MORSE

Matrices Over Runtime Systems @ Exascale

SOLHAR, Bordeaux, 2016 January 25

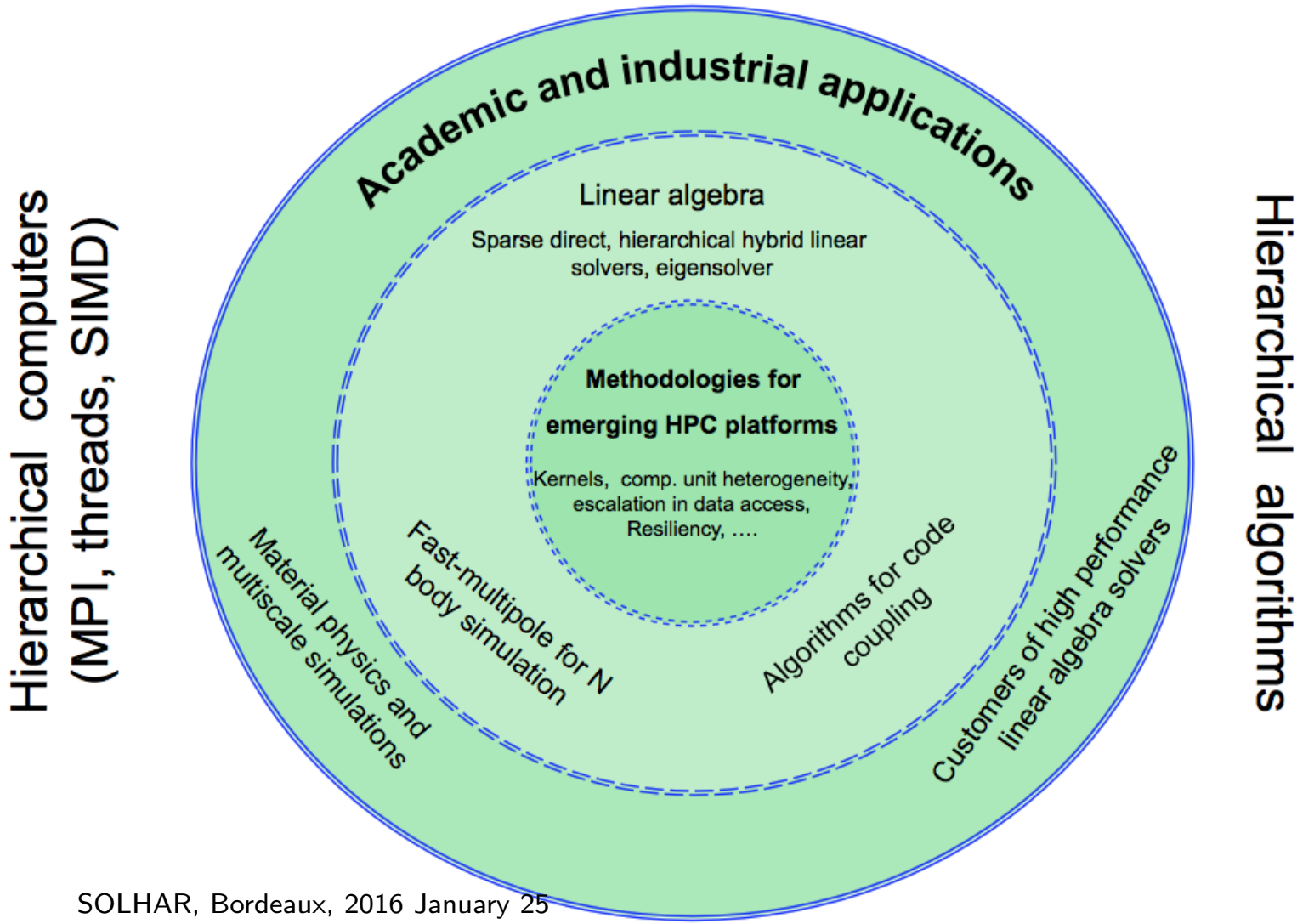
HiePACS - RealOpt - Runtime
Kaust - UCD - UTK
INRIA Bordeaux Sud-Ouest

Outline

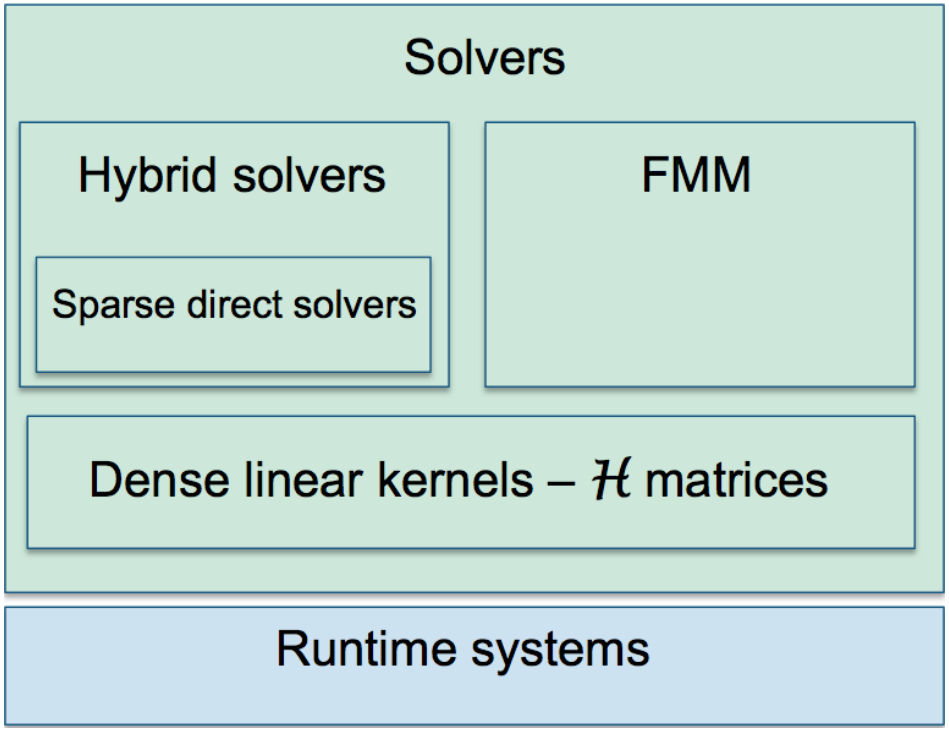
HiePACS (Leader: Luc Giraud)

HiePACS objectives: Contribute to the design of effective tools for frontier simulations arising from challenging research and industrial multi-scale applications towards exascale computing

HiePACS: scientific structure



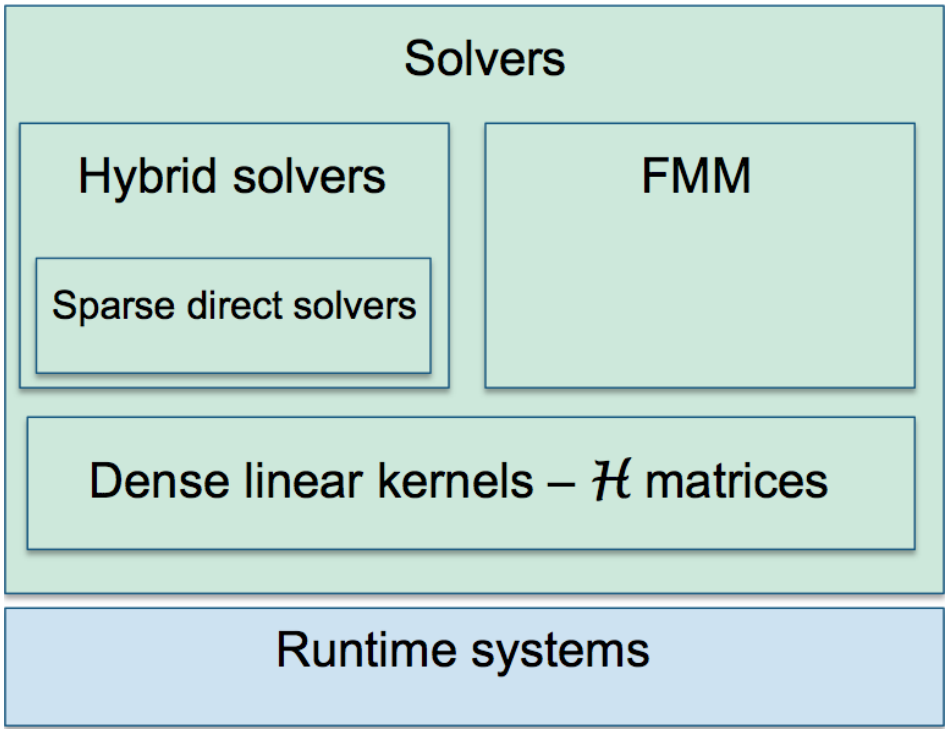
A view of HiePACS solvers



Chameleon: dense linear solver

- Tile algorithms: BLAS 3, some BLAS 2, LAPACK One-Sided, Norms
- Supported runtimes: Quark and StarPU, (PaRSEC soon)
- Ability to use cluster of heterogeneous nodes:
  - MPI+threads, CPUs (BLAS/LAPACK)+GPUs (cuBLAS/MAGMA)

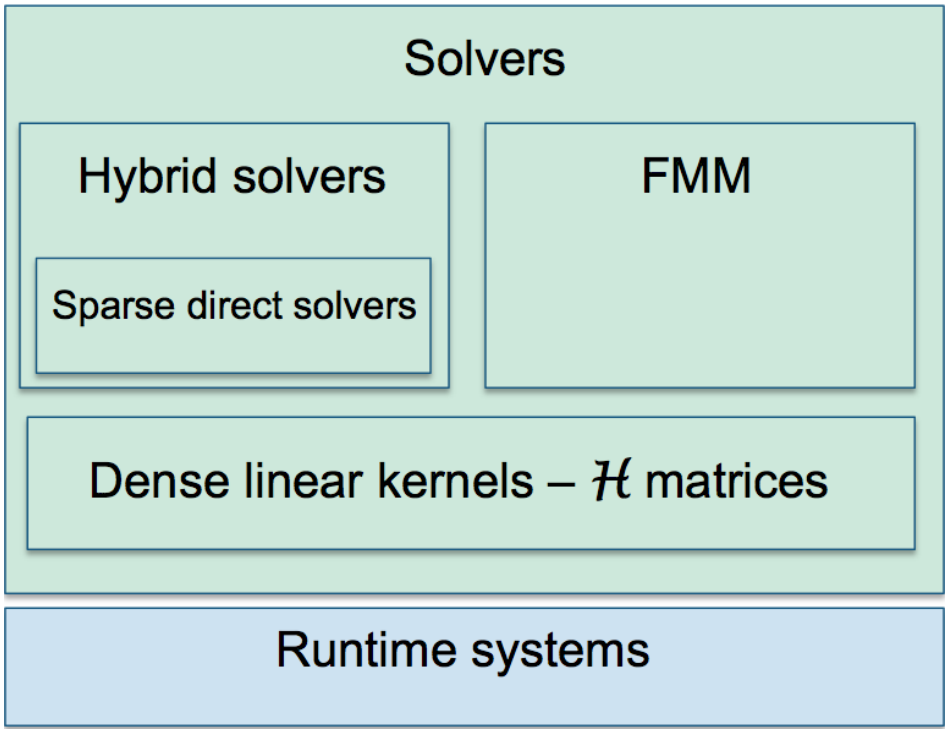
A view of HiePACS solvers



PaStiX: sparse linear solver

- **LL^T**, **LDL^T**, and **LU**, with static pivoting, supernodal approach
- Native version: MPI+threads
- Versions with runtimes: on top of PaRSEC or StarPU

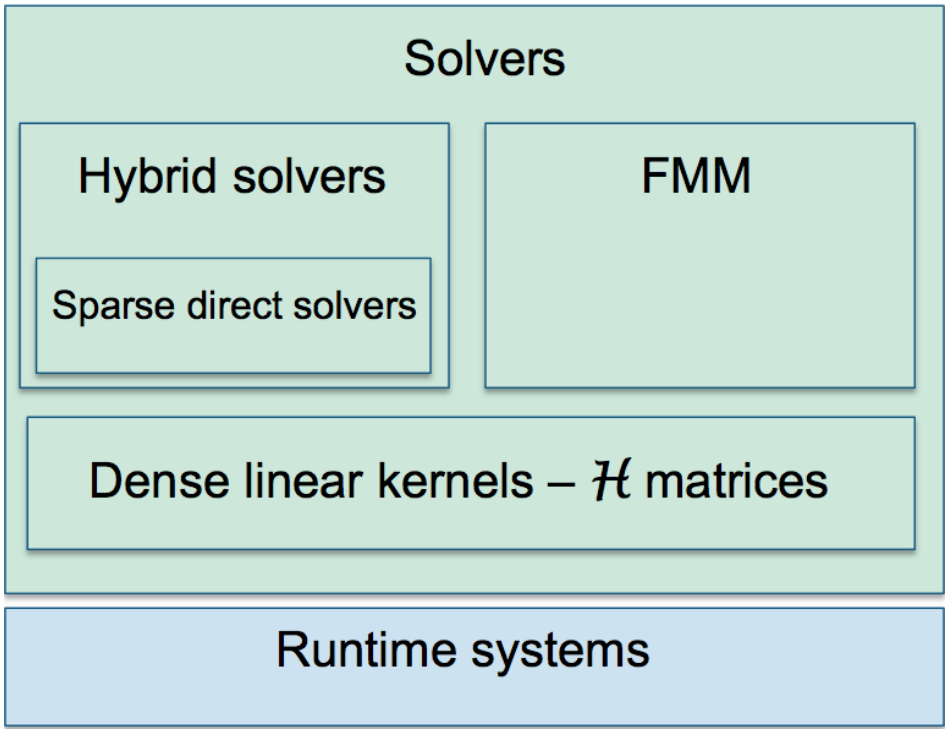
A view of HiePACS solvers



MaPHyS: hybrid direct/iterative sparse linear solver

- Solves $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a square non singular general matrix
- Native version: MPI+PaStiX/MUMPS+BLAS/LAPACK
- Do not support runtimes for now, work in progress

A view of HiePACS solvers



ScalFMM: scalable fast multipole methods

- Simulate N-body interactions using the Fast Multipole Method based on interpolation (Chebyshev or Lagrange)
- Native version: MPI+OpenMP+BLAS+FFTW
- Runtimes version: StarPU, OpenMP4 → StarPU (ADT K'STAR)

MORSE

Matrices Over Runtime Systems @ Exascale

INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

Inria
INVENTEURS DU MONDE NUMÉRIQUE

KAUST

University of Colorado Denver

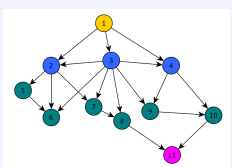
Linear algebra

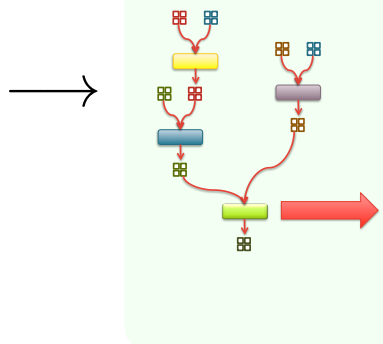
$$AX = B$$

Sequential-Task-Flow

```
for (j = 0; j < N; j++)
  Task (A[j]);
```

Direct Acyclic Graph

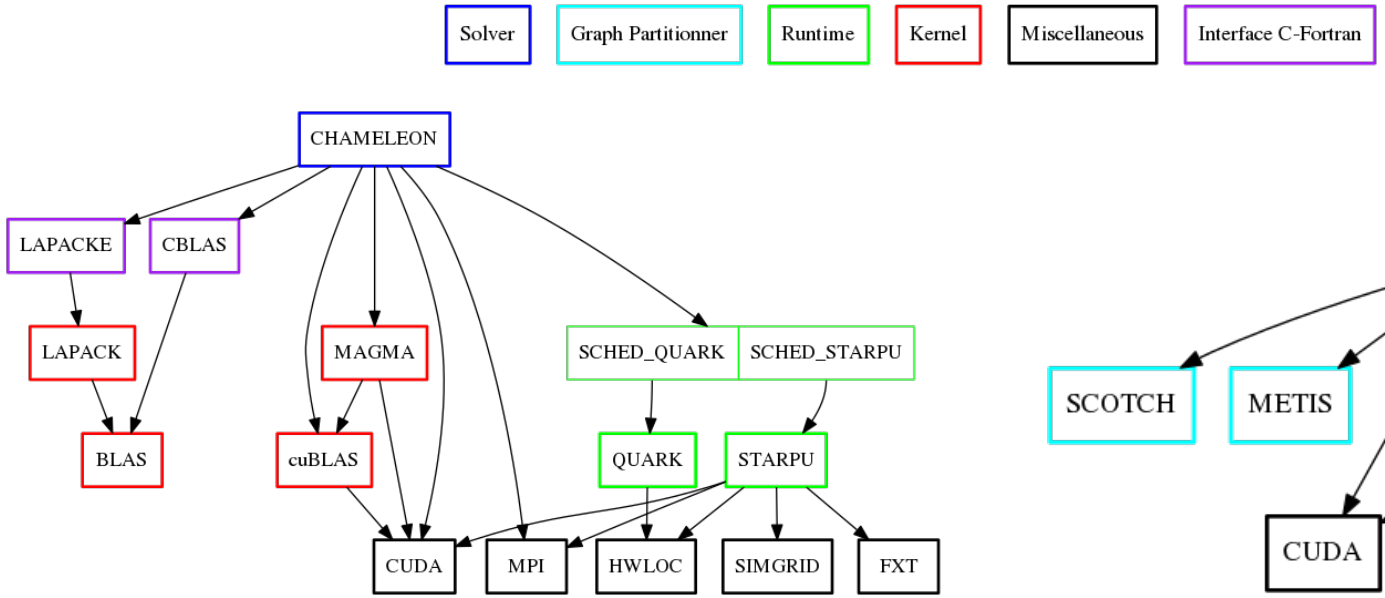


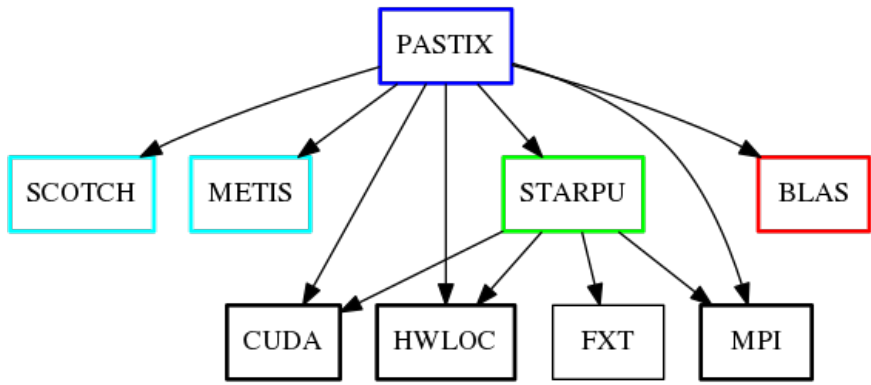


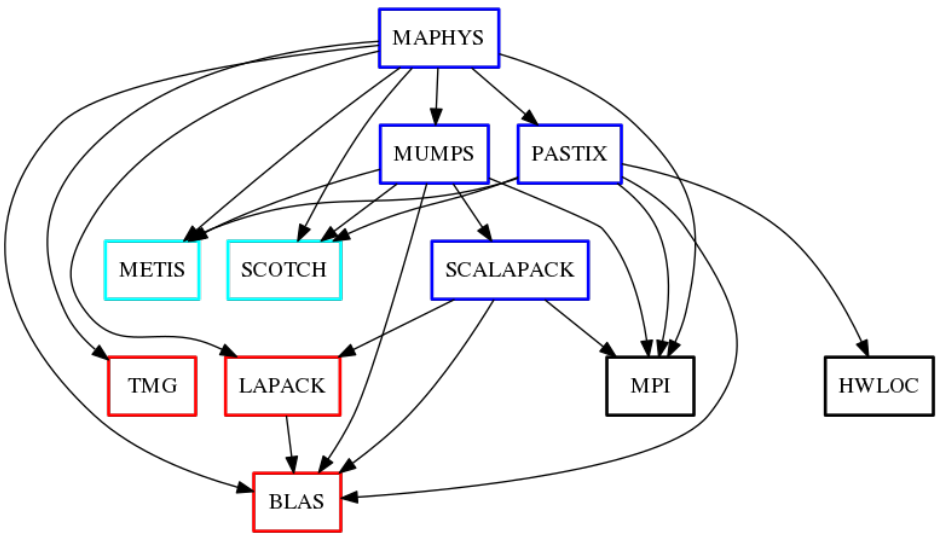
Runtime systems

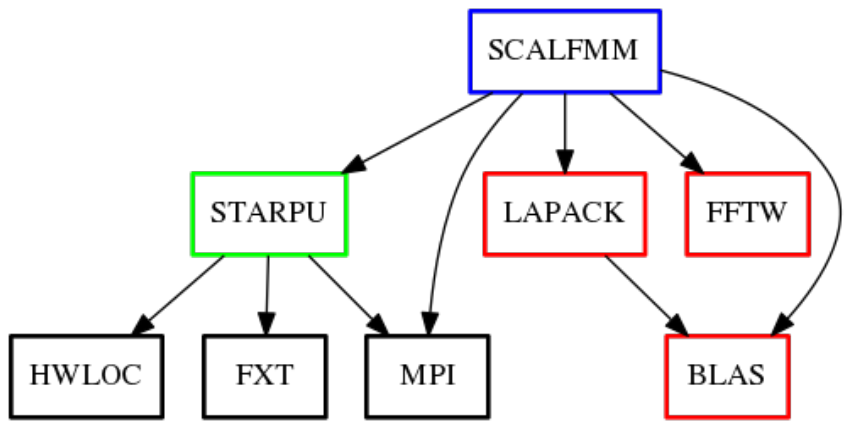
Heterogeneous platforms

How to deploy complex HPC software stacks?









