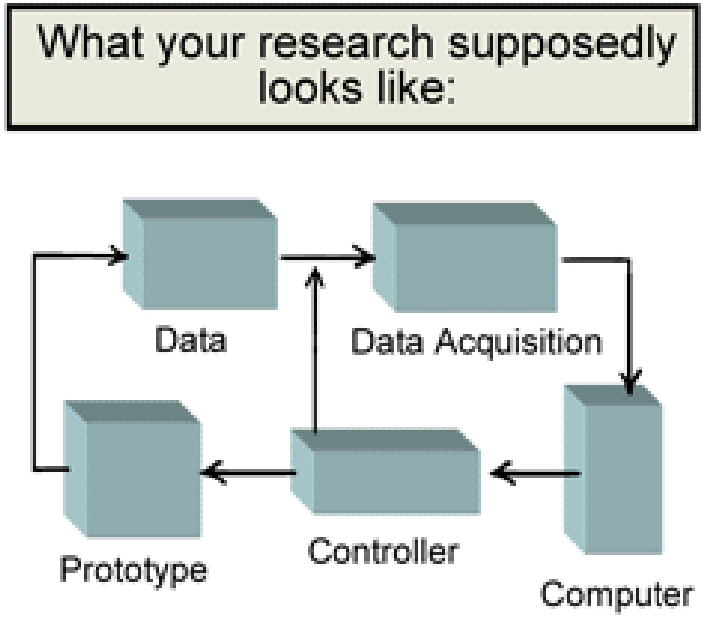


Figure 1. Experimental Diagram