Spack

http://scalability-llnl.github.io/spack/

- Python 2.7: no install needed, ready to be used

```
$ git clone https://github.com/scalability-llnl/spack.git
$ ./spack/bin/spack install gcc
```

- Easy way to set build variants, examples:

```
$ spack install openmpi %gcc@4.9.2
$ spack install netlib-lapack +shared
$ spack install parpack ^netlib-lapack ^openmpi@1.10.0
```

- Handle modulefiles, mirrors to work on clusters

```
$ spack load mpi
$ spack mirror create openmpi mpich hwloc netlib-blas
$ spack mirror add
```

Morse in Spack: a fork where new packages can be found

Engineer F. Pruvost (HiePACS / Sed)

- Available online - git repository:

  https://github.com/fpruvost/spack/ - morse branch

```
$ git clone https://github.com/fpruvost/spack.git
$ cd spack && git checkout morse
$ ./bin/spack install maphys
```

- Build variants examples:

```
$ spack install maphys ~examples +mumps
$ spack install pastix +starpu ^starpu@1.1.2 ^mkl-blas
$ spack install starpu@svn-1.2 +debug +cuda +mpi +fxt +examples
```

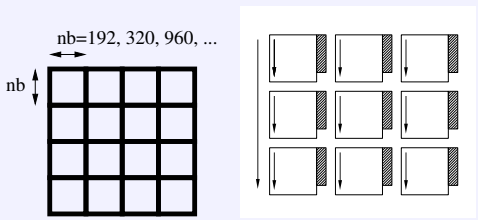
Online tutorial:

http://morse.gforge.inria.fr/tuto_spack-morse/tuto_spack.html

Case study: dense linear algebra

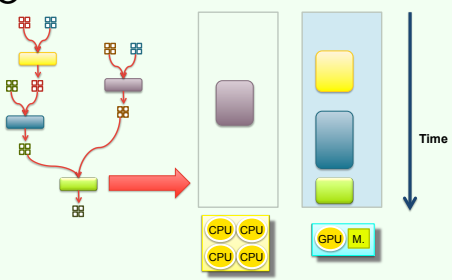
Chameleon = Sequential Task Flow (STF) design of dense linear algebra tiles algorithms on top of runtime systems

Tile matrix layout



Runtime systems

- QUARK
- StarPU



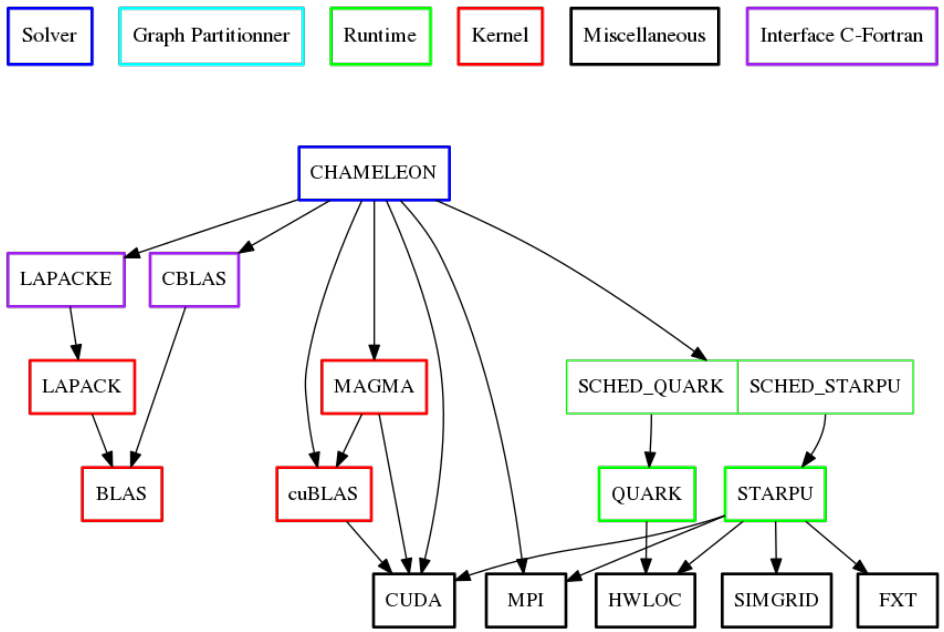
STF algorithms

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

Optimized kernels

- BLAS, LAPACK
- cuBLAS, MAGMA

How to deploy the Chameleon software stack?

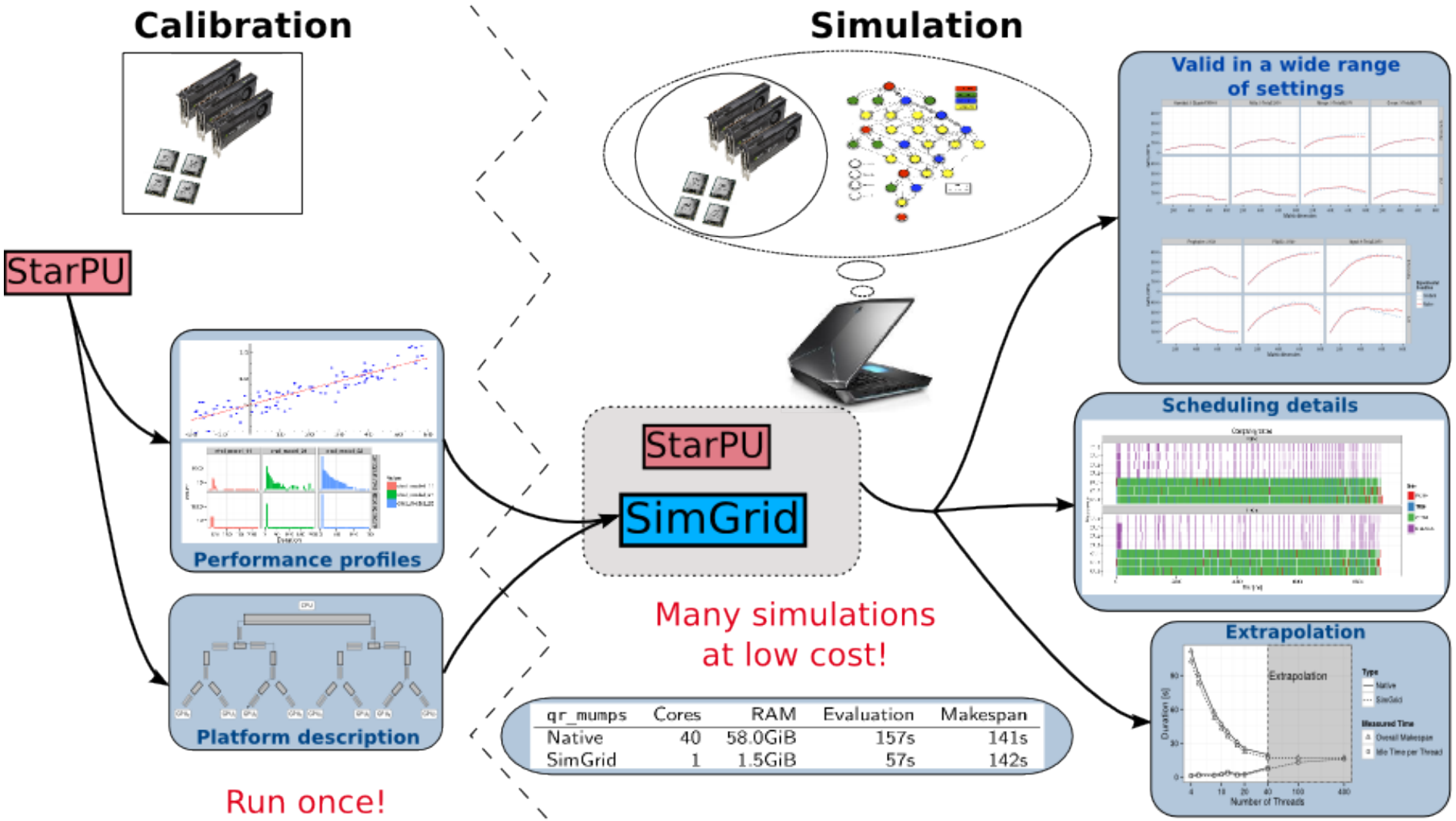


Online tutorial:

http://morse.gforge.inria.fr/tuto_chameleon/chameleon-tutorial-2015-12-16-inria.html

Simulating Chameleon using StarPU-SimGrid

PhD L. Stanisic



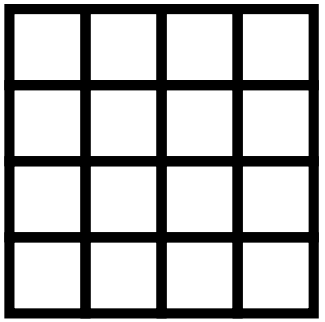
qr_mumps	Cores	RAM	Evaluation	Makespan
Native	40	58.0GiB	157s	141s
SimGrid	1	1.5GiB	57s	142s

Outline

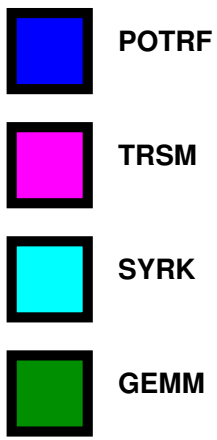
Outline

STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels



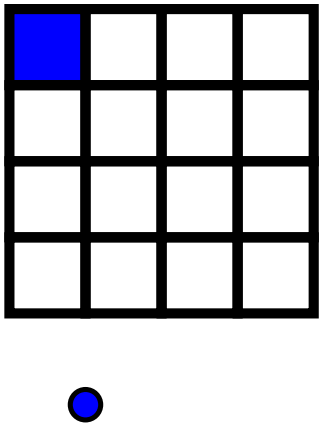
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

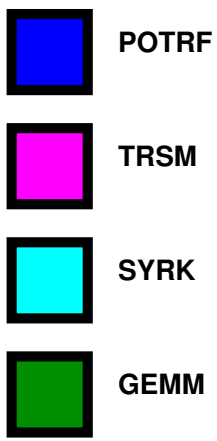


STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```

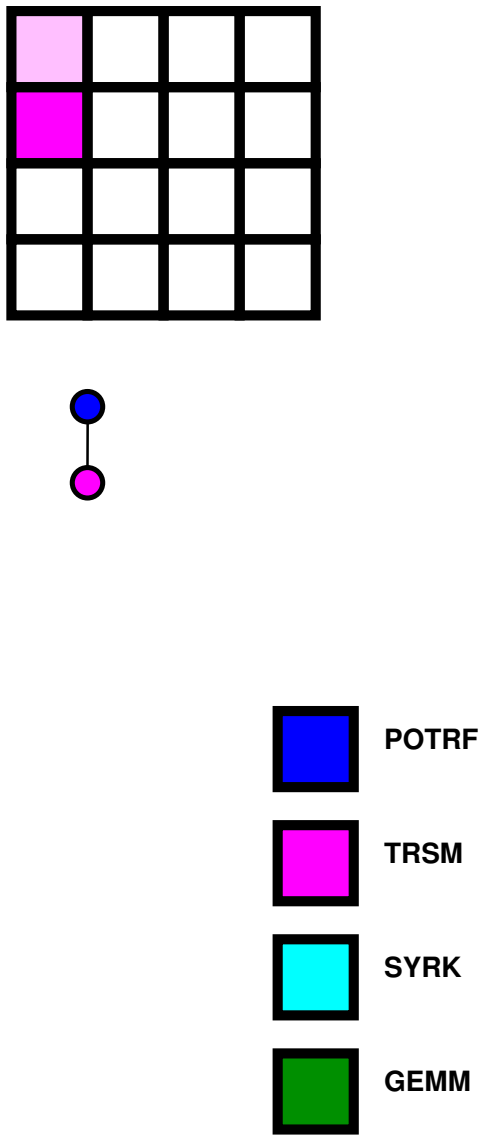




STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

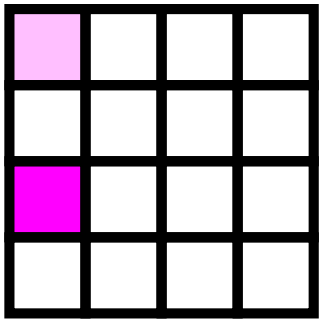
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

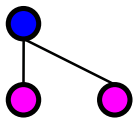


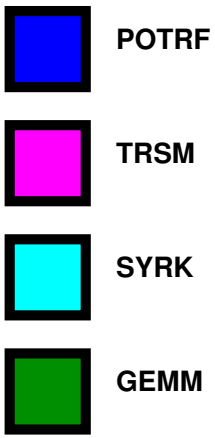
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



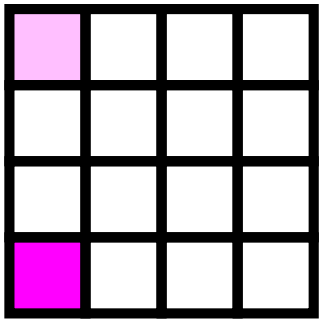


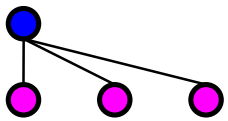


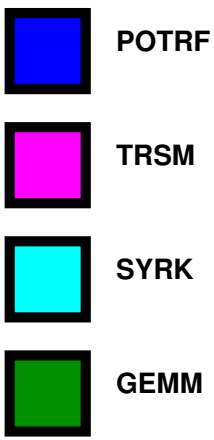
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



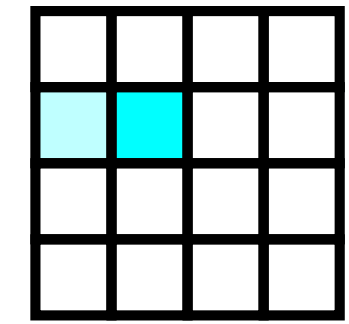


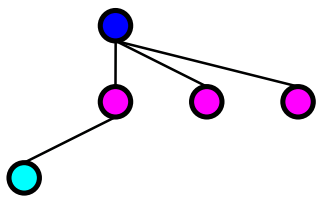


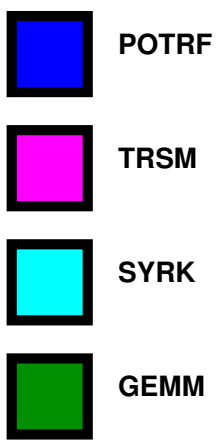
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



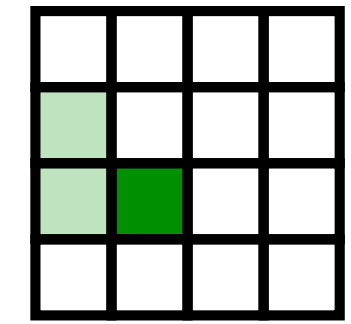


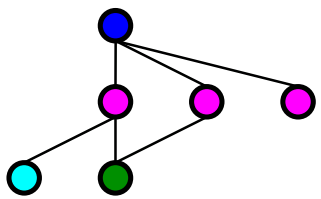


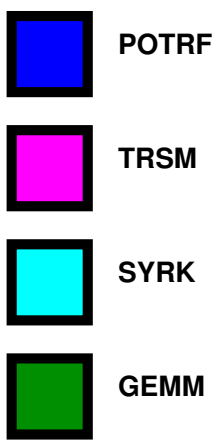
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



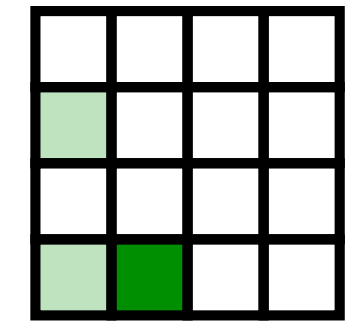


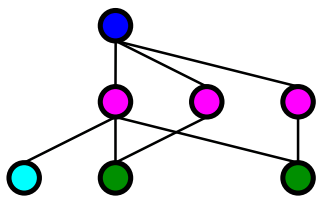


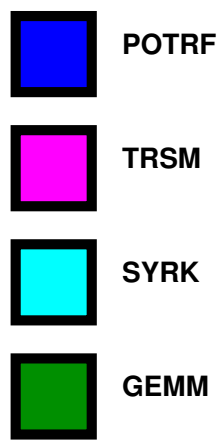
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



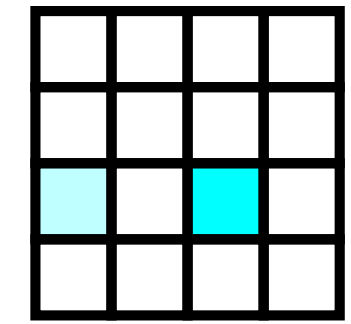


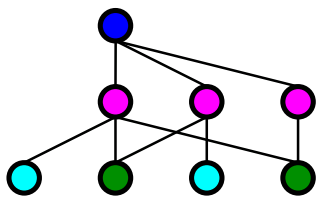


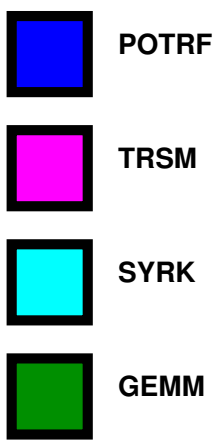
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



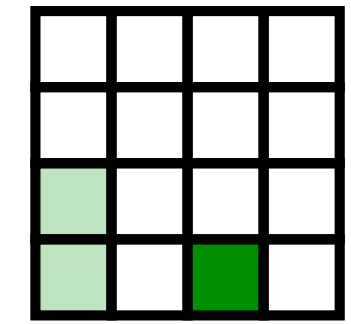


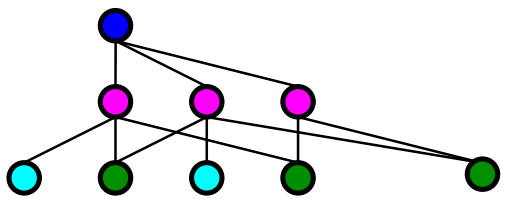


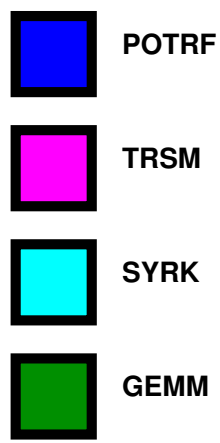
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