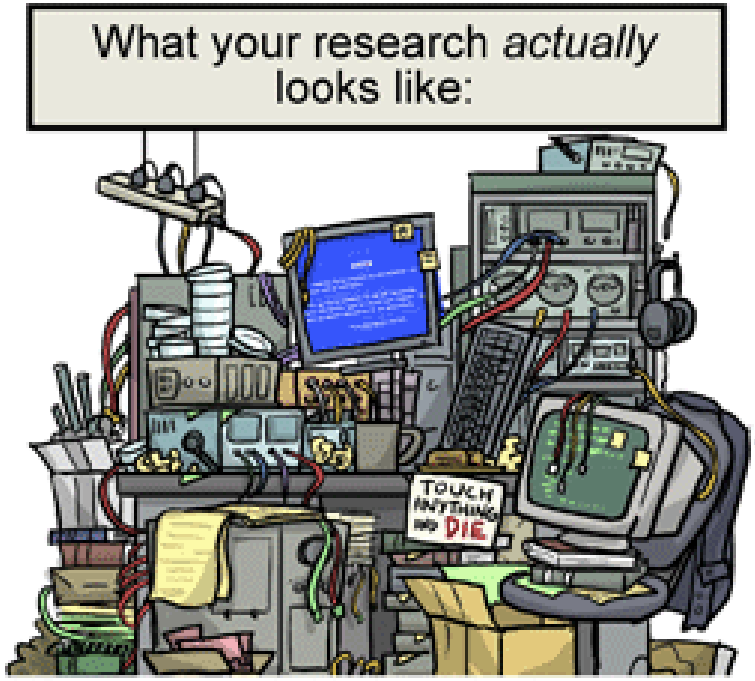
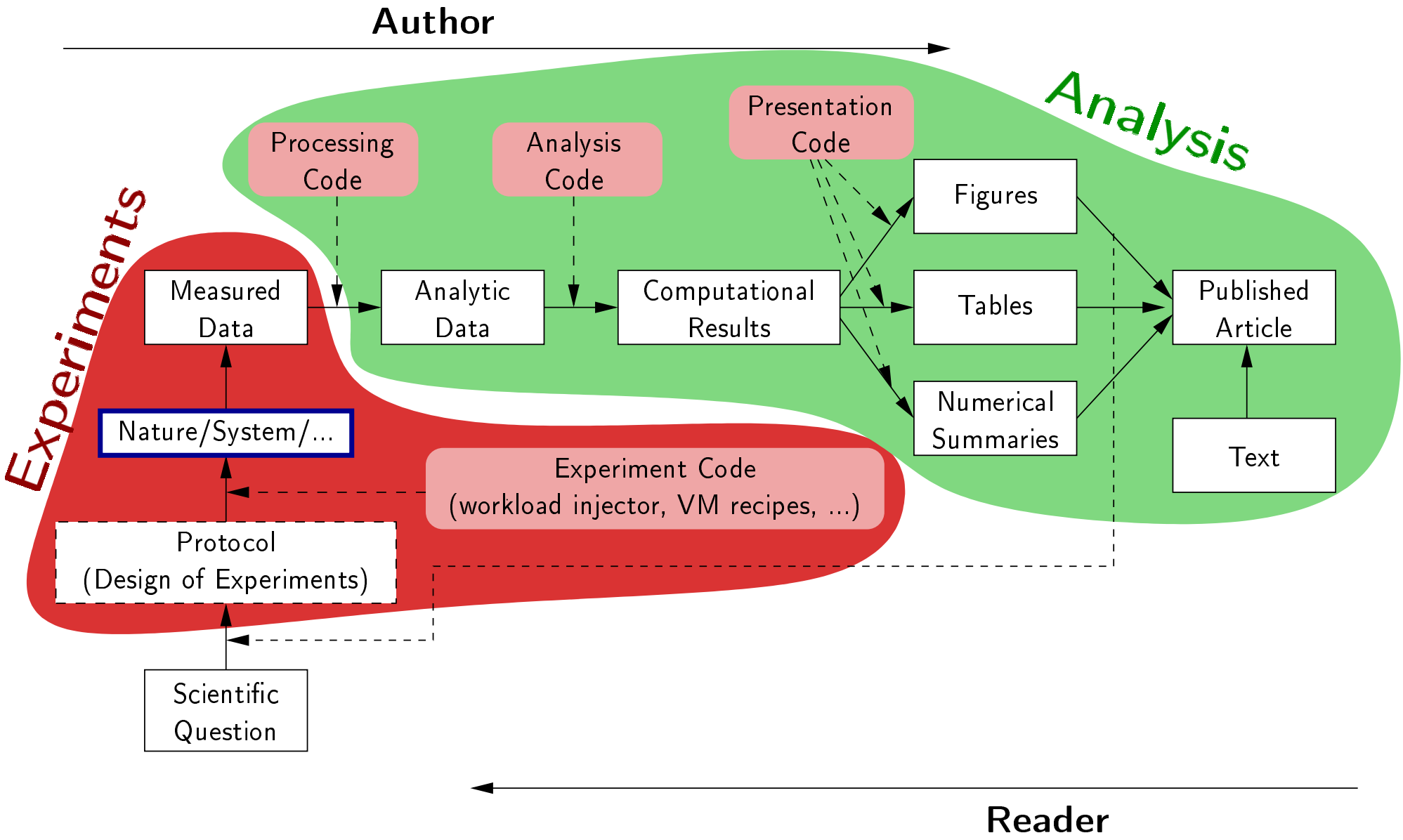


Figure 2. Experimental Mess

- Large, hybrid, prototype hardware/software (hard to control)
- Costly experiments with numerous parameters
- Non-deterministic executions (overall duration, traces, . . .)
- Workflows specific to the studies (hardly applicable in general)

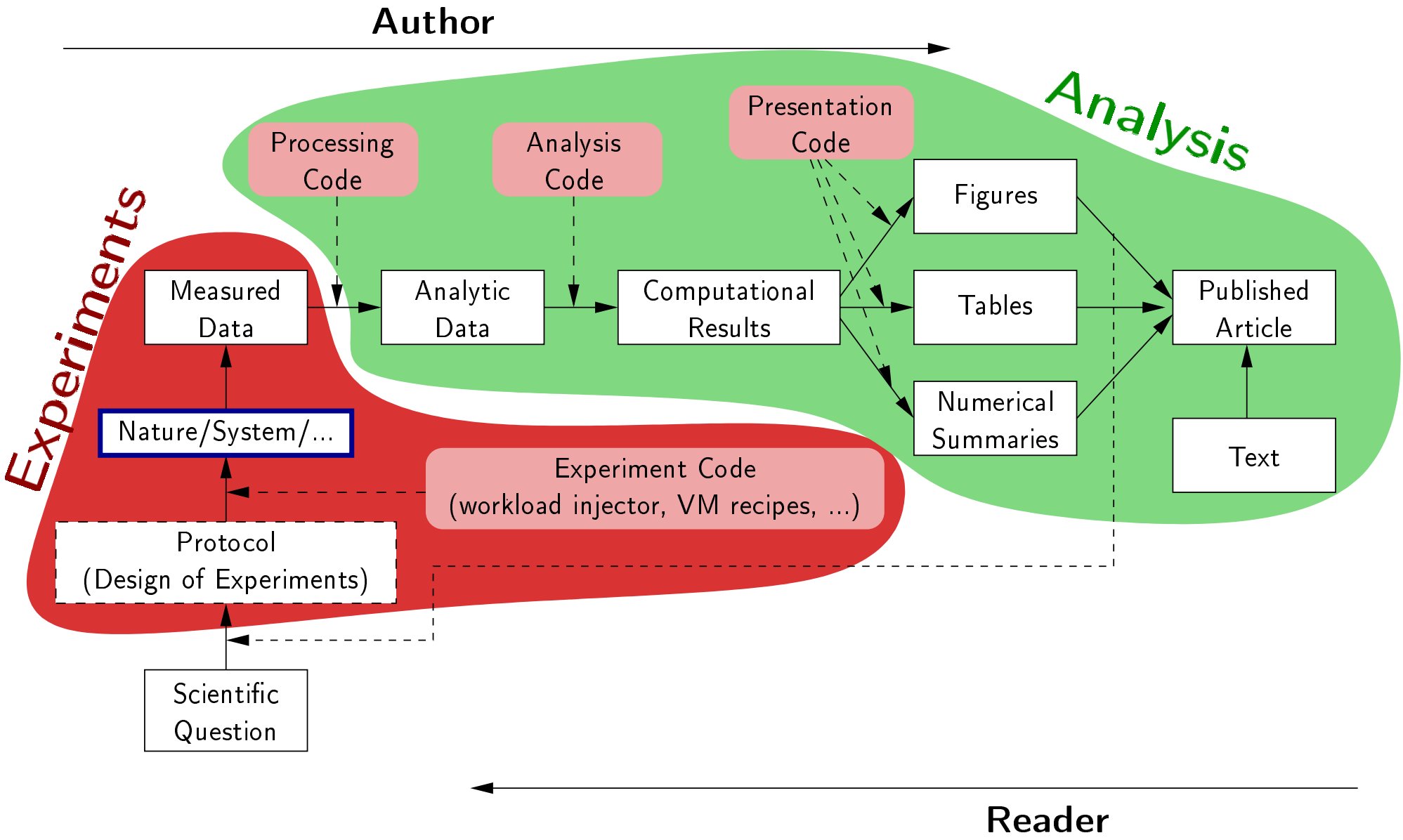
$\rightsquigarrow$ difficult to make research results reproducible

Reproducible Research: Trying to Bridge the Gap



Inspired by Roger D. Peng's lecture on reproducible research, May 2014

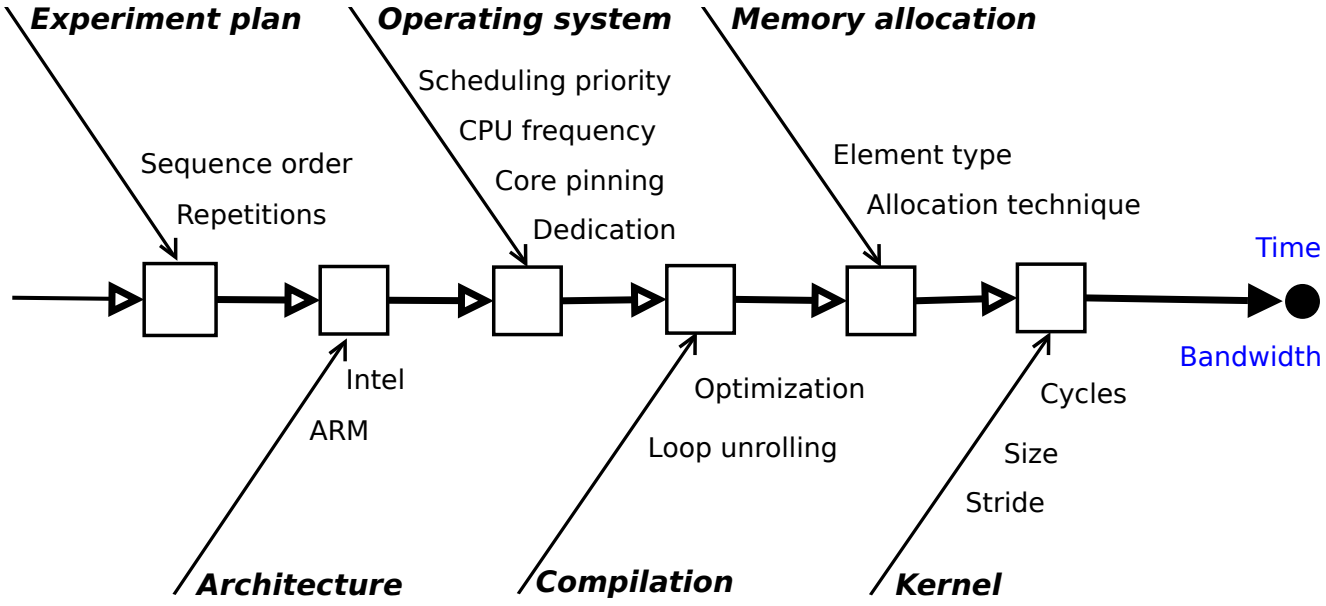
Reproducible Research: Trying to Bridge the Gap