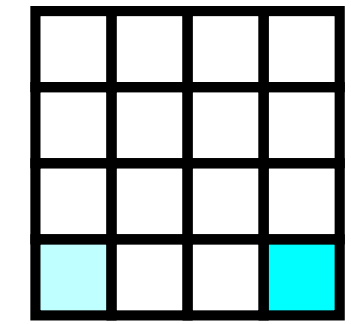


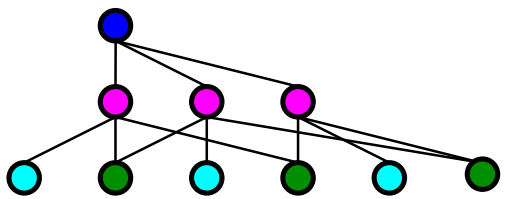


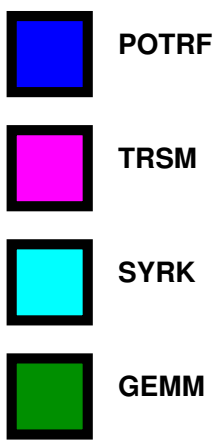
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```
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__();
```



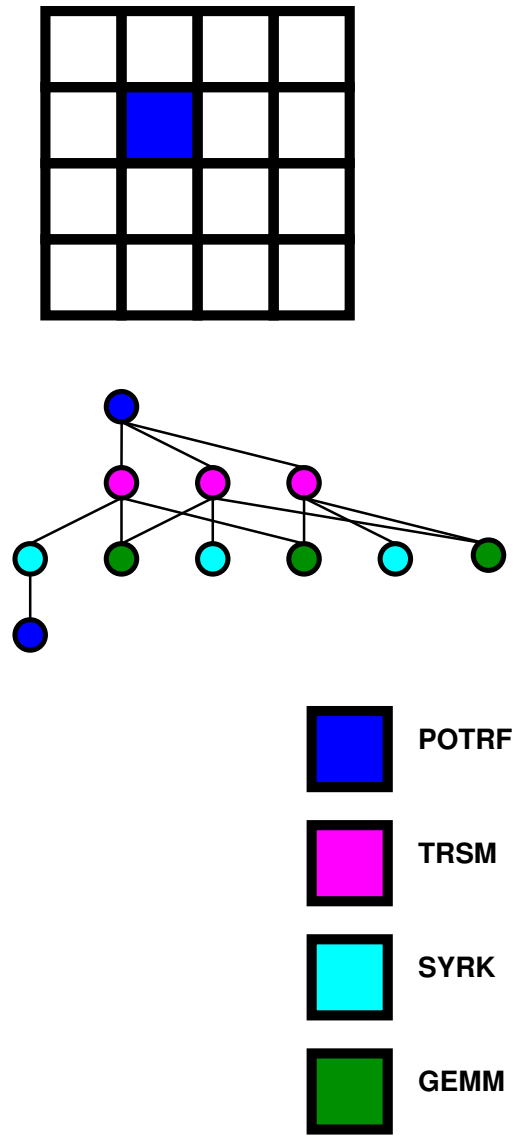




STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

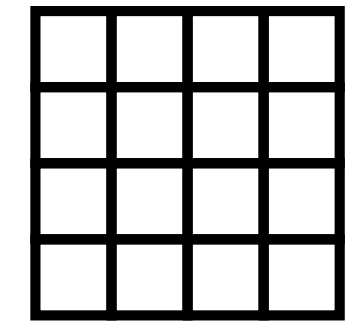
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

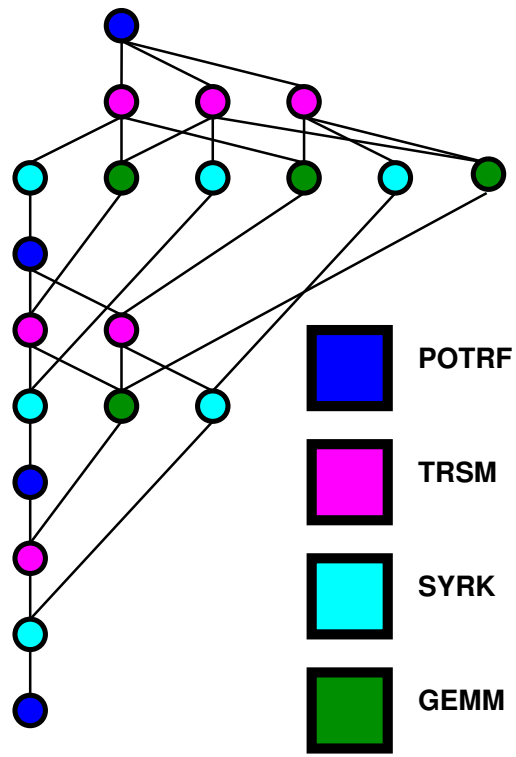


STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

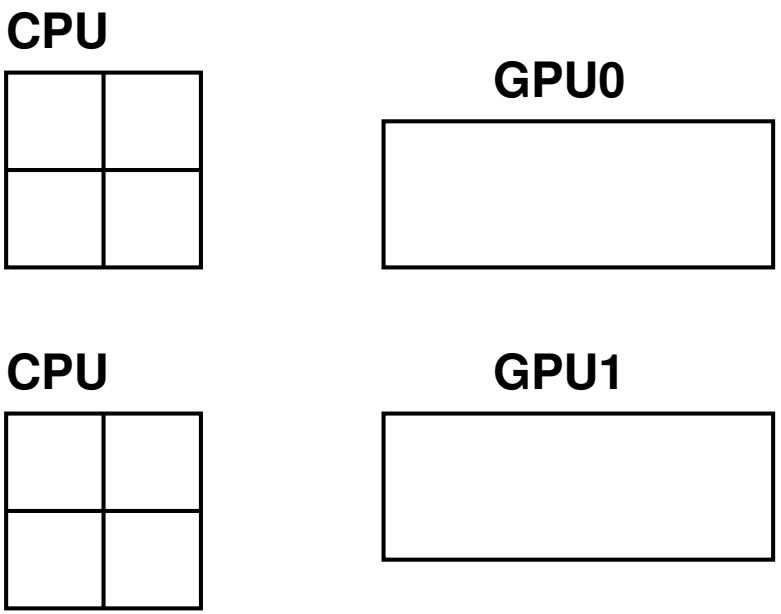
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

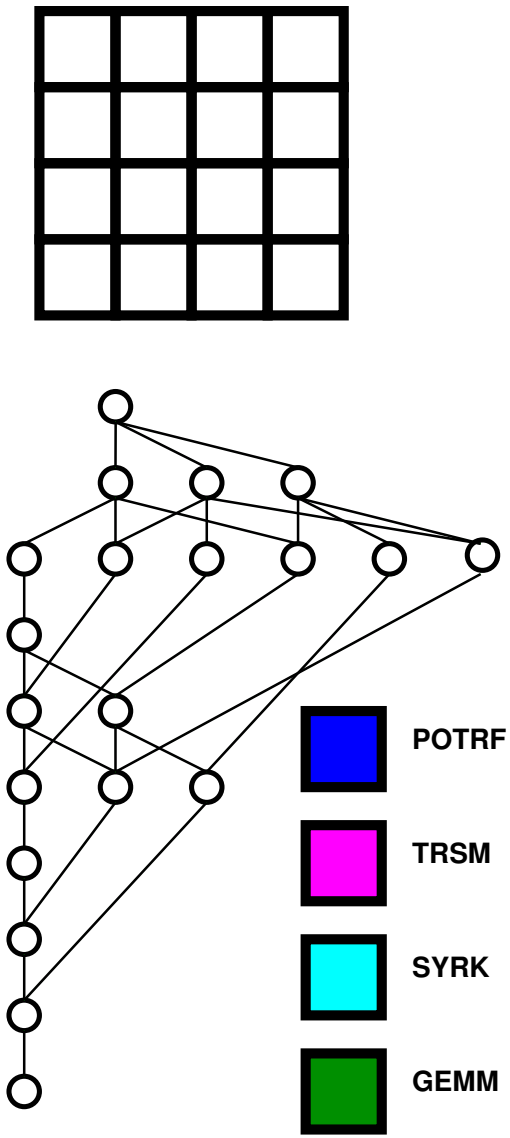




On an heterogeneous node

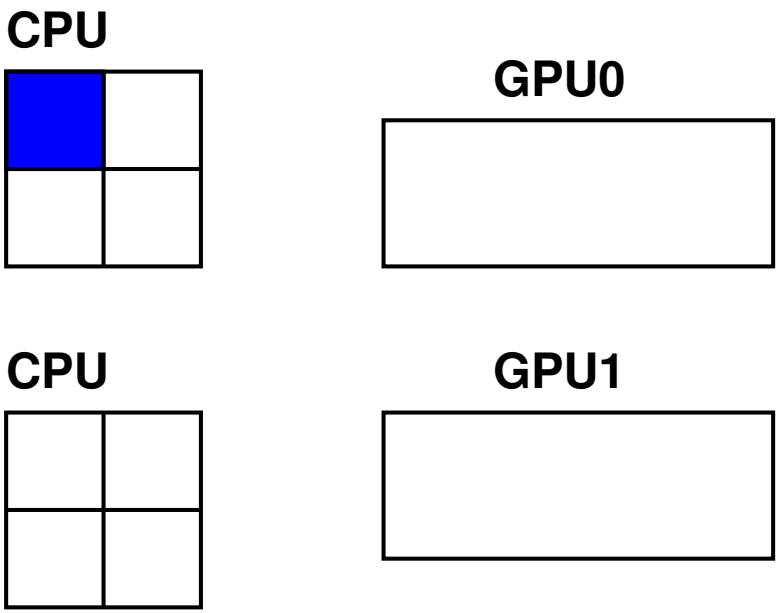
work on tiles $\rightarrow$ CPU + GPU kernels

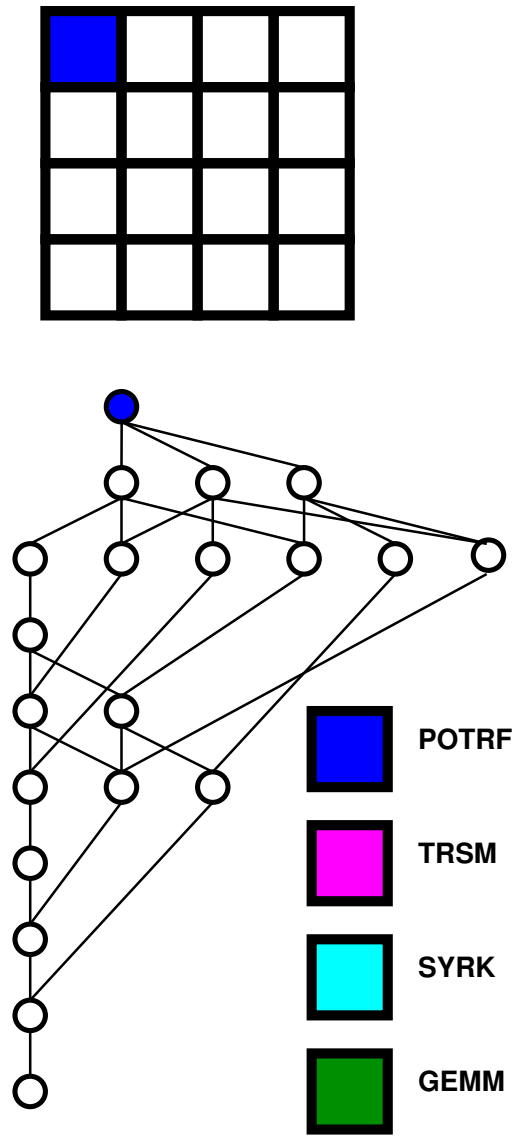




On an heterogeneous node

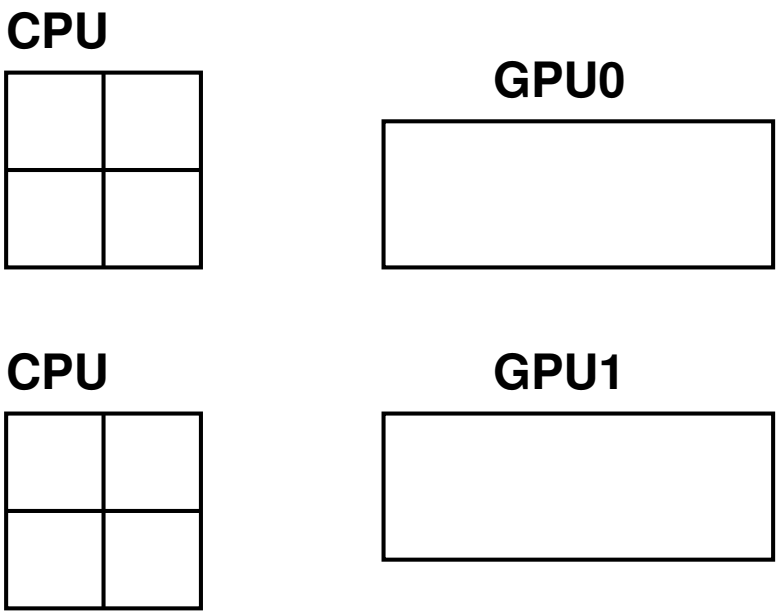
work on tiles → CPU + GPU kernels

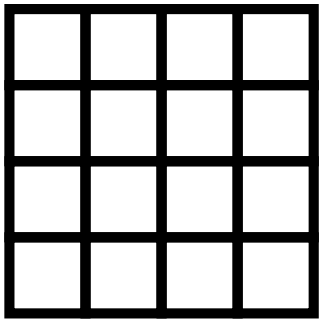


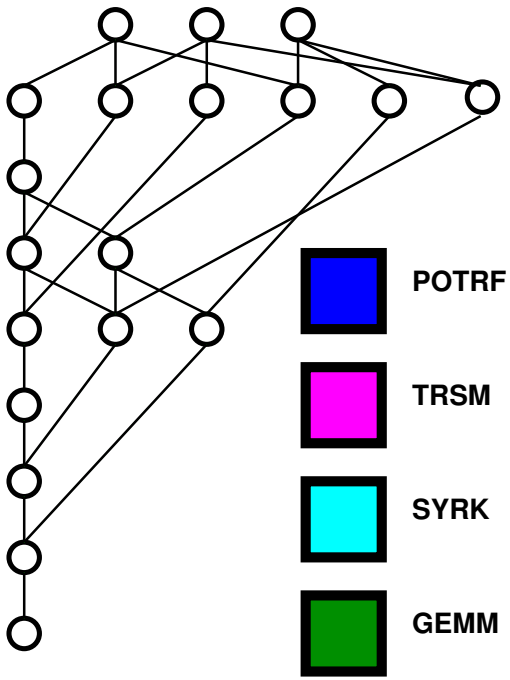


On an heterogeneous node

work on tiles → CPU + GPU kernels

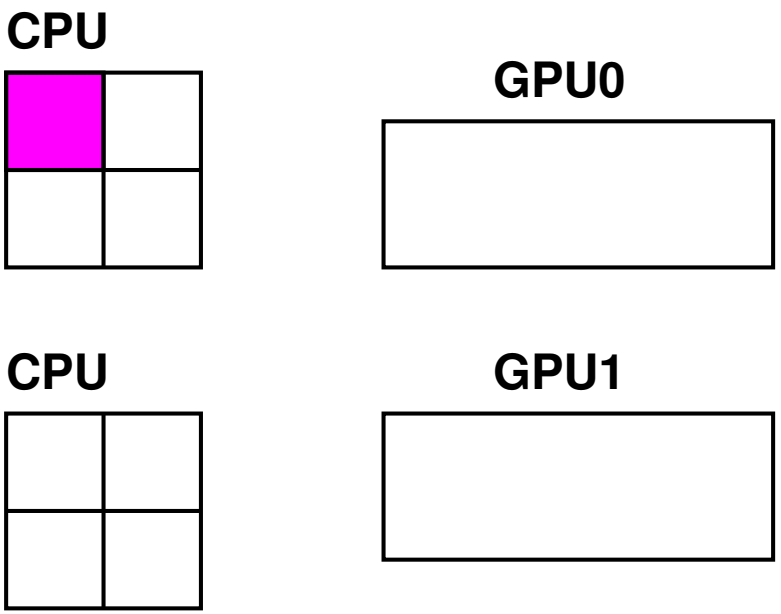


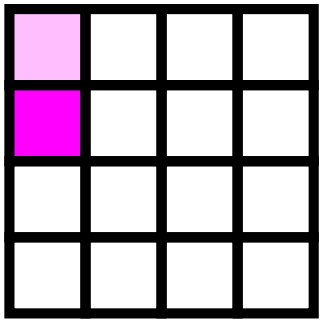


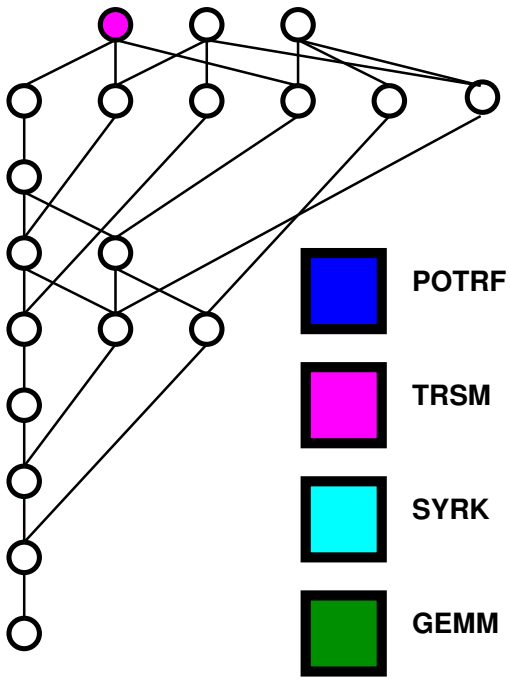


On an heterogeneous node

work on tiles → CPU + GPU kernels

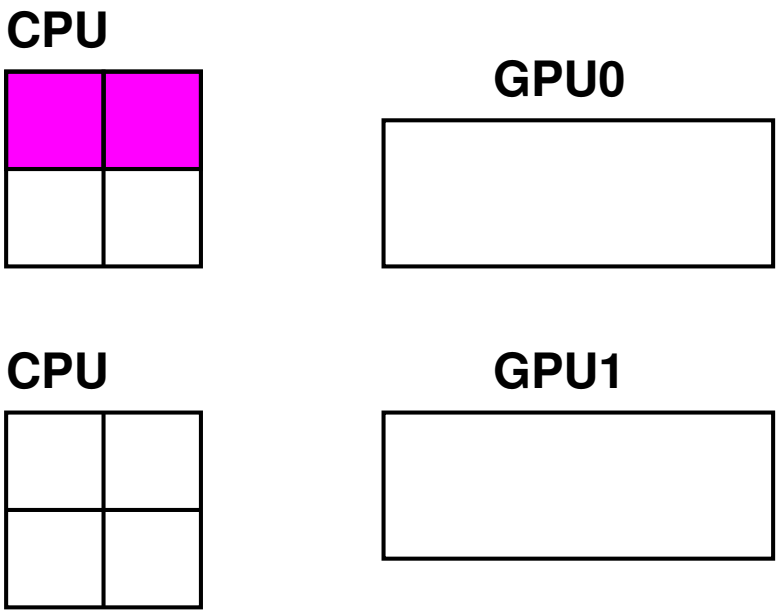


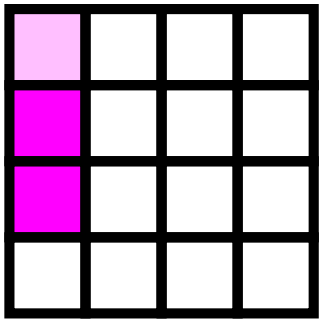


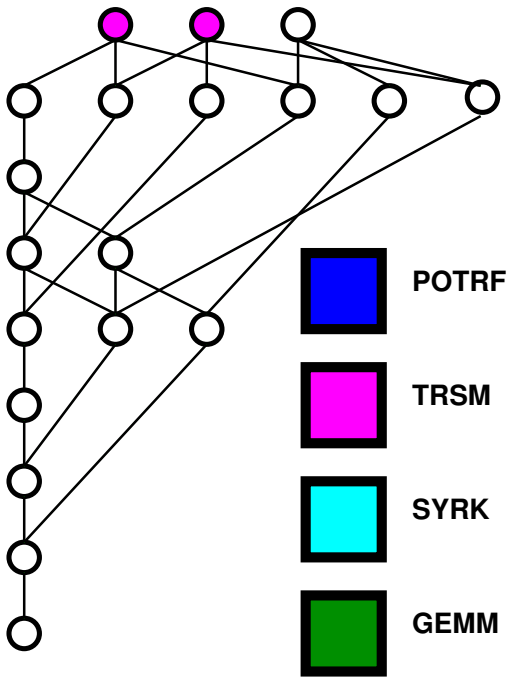


On an heterogeneous node

work on tiles → CPU + GPU kernels

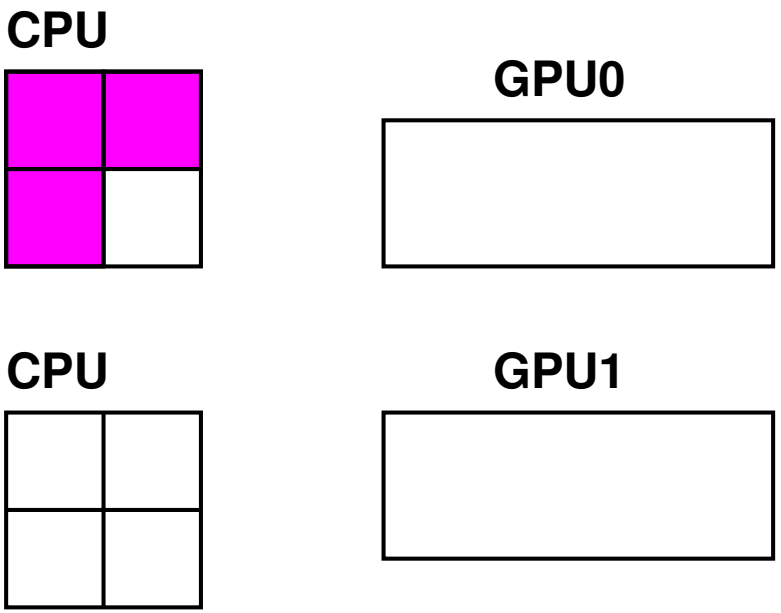


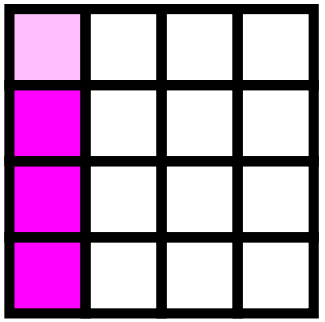


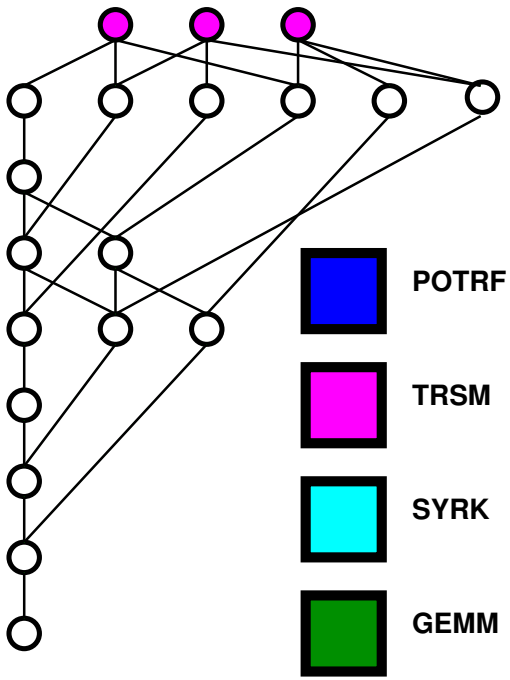


On an heterogeneous node

work on tiles → CPU + GPU kernels

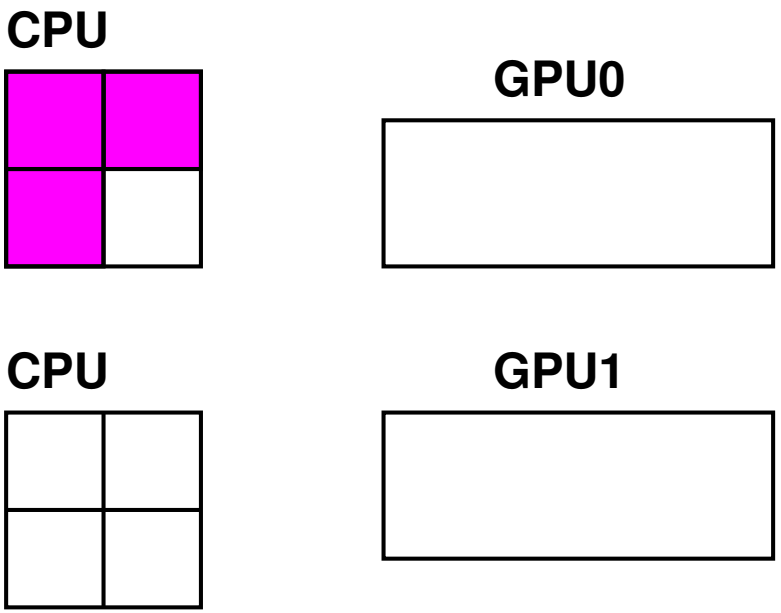


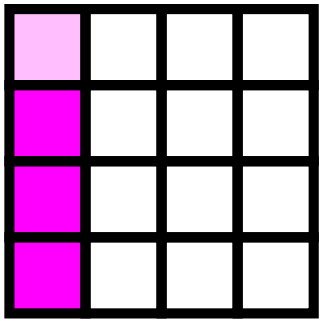


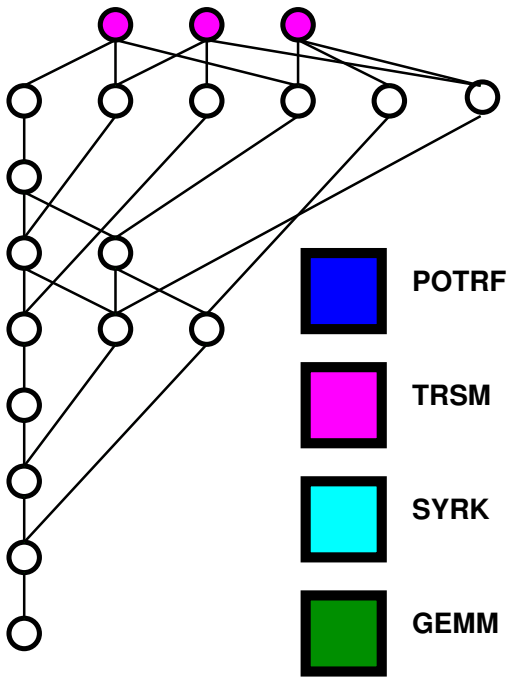


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels



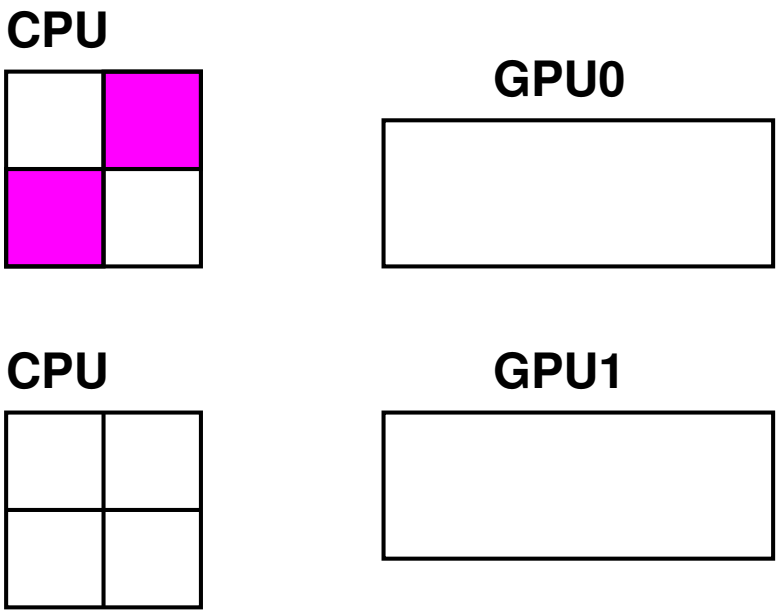


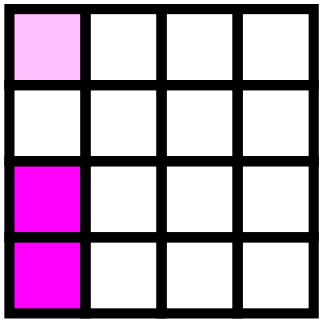