Our approach: use a lightweight combination of existing generic tools

Experiments

- Full control of design of experiments
- Automatize process
- Gather as much useful meta-data as possible for each experiment

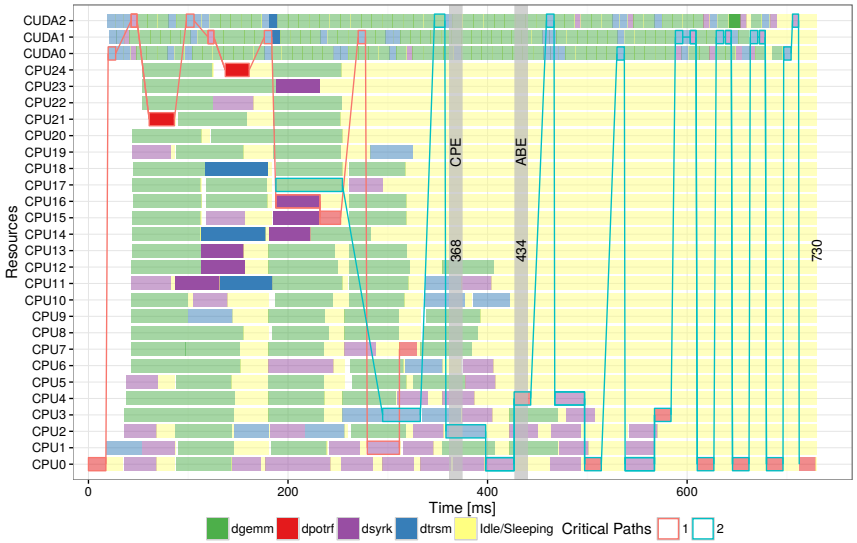


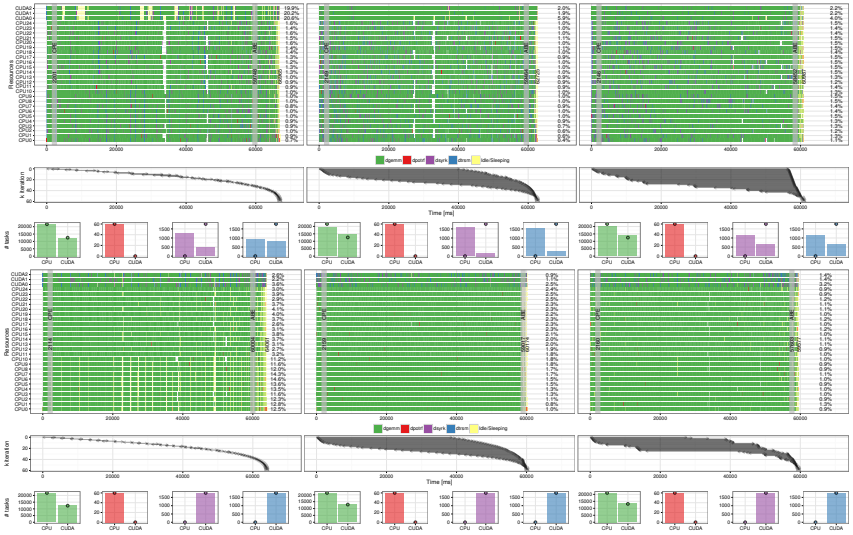
Publication

[5] L. Stanisic, L. M. Schnorr, A. Degomme, F. Heinrich, A. Legrand, and B. Videau. Characterizing the Performance of Modern Architectures Through Opaque Benchmarks: Pitfalls Learned the Hard Way. *submitted to International Workshop on Reproducibility in Parallel Computing (REPPAR)*, 2017.