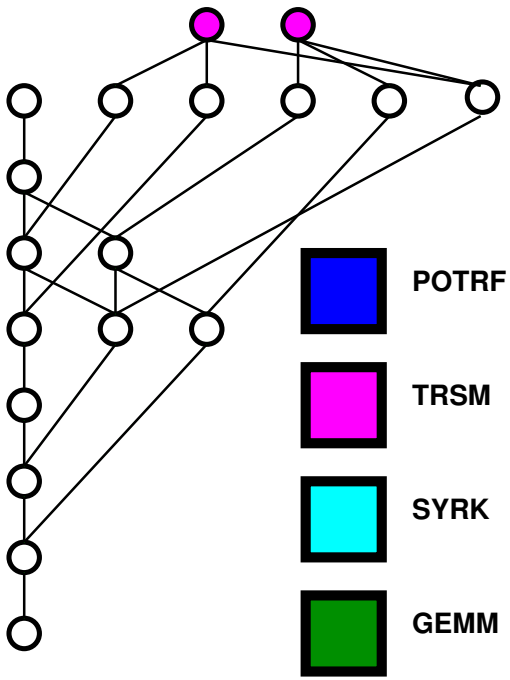
- Handles dependencies

On an heterogeneous node

work on tiles → CPU + GPU kernels



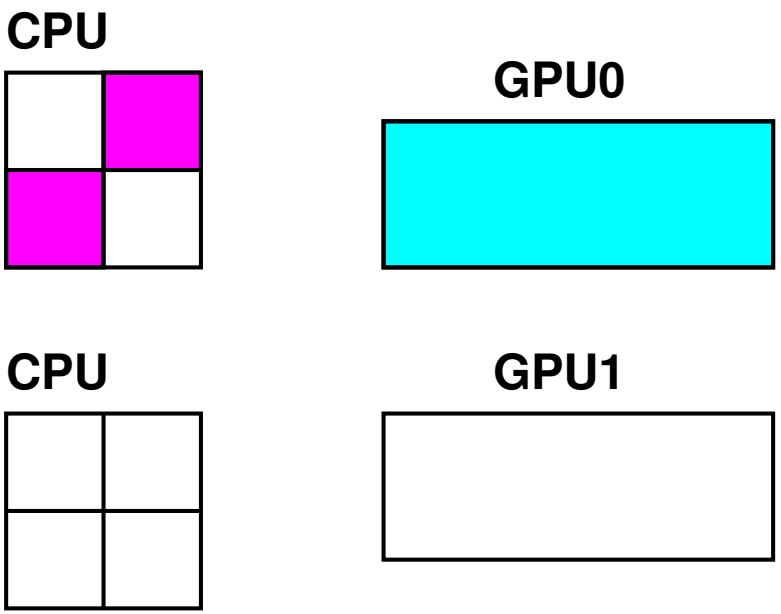


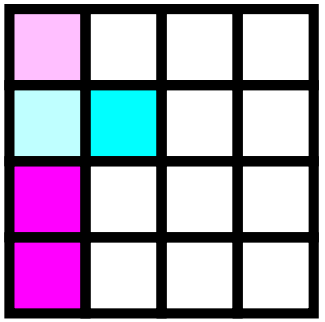


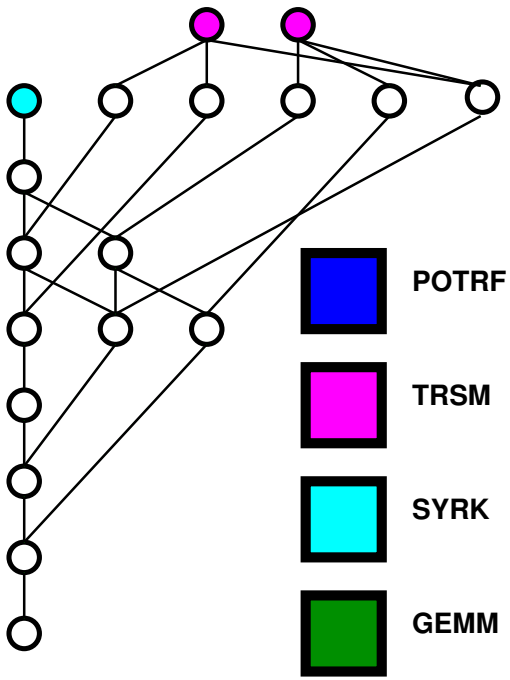
- Handles dependencies

On an heterogeneous node

work on tiles → CPU + GPU kernels



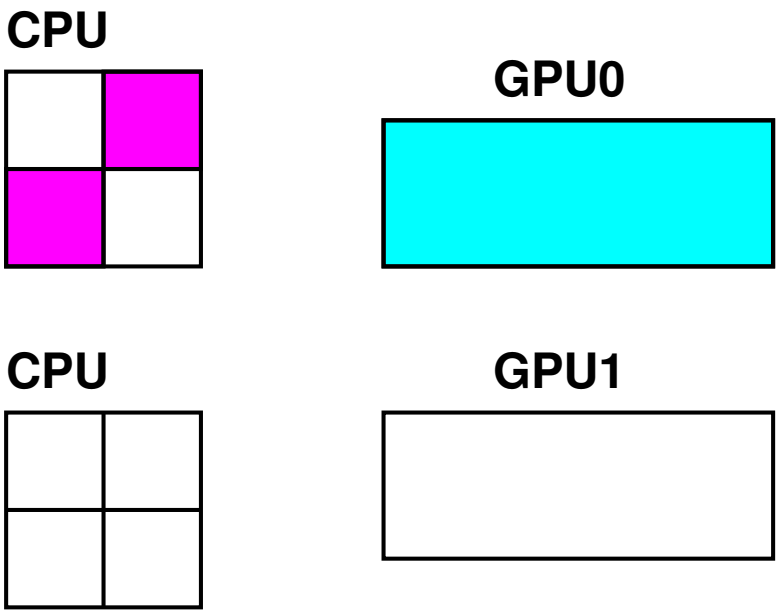


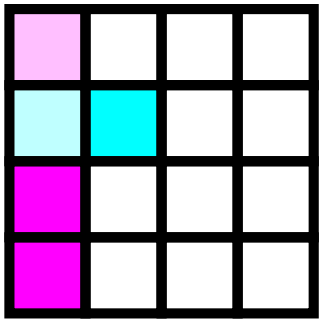


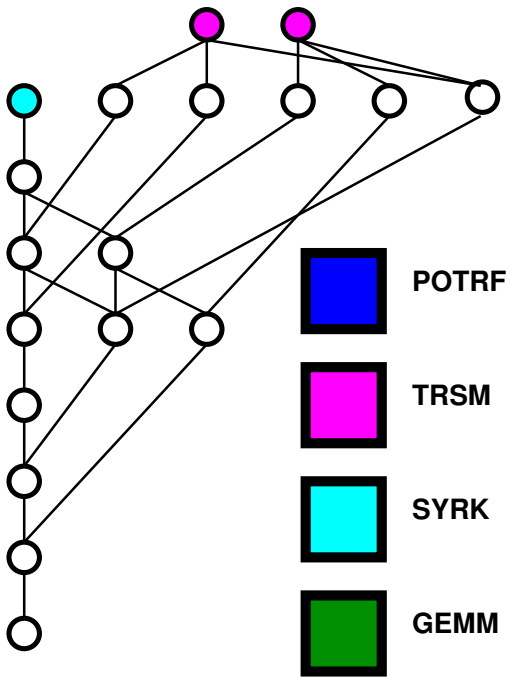
- Handles dependencies

On an heterogeneous node

work on tiles → CPU + GPU kernels



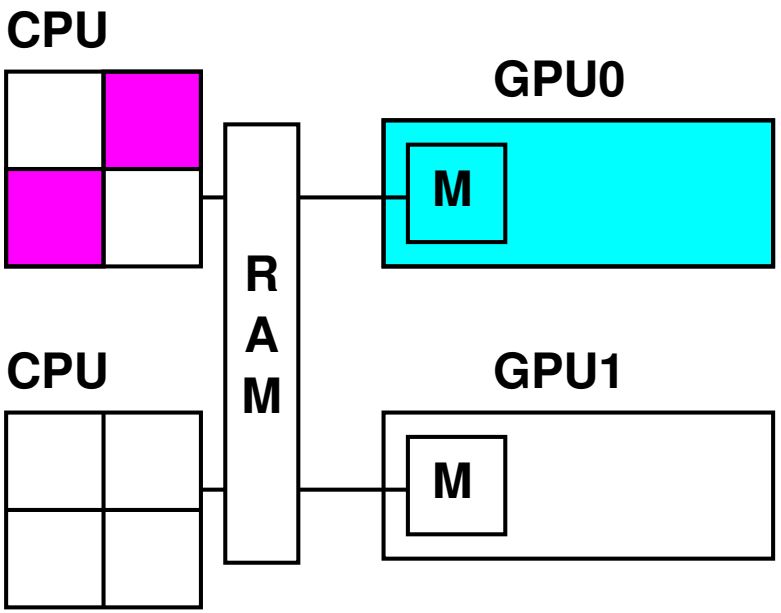


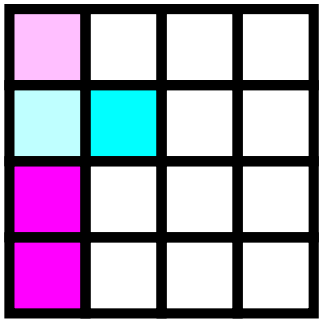


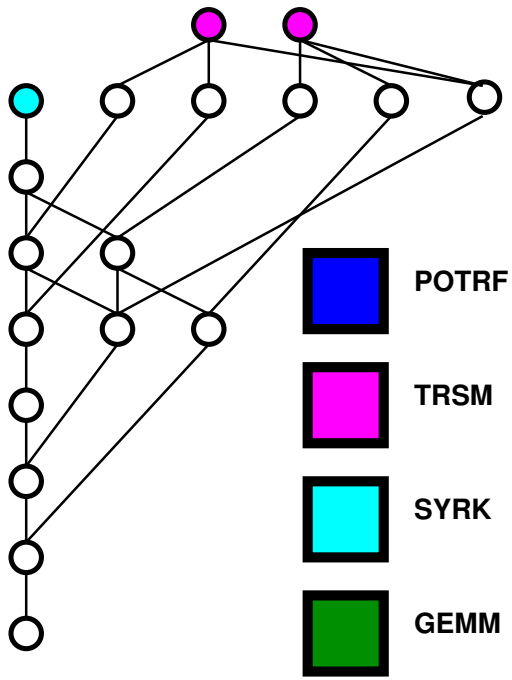
- Handles dependencies
- Handles scheduling

On an heterogeneous node

work on tiles → CPU + GPU kernels



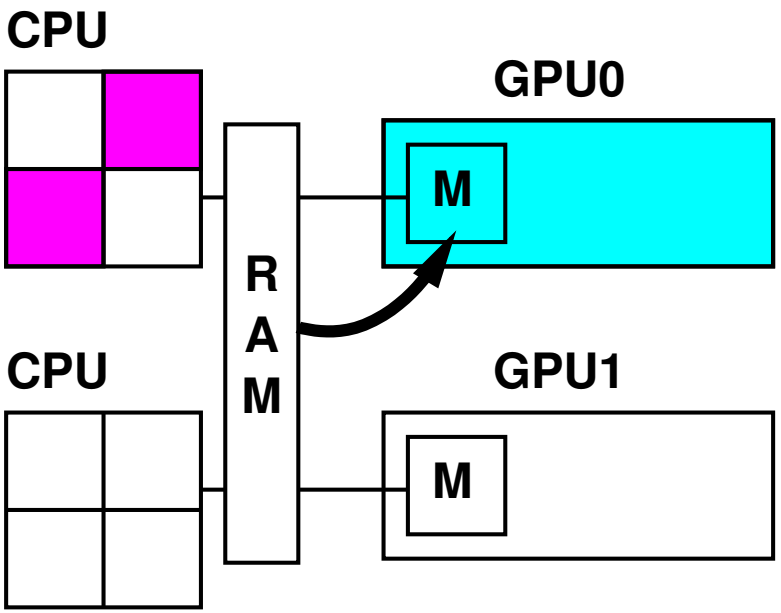


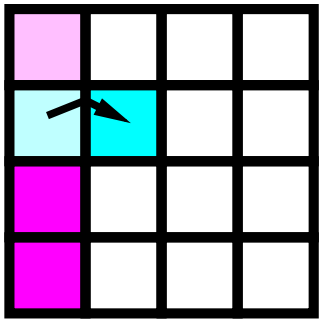


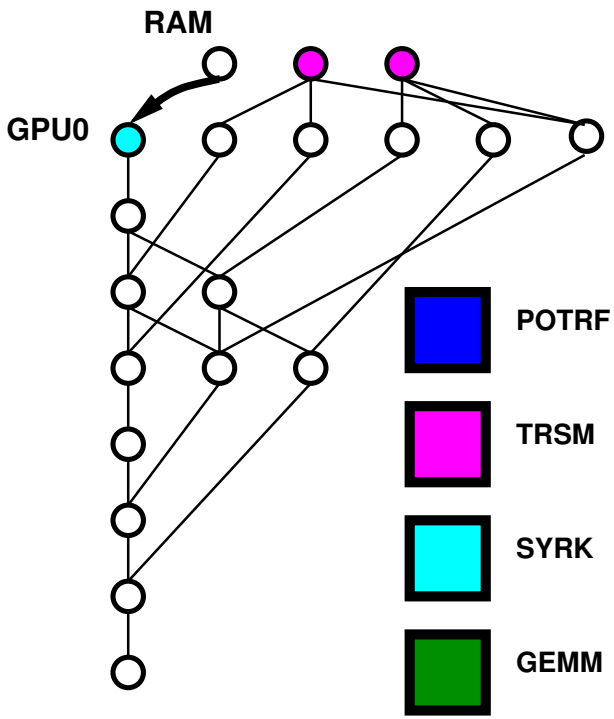
- Handles dependencies
- Handles scheduling

On an heterogeneous node

work on tiles → CPU + GPU kernels







- Handles dependencies
- Handles scheduling
- Handles data consistency

Outline

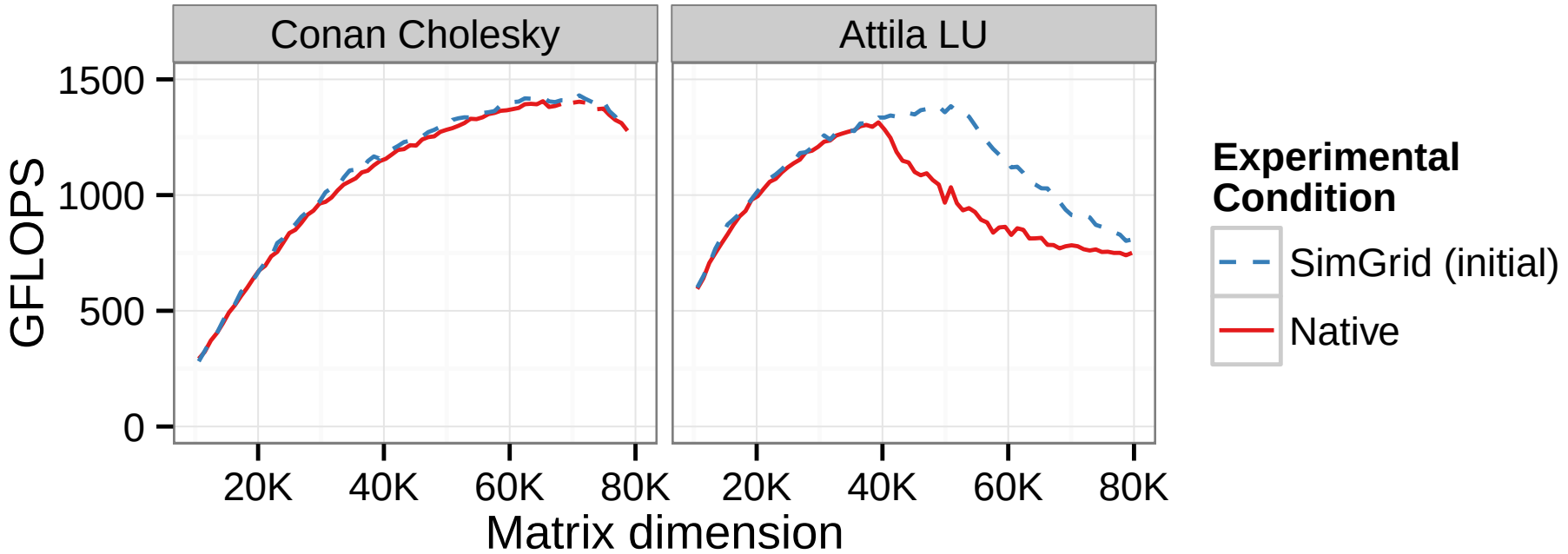
Dense Linear Algebra Applications

PhD L. Stanisic

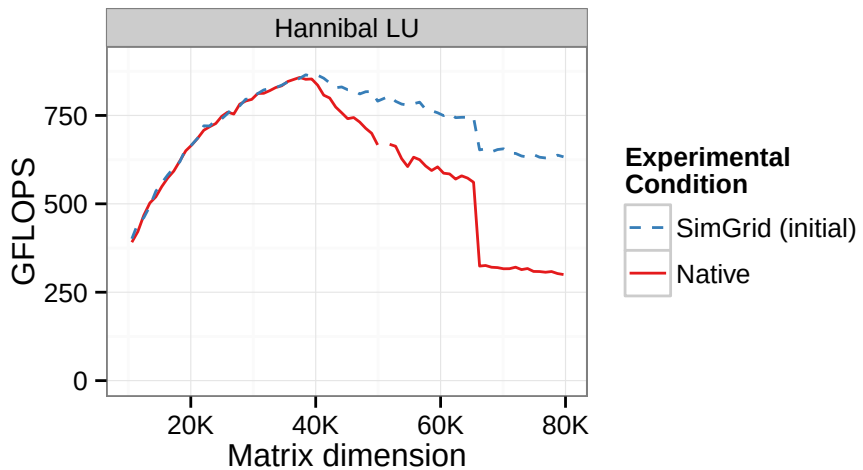
- Started with regular dense kernels and a fixed tile size
- Used two different matrix decomposition algorithms:
  - Cholesky
  - LU
- Used a wide diversity of machines

Name	Processor	#Cores	Memory	GPUs
hannibal	X5550	2×4	2×24GiB	3×QuadroFX5800
attila	X5650	2×6	2×24GiB	3×TeslaC2050
mirage	X5650	2×6	2×18GiB	3×TeslaM2070
conan	E5-2650	2×8	2×32GiB	3×TeslaM2075
frogkepler	E5-2670	2×8	2×16GiB	2×K20
pilipili2	E5-2630	2×6	2×32GiB	2×K40
idgraf	X5650	2×6	2×36GiB	8×TeslaC2050
idchire	E5-4640	24×8	24×31GiB	/

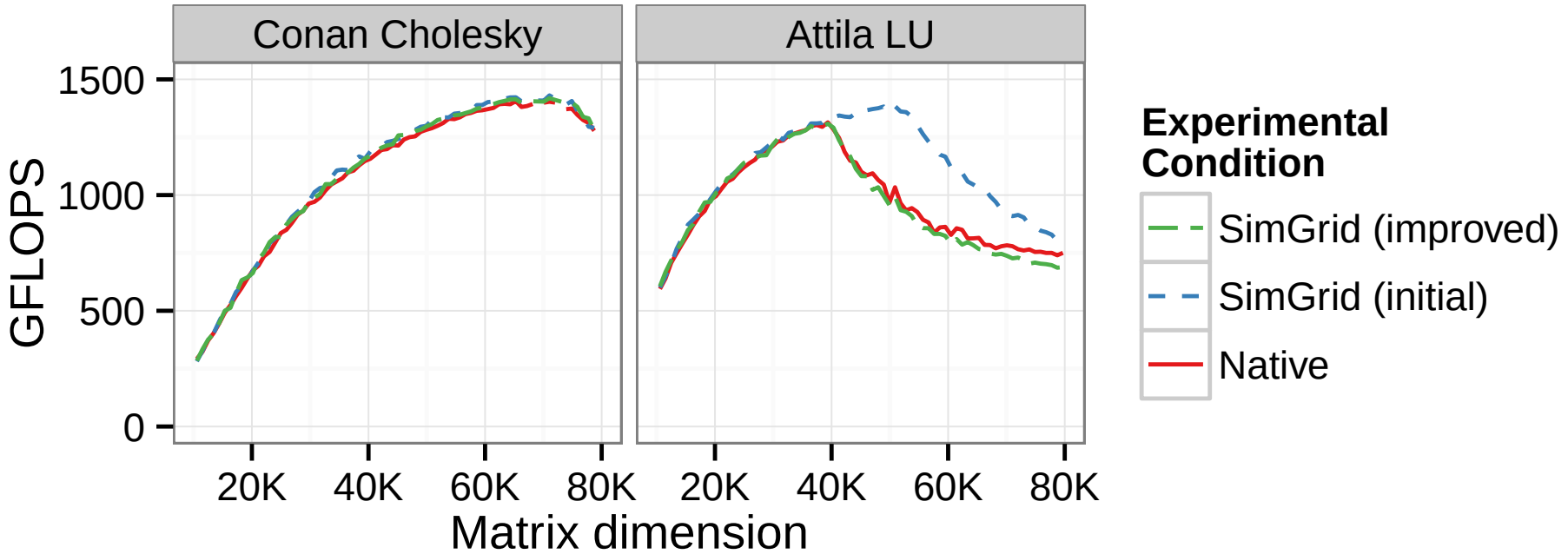
Initial Results



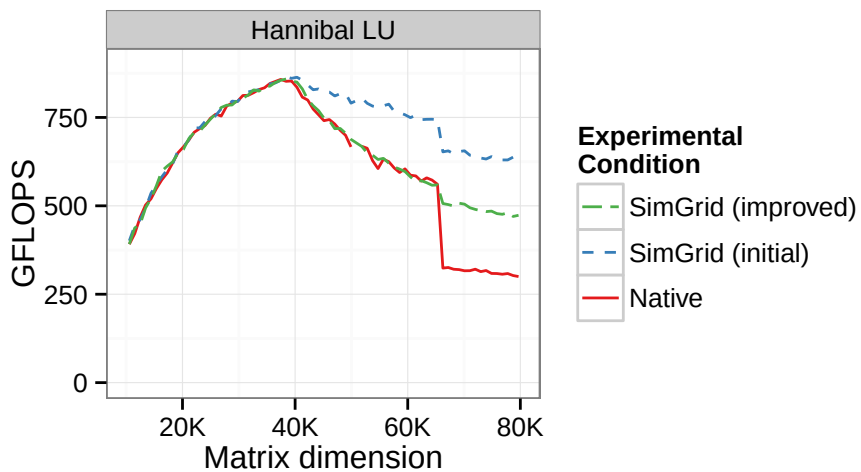
- Most results are satisfactory
- Some need improved models