Analysis

- Write papers/reports with completely reproducible analysis
- Rely on literate programming tools (IPython/Jupiter, Orgmode)
- Modular scripting approach (shell + R)



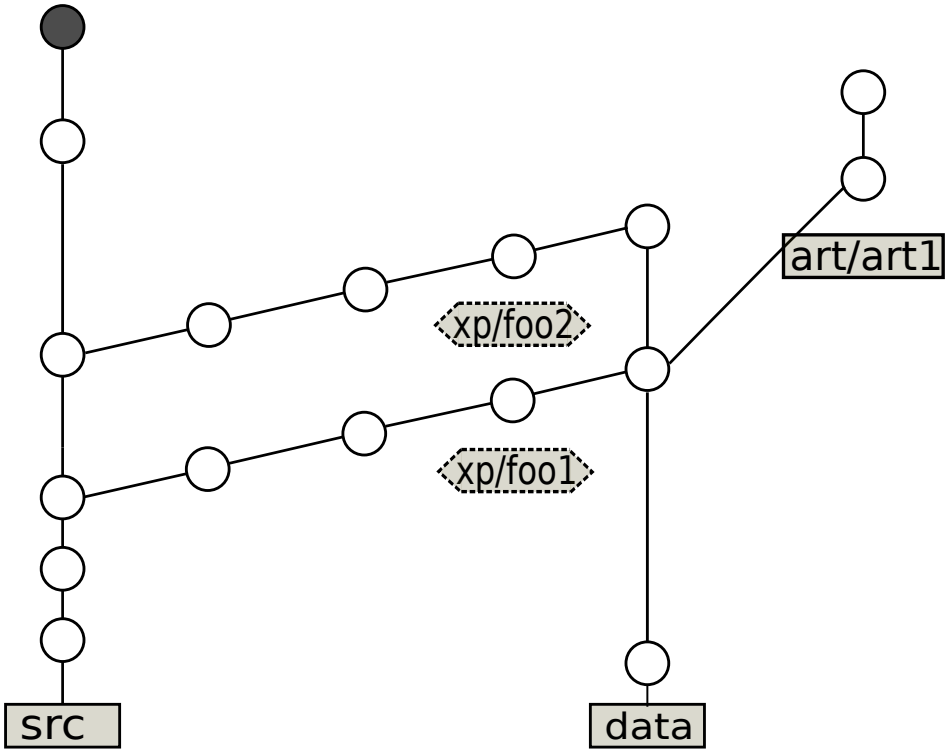


Publication

[6] V. G. Pinto, L. Stanisic, A. Legrand, L. M. Schnorr, and S. Thibault. Analyzing Dynamic Task-Based Applications on Hybrid Platforms: An Agile Scripting Approach. *3rd Workshop on Visual Performance Analysis (VPA)*, Nov 2016, Salt Lake City, United States

Workflow for the Whole Research Process

- Documentation and experimentation journal (laboratory notebook)
- Unique Git branching system for better project history



Publications

[7] L. Stanisic, A. Legrand, and V. Danjean. An Effective Git And Org-Mode Based Workflow For Reproducible Research. *ACM SIGOPS Operating Systems Review*, 49:61 – 70, 2015. Special Topic: Repeatability and Sharing of Experimental Artifacts.
[8] L. Stanisic and A. Legrand. Effective Reproducible Research with Org-Mode and Git. In *1st International Workshop on Reproducibility in Parallel Computing*, Porto, Portugal, Aug. 2014.

Achievements

Design:

- Original approach based on well-known tools
- Helps filling the author/reader gap in our context
- Applicable and extendable to other research fields

Application:

- Used this approach for many studies, presentations and papers
- Efficiently handled ≈10,000 experiments (40GiB) and ≈2,000 commits

Evangelism:

- Our closest colleagues successfully adopting this approach
- Presented our methods on numerous occasions (RR webinar, conferences, workshops, ANR project meetings, . . .)

Outline

Experience

- Modeling, simulation and performance evaluation
- Methodology for reproducible research
- Statistical analysis, visualizations

- Code and performance debugging and optimizations
- Working with large, hybrid, prototype hardware and software
- Contributions to many large code projects:

StarPU (C)	SimGrid (C/C++)
qr_mumps (C/Fortran)	ScalFMM (C++)
Chameleon (C/Fortran)	

Summary

Regular algorithms
Dynamic task-based HPC applications
Research methodology

Real-life applications
Collaboration with other domain experts

Benchmarks
Basic modeling

Numerical (irregular) libraries
Performance optimization
Large scale executions

Summary

Regular algorithms
Dynamic task-based HPC applications
Research methodology

Real-life applications
Collaboration with other domain experts

Benchmarks
Basic modeling

Numerical (irregular) libraries
Performance optimization
Large scale executions

Thank you!

http://mescal.imag.fr/membres/luka.stanisic/

Ongoing Research: Multiple Nodes

- StarPU-MPI + SimGrid for large scale distributed memory studies
- Requires combining two modules of SimGrid framework
  $\rightsquigarrow$ technically challenging, need to rewrite internals
- Large number of resources, kernels and communications in parallel
  $\rightsquigarrow$ need to optimize simulator performance
- Multiple network models (PCI bus and Ethernet/Infiniband)
  $\rightsquigarrow$ contention harder to model