- For attila and hannibal SimGrid is too optimistic $\rightsquigarrow$ why?



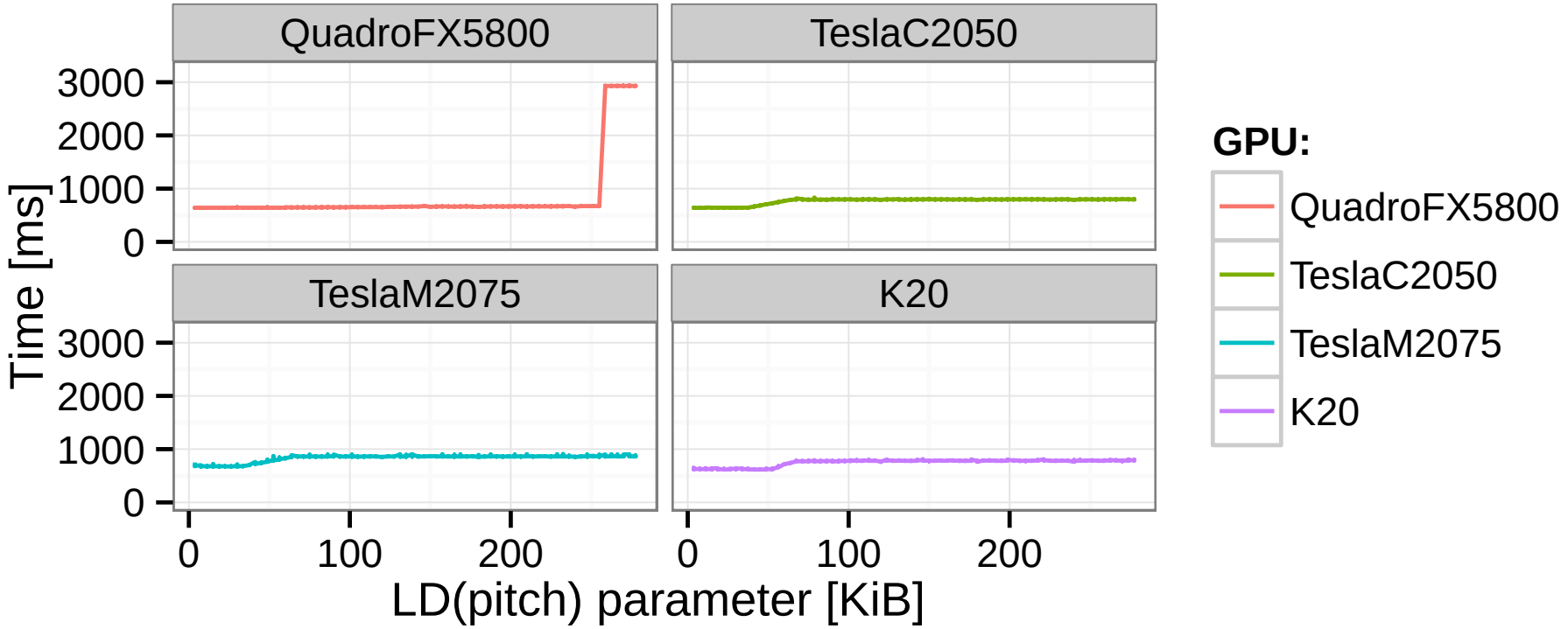
Improving GPU Memory Models



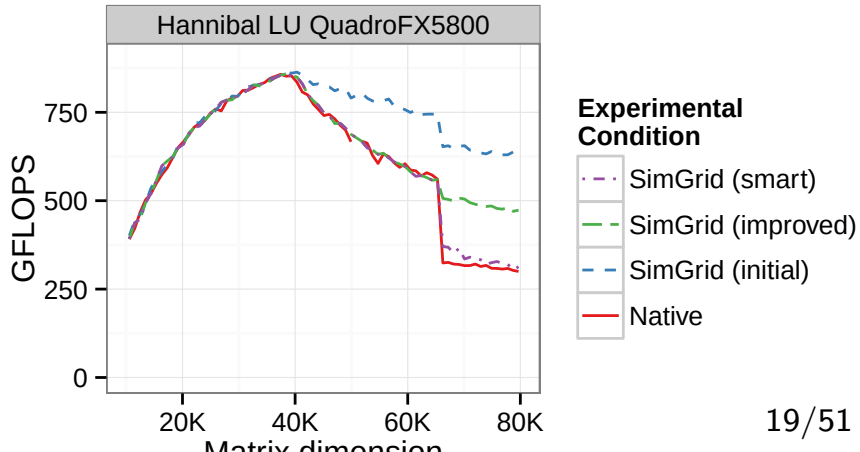
- Taking GPU size into account (attila and hannibal have 3GiB)
- Using improved network model
- Performance drop for large matrices on hannibal $\rightsquigarrow$ why?



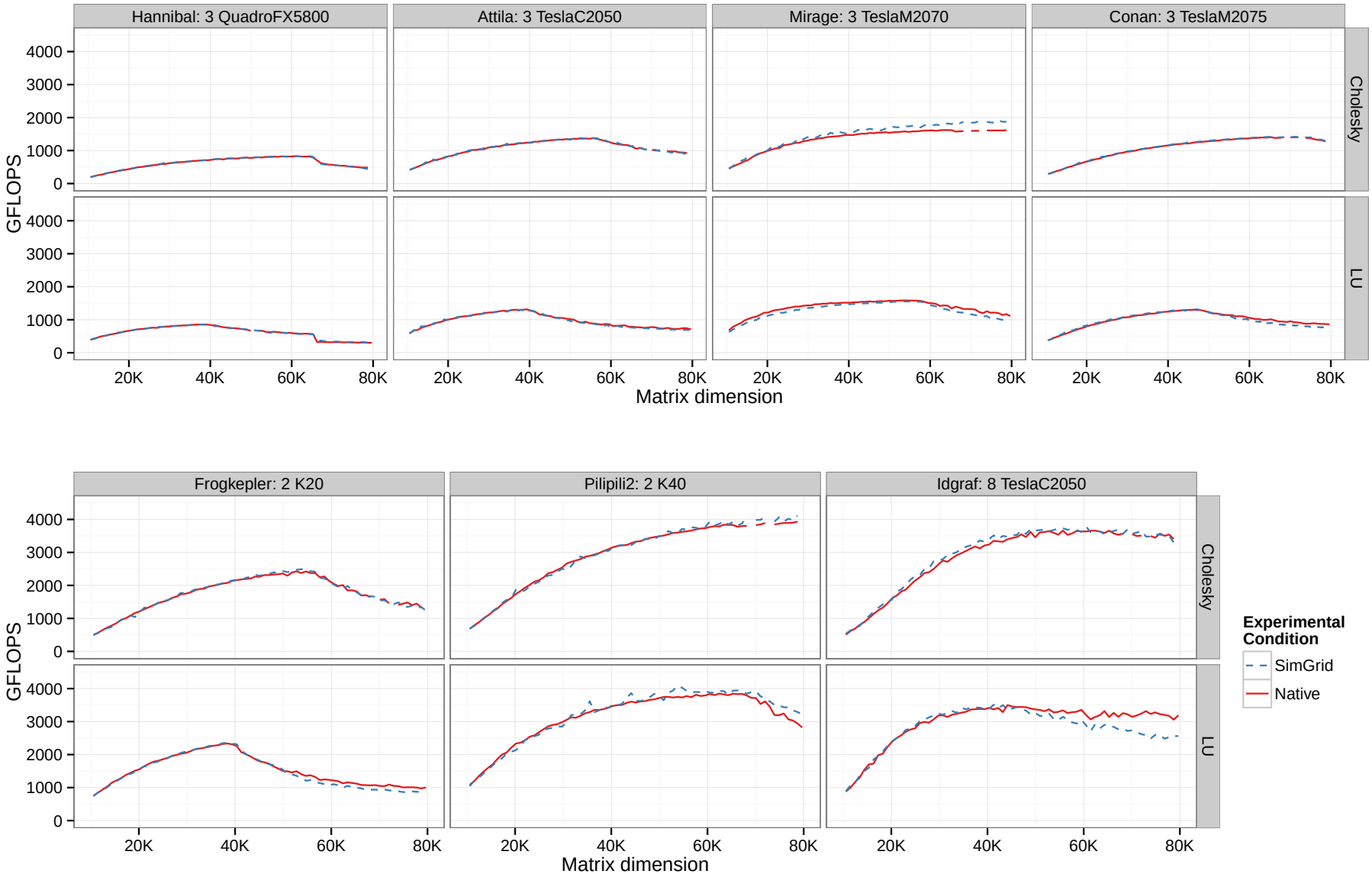
Improving QuadroFX5800 Transfer Model



- Transfer time of 3.6 MB using `cudaMemcpy2D` differs
- Problem with older GPUs when copying with large strides

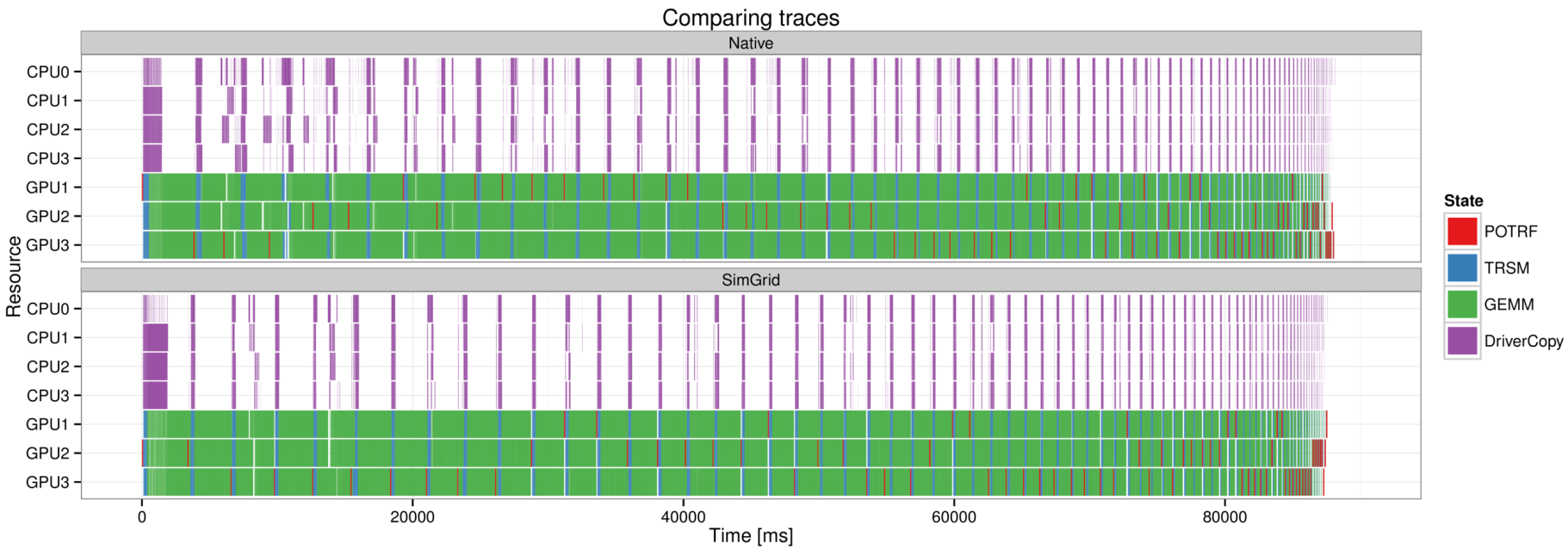


Overview of Simulation Accuracy



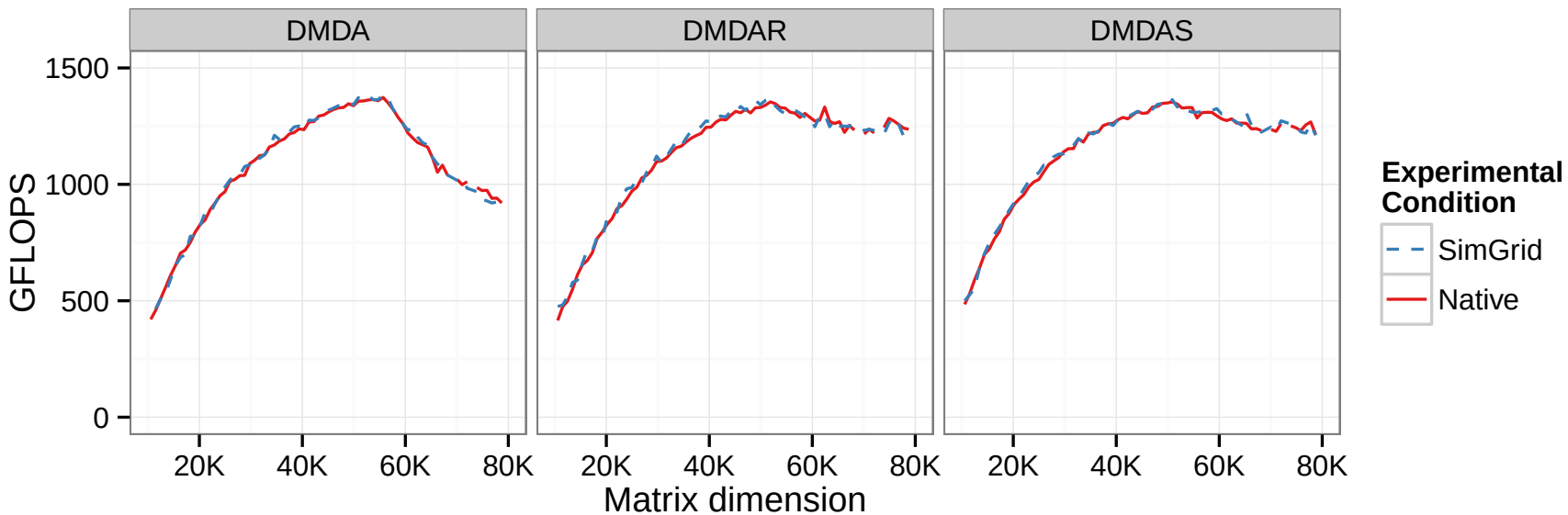
Visual Trace Comparing

- 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

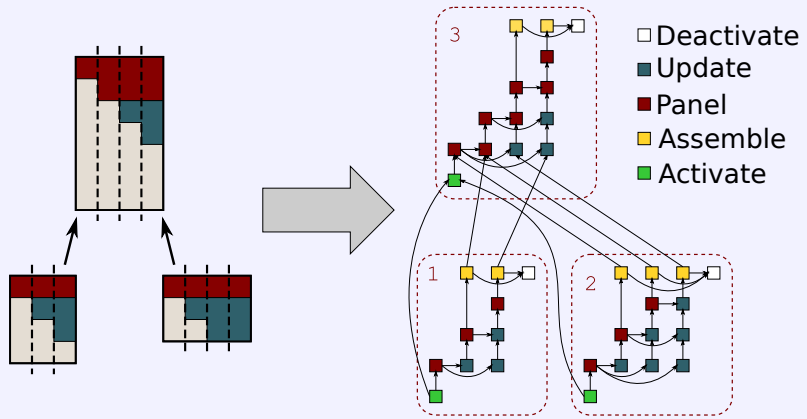


Outline

Simulating Sparse Solvers

qr_mumps

- Task-based multifrontal QR solver - A. Buttari (CNRS / IRIT)
  - Tree parallelism: nodes in separate branches can be treated independently
  - Node parallelism: large nodes can be treated by multiple process
    - ... on top of StarPU - PhD F. Lopez (N7 - IRIT)

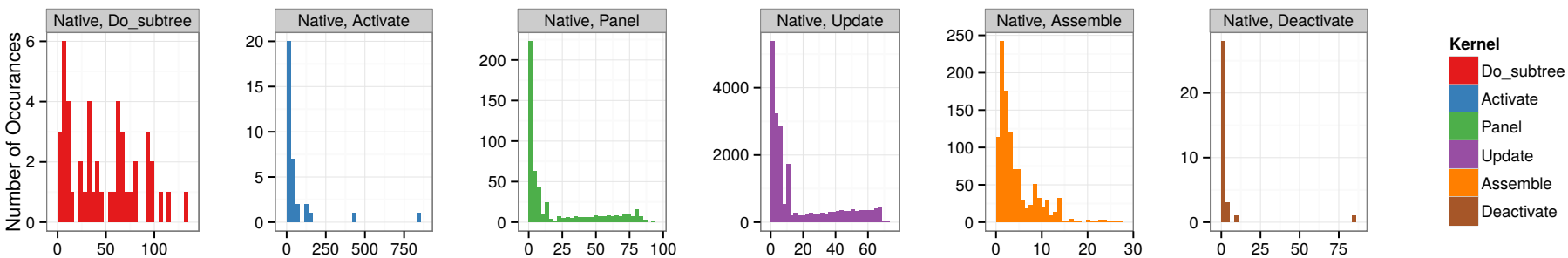


Porting qr_mumps on top of SimGrid

- Changing `main` for the subroutine
- Changing compilation process
- Careful kernel modeling as matrix dimension keeps changing

Difference between Dense and Sparse Kernels

- Dense kernels (`POTRF`, `GEMM`, ...) during single experiment are always executed with the same block size $\rightsquigarrow$ duration is very 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:

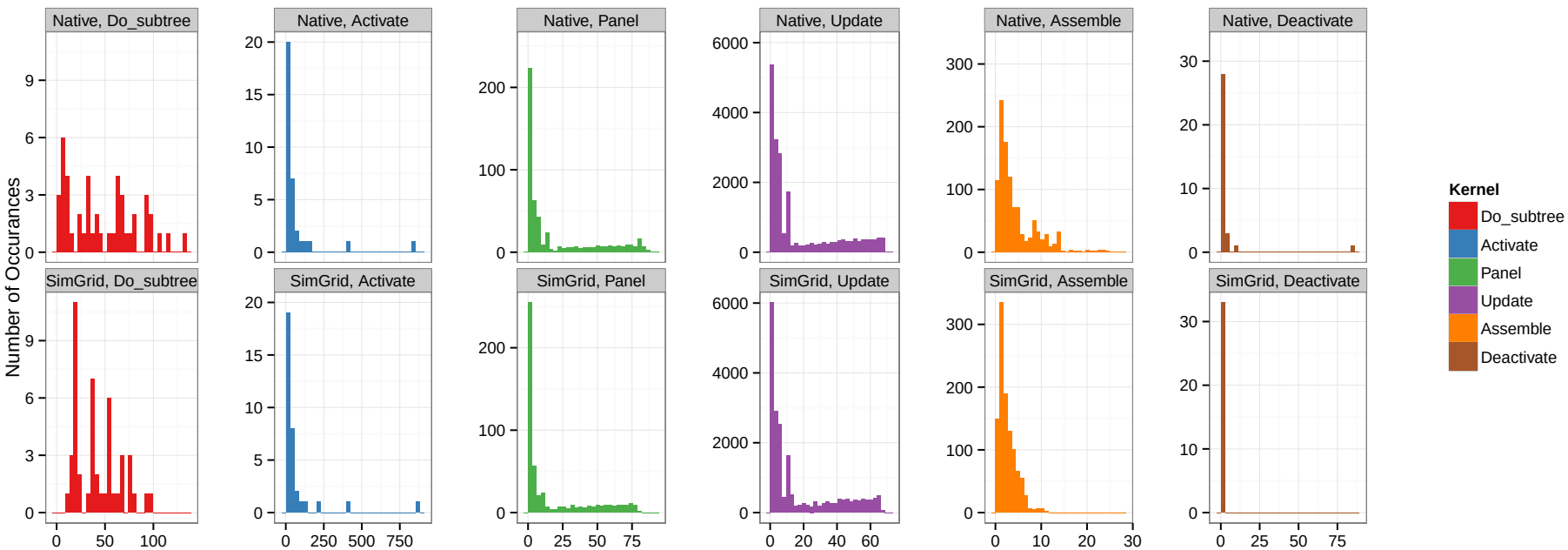
$$T_{\texttt{Panel}} = a + 2b(NB^2 \times MB) - 2c(NB^3 \times BK) + \frac{4d}{3}NB^3$$

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

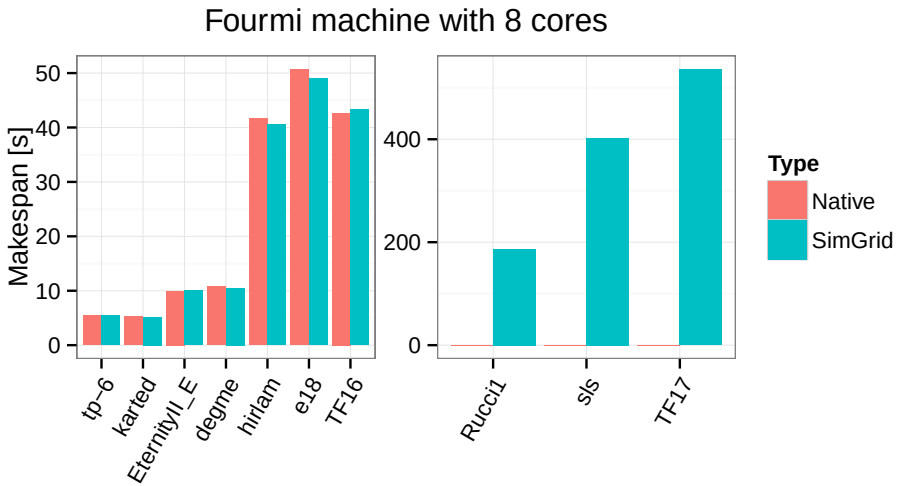
	Panel Duration	
Constant	-2.49×10^{1} $(-2.83 \times 10^{1}, -2.14 \times 10^{1})$	***
$NB^2 \times MB$	5.49×10^{-7} $(5.46 \times 10^{-7}, 5.51 \times 10^{-7})$	***
$NB^3 \times BK$	-5.52×10^{-7} $(-5.57 \times 10^{-7}, -5.48 \times 10^{-7})$	***
NB^3	1.50×10^{-5} $(1.30 \times 10^{-5}, 1.70 \times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

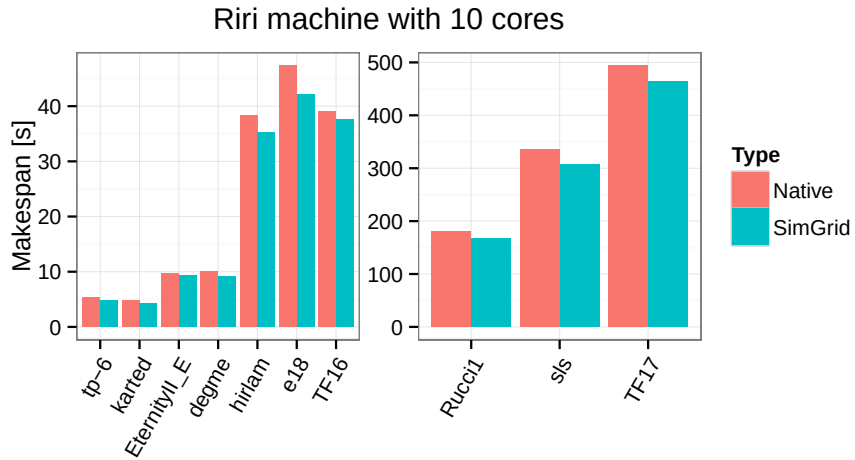
Comparing Kernel Duration Distributions

	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

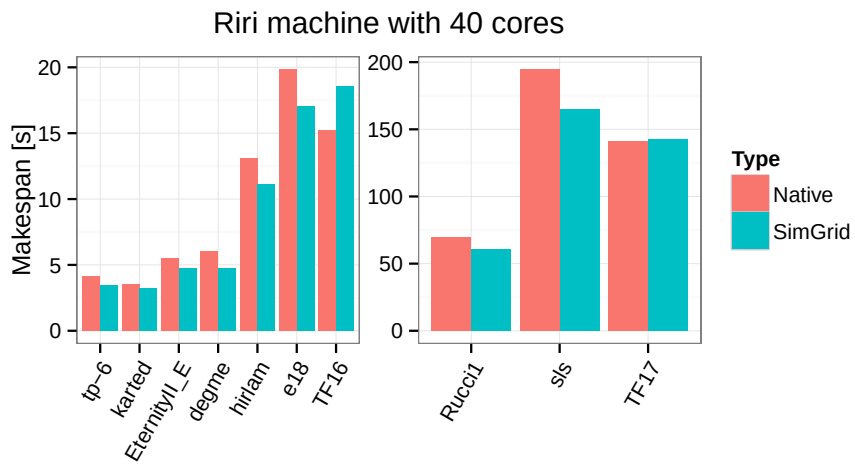


Overview of Simulation Accuracy



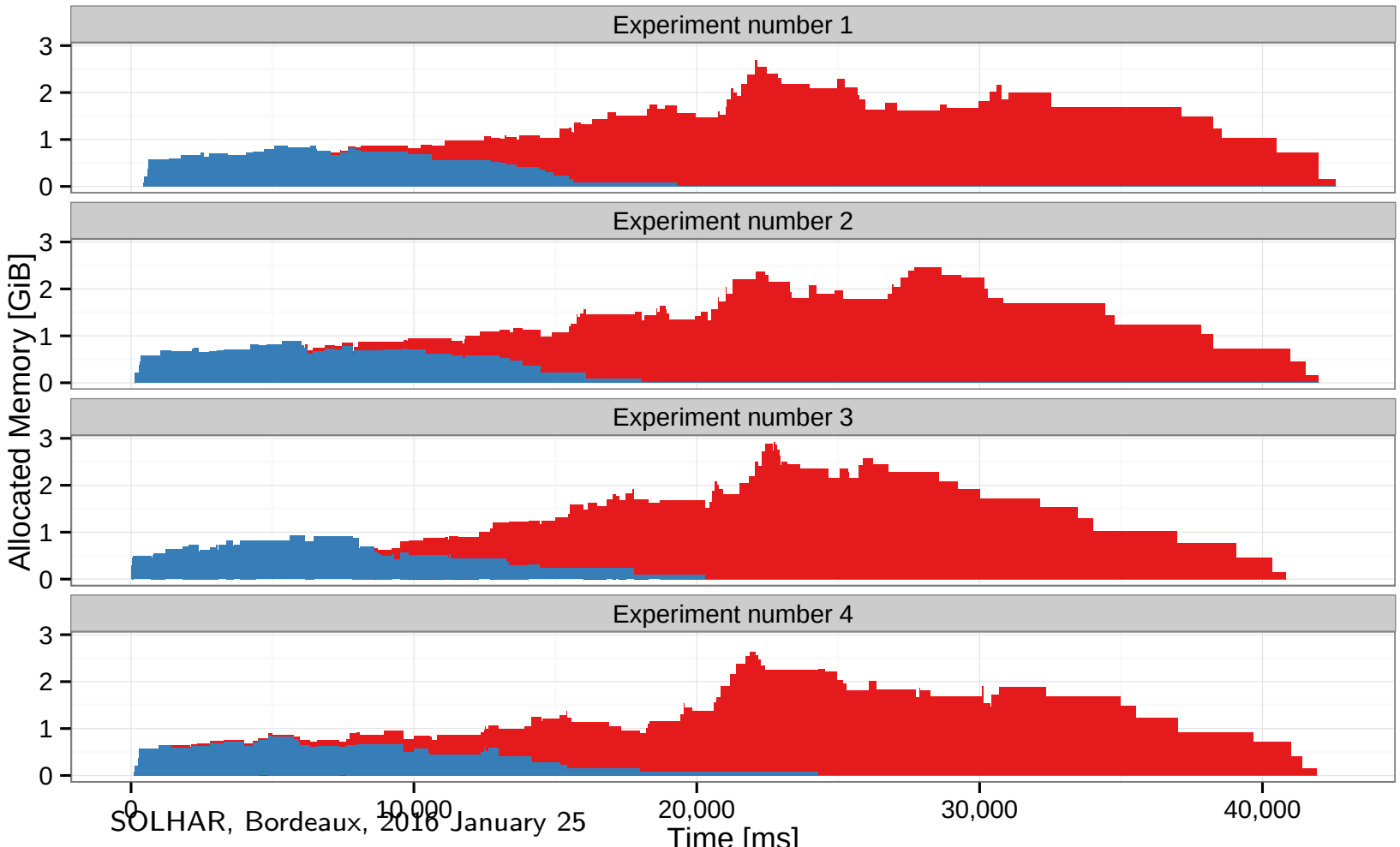


- Most of the time, simulation is slightly optimistic
- With bigger and architecturally more complex machines, error increases



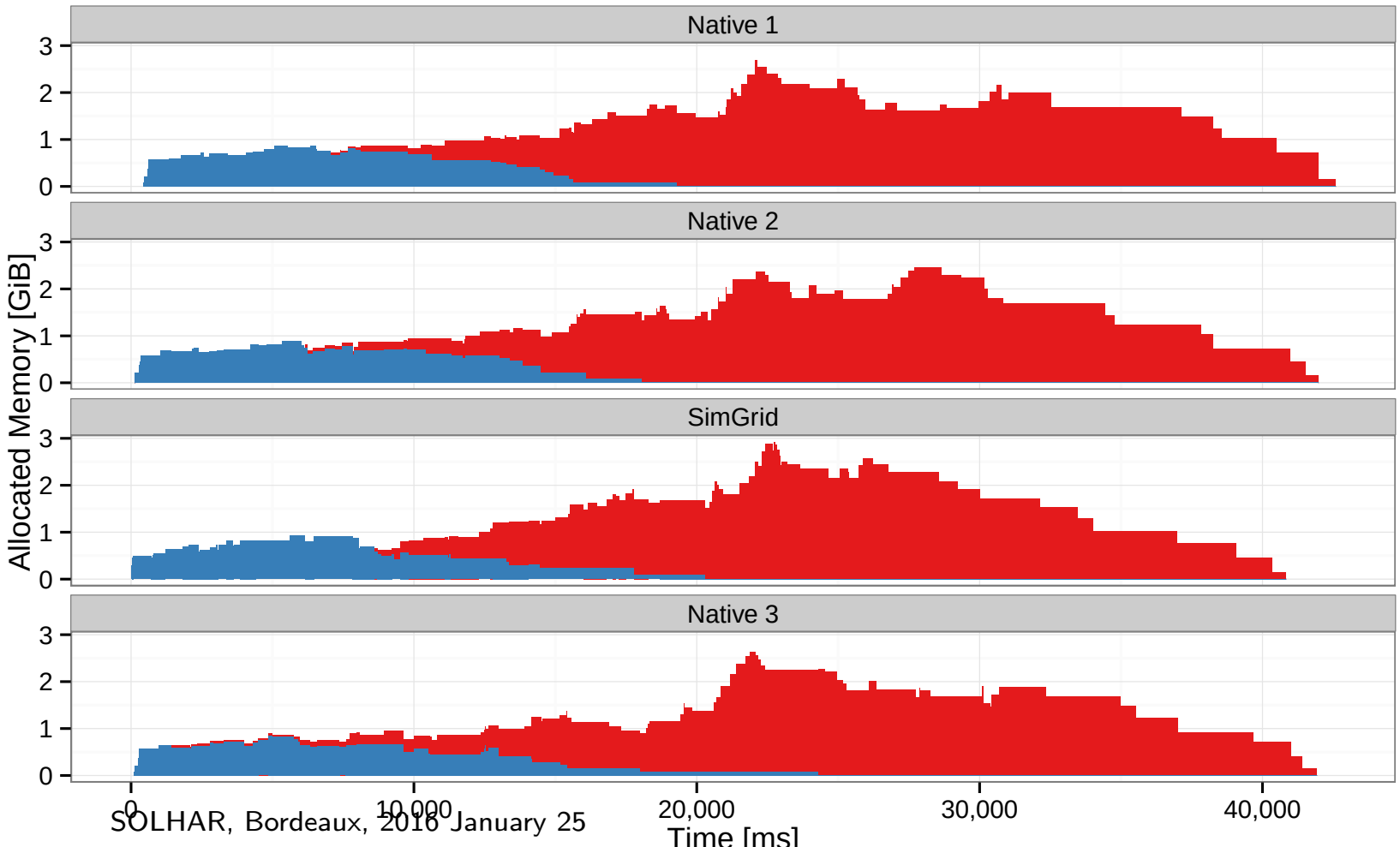
Studying Memory Consumption

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



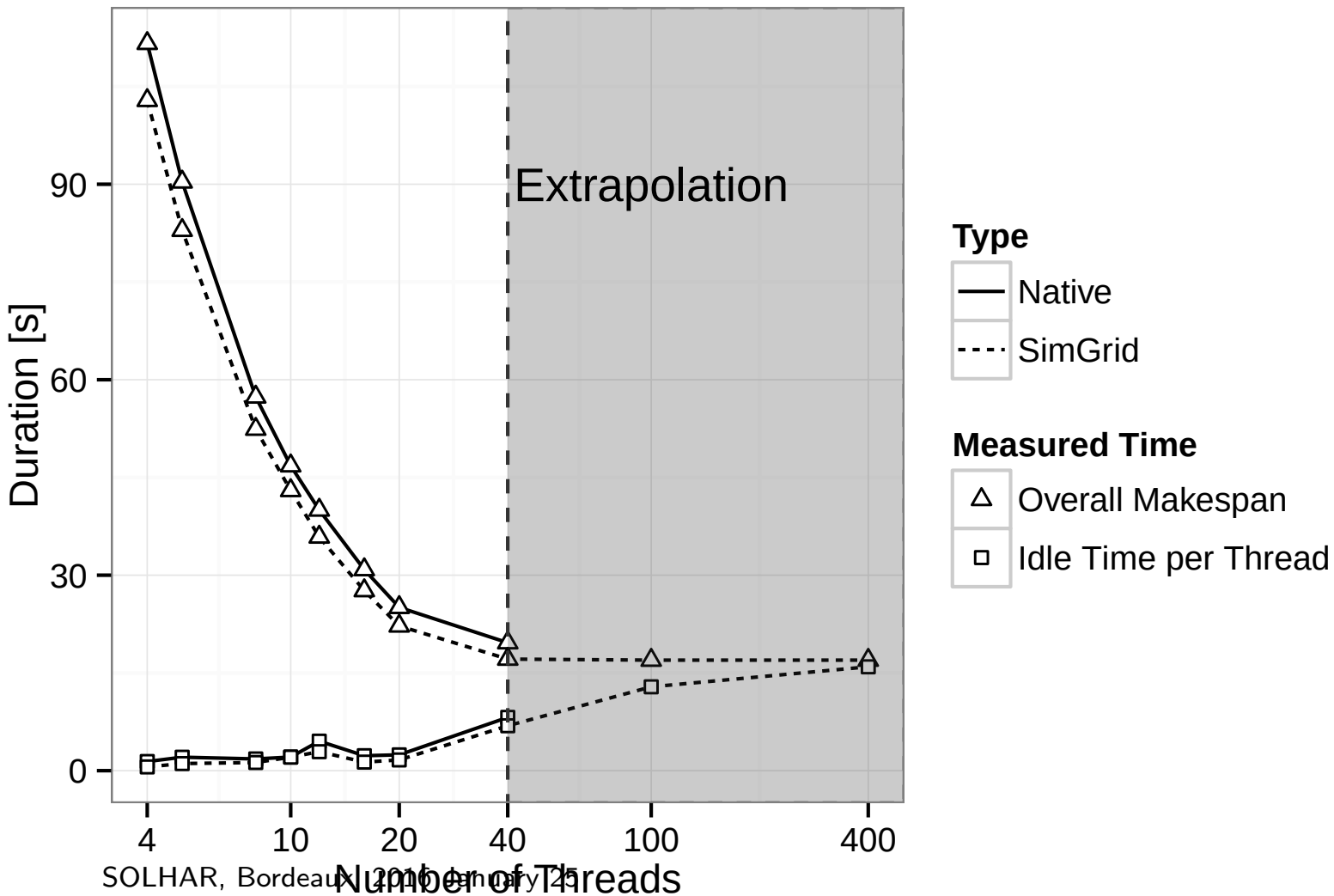
Studying Memory Consumption

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



Extrapolating to Larger Machines

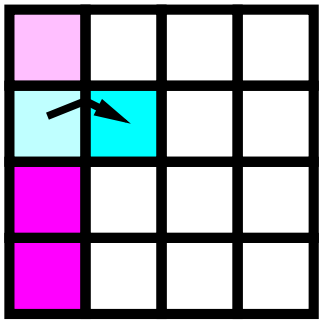
- Predicting performance in idealized context
- Studying the parallelization limits of the problem



Outline

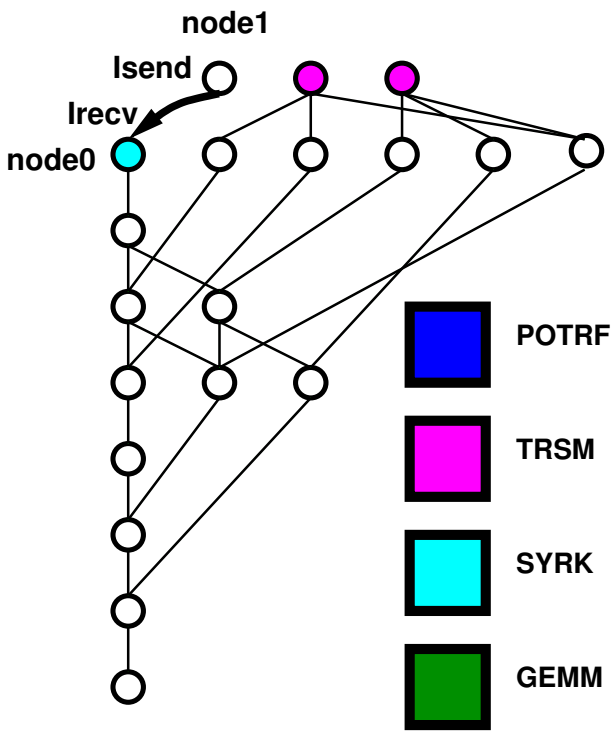
Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?



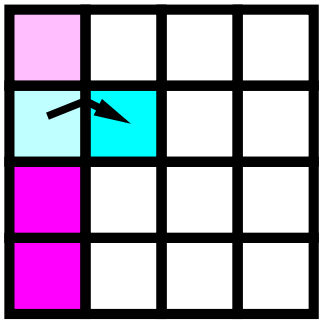
Questions

Existing methods



Address scalability

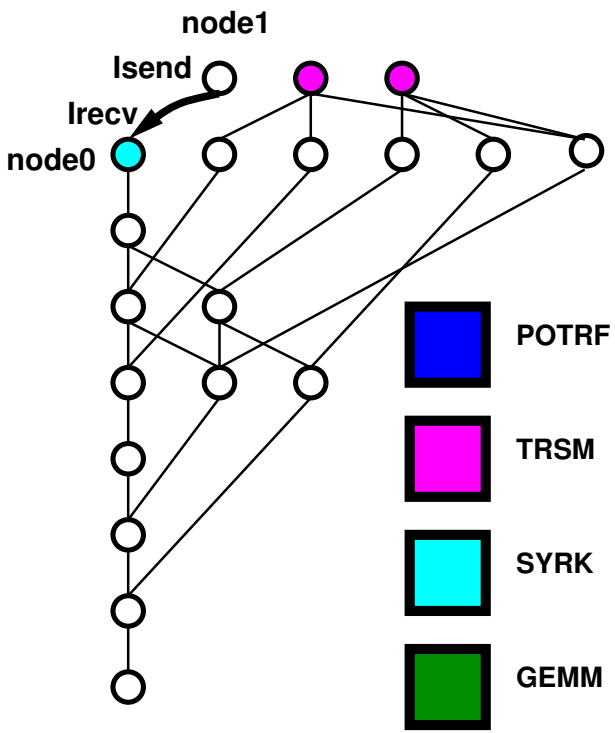
PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?



Questions

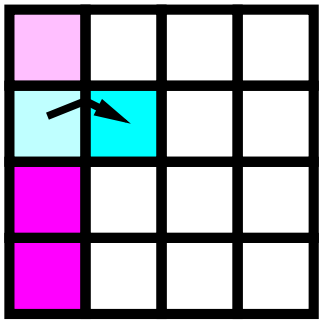
- How to establish the mapping?

Existing methods



Address scalability

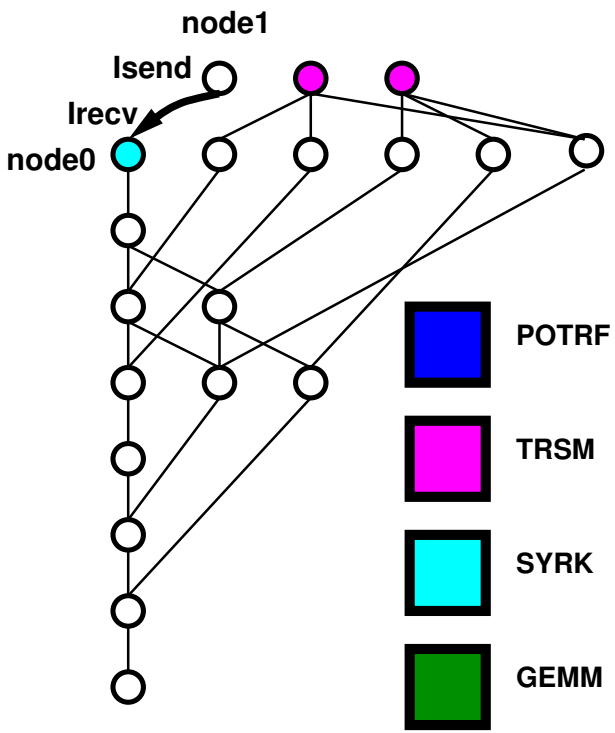
PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?



Questions

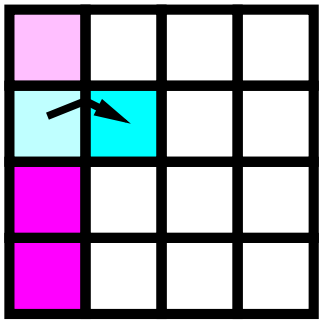
- How to establish the mapping?
- How to manage communications?

Existing methods



Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

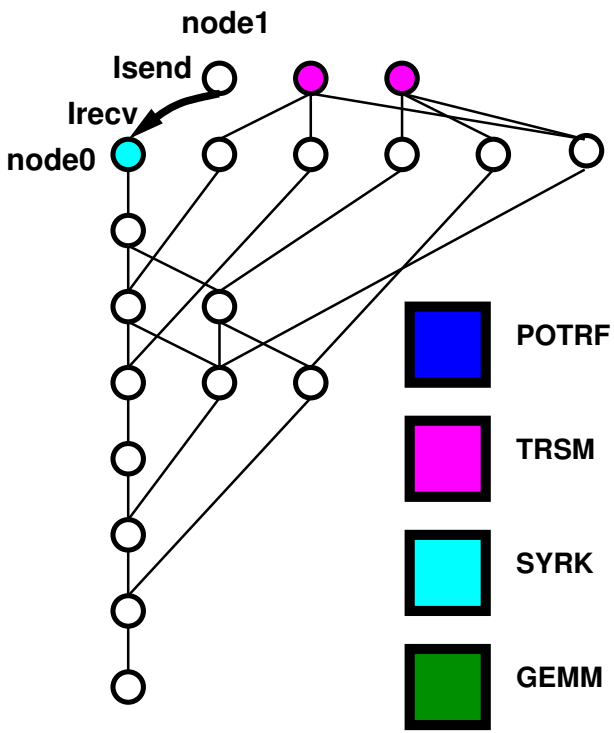


Questions

- How to establish the mapping?
- How to manage communications?

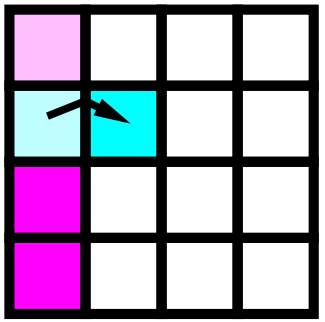
Existing methods

- Explicit MPI communications tasks



Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

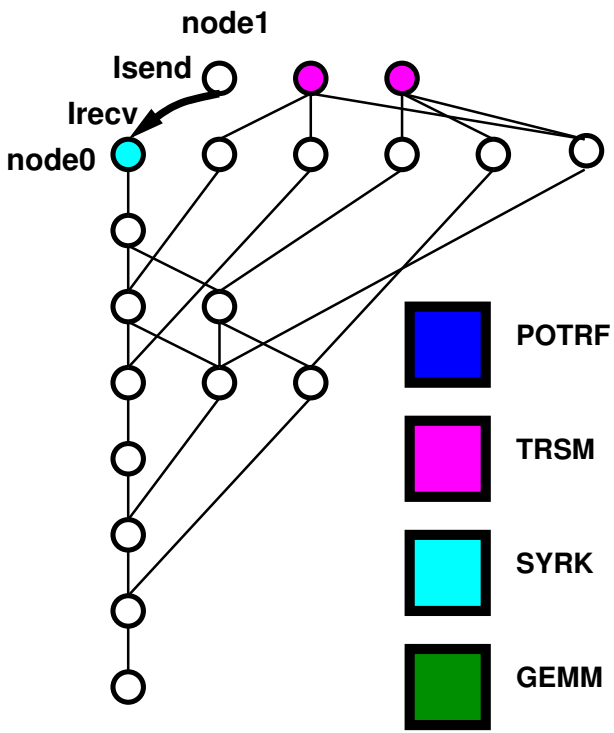


Questions

- How to establish the mapping?
- How to manage communications?

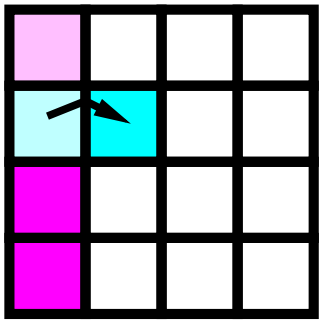
Existing methods

- Explicit MPI communications tasks
- PTG model (PaRSEC)



Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

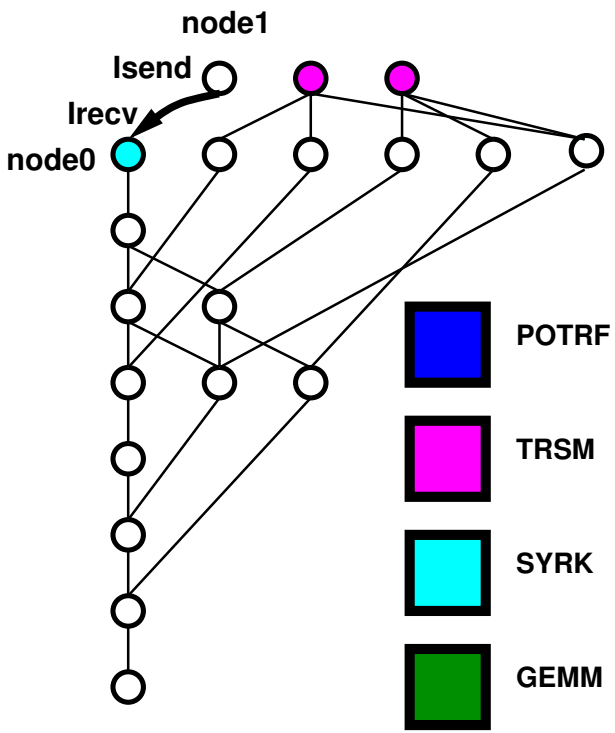


Questions

- How to establish the mapping?
- How to manage communications?

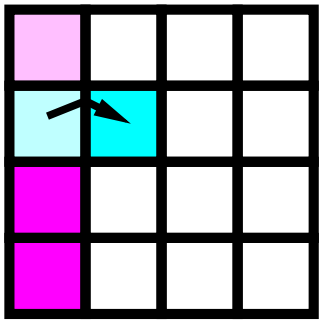
Existing methods

- Explicit MPI communications tasks
- PTG model (PaRSEC)
- STF model - Master/Slave (clusterSS)



Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

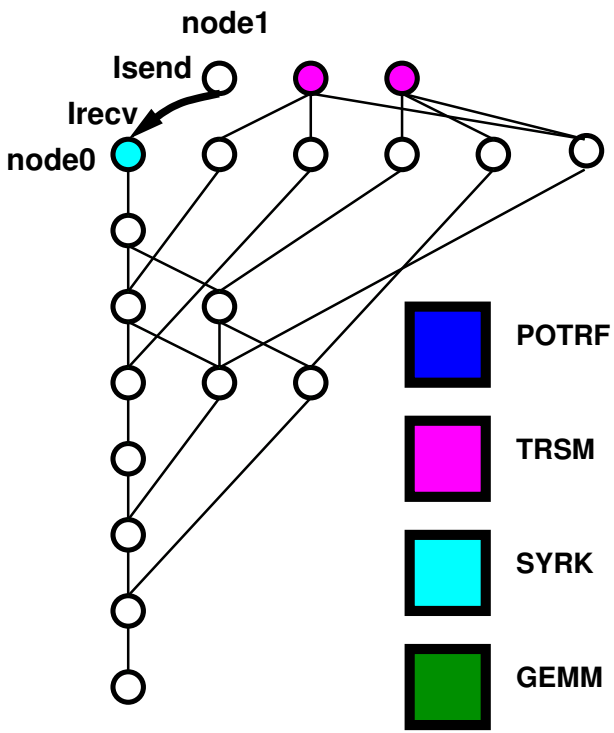


Questions

- How to establish the mapping?
- How to manage communications?

Existing methods

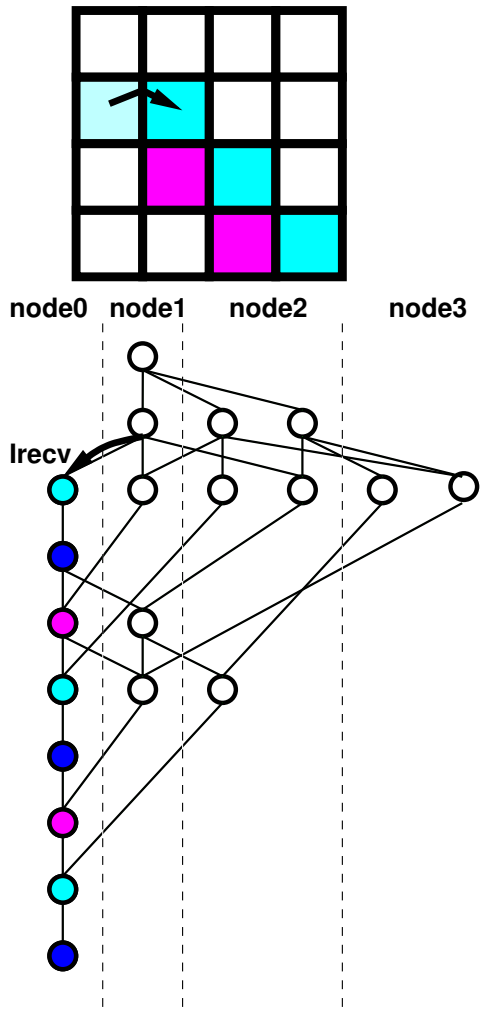
- Explicit MPI communications tasks
- PTG model (PaRSEC)
- STF model - Master/Slave (clusterSS)
- STF model - Replicated unrolling (StarPU)



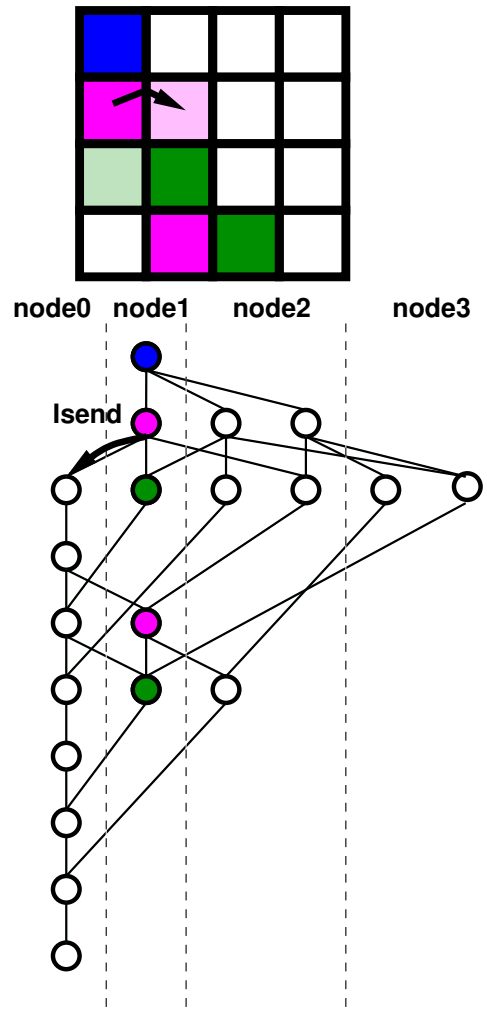
Data transfers between nodes

Method considered

- All nodes unroll the whole task graph
- They determine tasks they will execute
- They can infer required communications
- No negotiation between nodes (not master-slave)
- Unrolling can be pruned



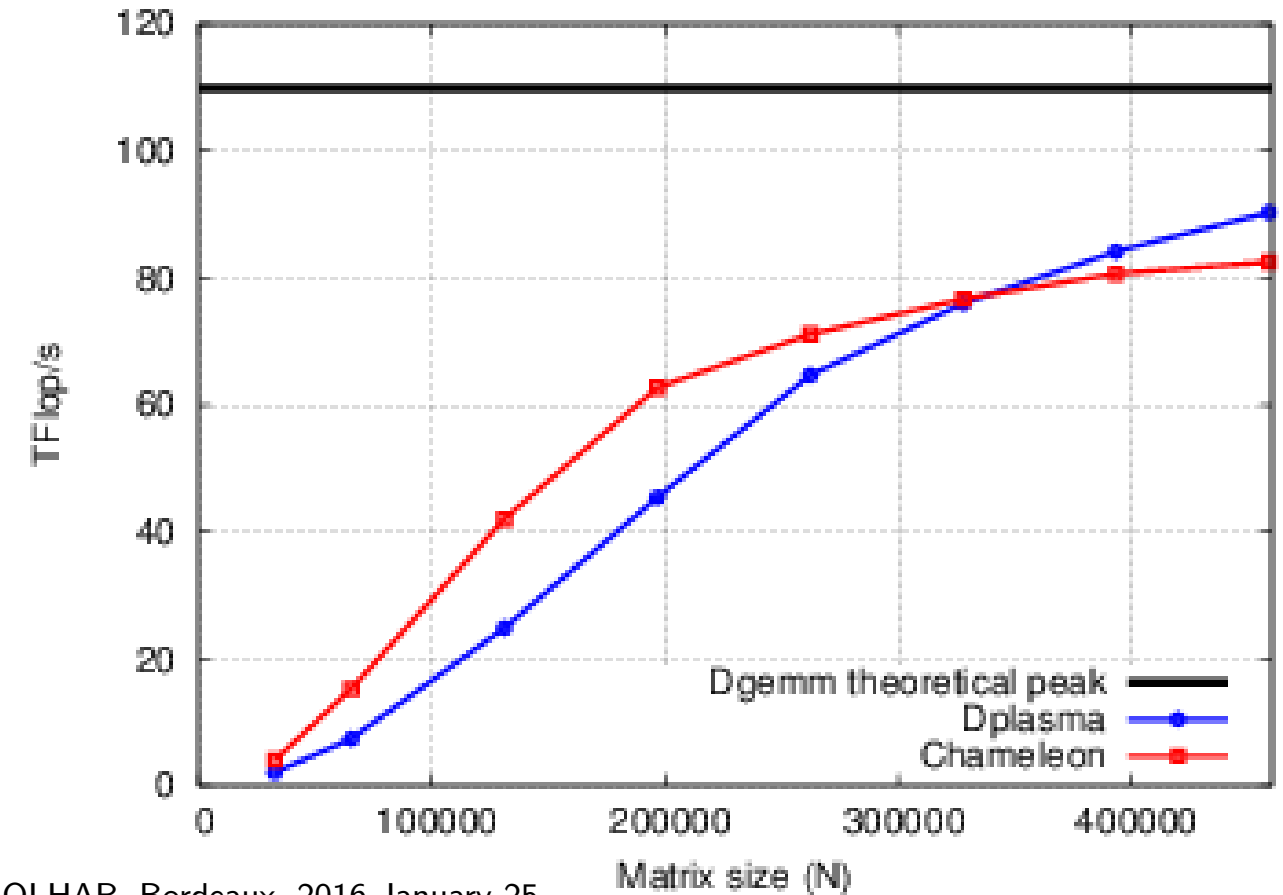
Node 0 execution



Node 1 execution

Performance

- 144 TERA-100 Hybrid nodes
  - collaboration with CEA-CESTA
- CPU: 2 Quad-core Xeon E5620 (per node)
- GPU: 2 NVIDIA Tesla M2090 (per node)



Outline

StarPU-SimGrid for MPI applications

Postdoc L. Stanisic

- Technically challenging as it combines two separate SimGrid interfaces (modules) :
  - MSG - internode computation and communication (CPU-GPU)
  - SMPI - intranode communication
- SimGrid not initially designed for such cases
- Need to improve SimGrid internals

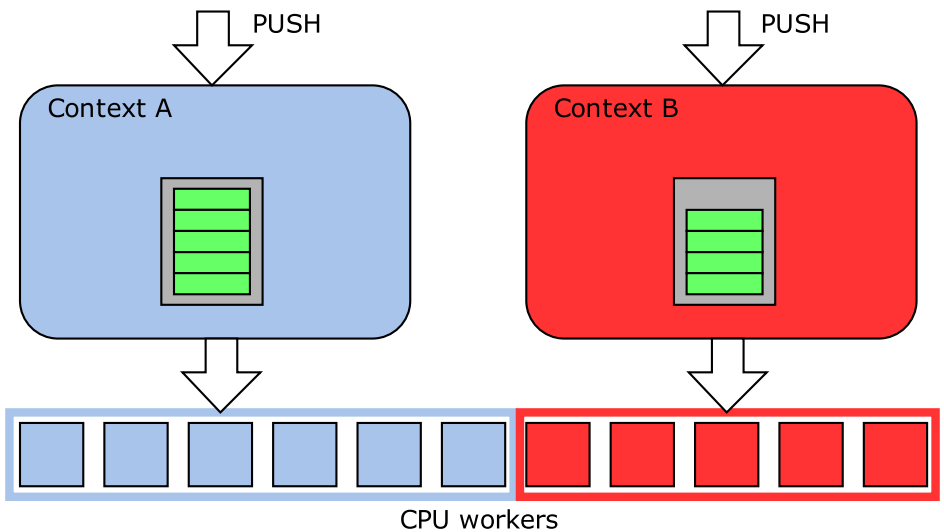
Current status : have a prototype, working to make it applicable in general

Outline

StarPU contexts - New use case

PhDs A. Hugo and T. Cojean (HiePACS / Storm)

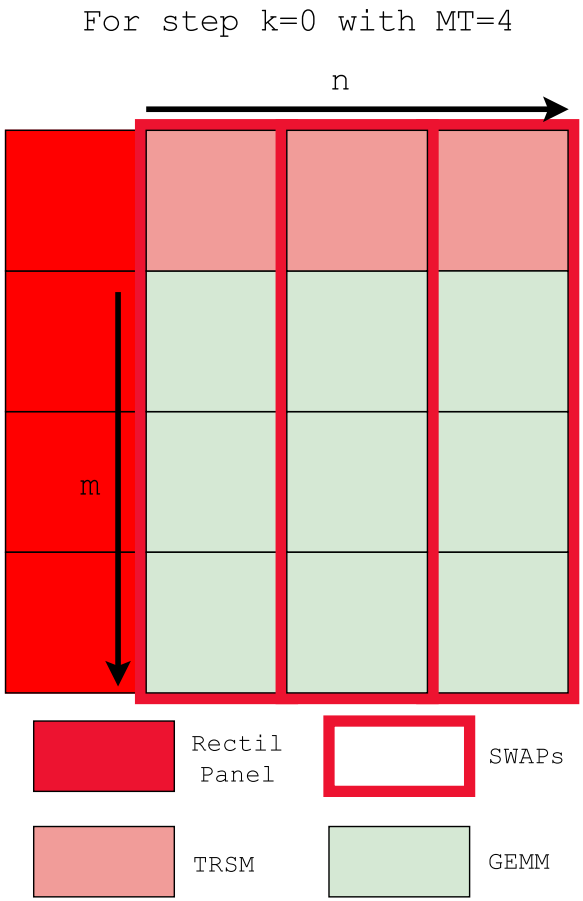
- We add a new use case
- Based on existing architecture and primitives
- No StarPU scheduler (managed by “internal” runtime)
- Particular worker management : one awake StarPU worker for fork-join parallelism



Tile LU factorization with numerical Partial Pivoting

PhDs A. Hugo and T. Cojean (HiePACS / Storm)

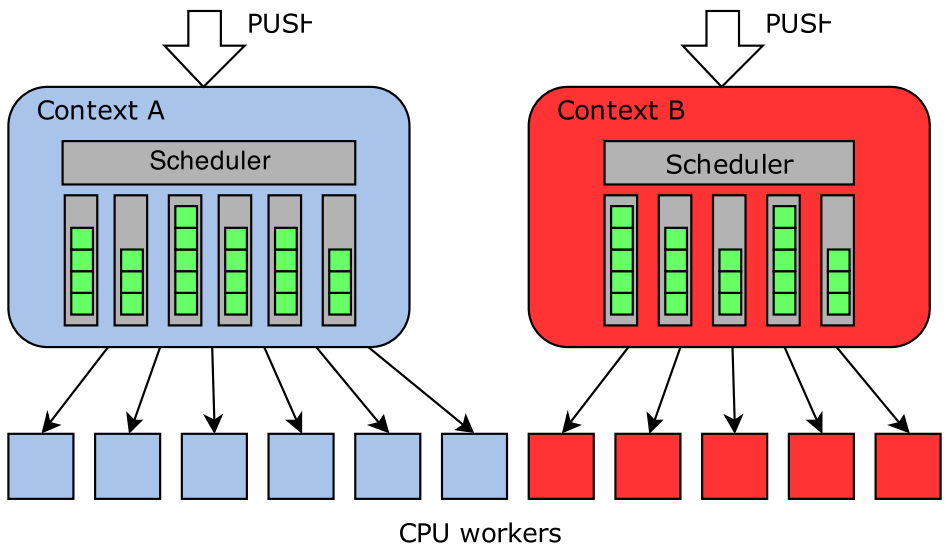
```
1   //If A is a square matrix
2   for k : 0 → MT-1
3
      RECTIL_PANEL(submatrix(A_{k,k}, ..., A_{MT,k}), pivs[k])
4
5     //Waits for the pivots
6     data_acquire(pivs[k])
7
8     for n : k+1 → MT-1
9       //SWAP if required according
      to pivs[k] array
10      SWAPs(submatrix(A_{k,n}, ..., A_{MT,n}))
11      TRSM(A_{k,k}, A_{k,n})
12      for m : k+1 → MT-1
13      GEMM(A_{m,k}, A_{k,n}, A_{m,n})
14
15  for k : 0 → MT-1
16    for n : 0 → k
17      SWAPs(submatrix(A_{k,n}, ..., A_{MT,n}))
18
```



StarPU contexts

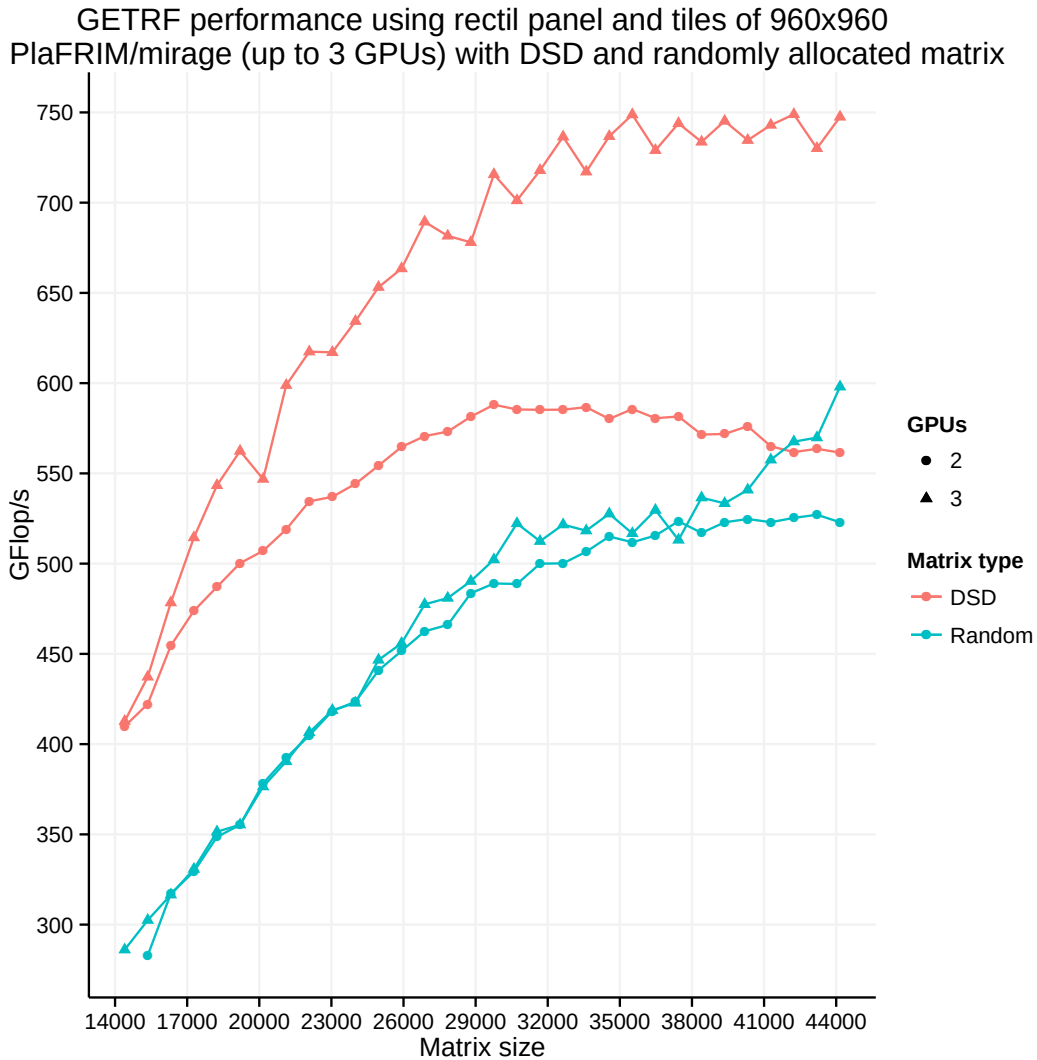
PhDs A. Hugo and T. Cojean (HiePACS / Storm)

- StarPU possesses workers and contexts
- Workers execute a code on a resource. They represent :
  - a CPU core
  - a GPU, ...
- Contexts groups workers and schedules tasks on them
  - Isolate concurrent parallel codes
  - Can expand and shrink (resource management)

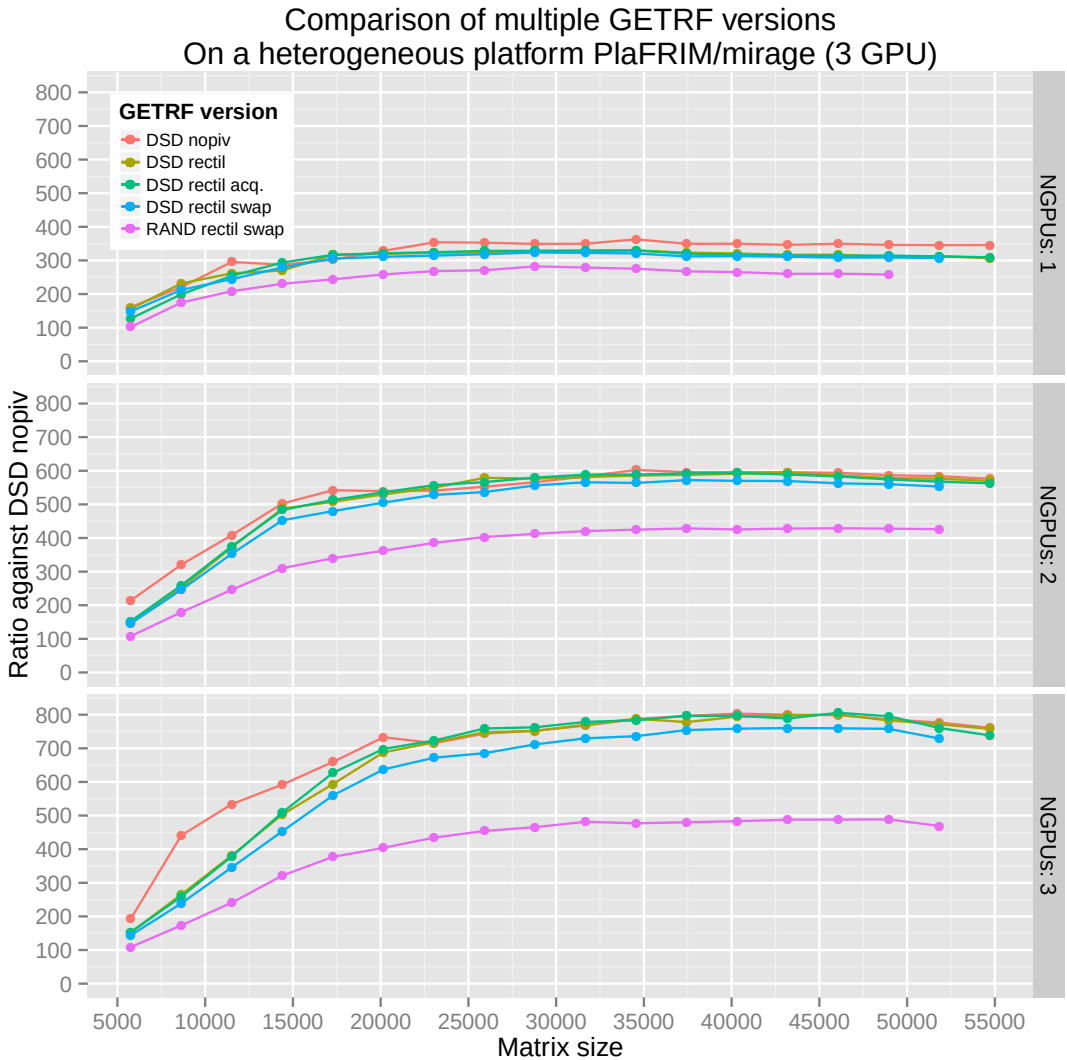


Performance results - Heterogeneous node

DSD matrix vs. Randomly generated



Performance results - Plafrim Mirage node



Outline

Simulating parallel workers

Postdoc L. Stanisic and PhD T. Cojean

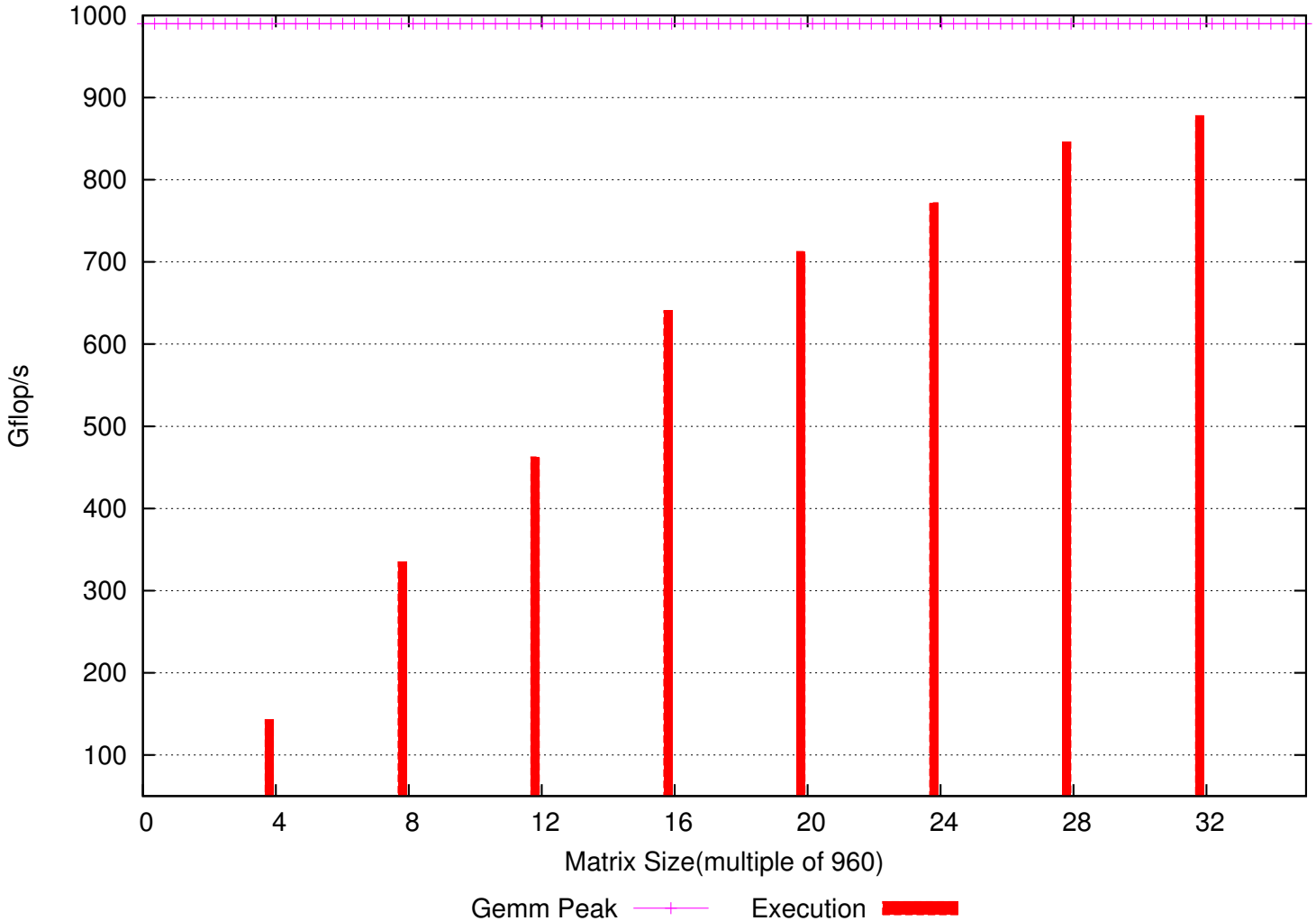
- Can apply StarPU-SimGrid for a single node approach
- StarPU knows if the worker is parallel or not
- StarPU will tell SimGrid which timings needs to be inject

Current status : looks as easy to be implemented, but need to verify with StarPU experts (Samuel)

Outline

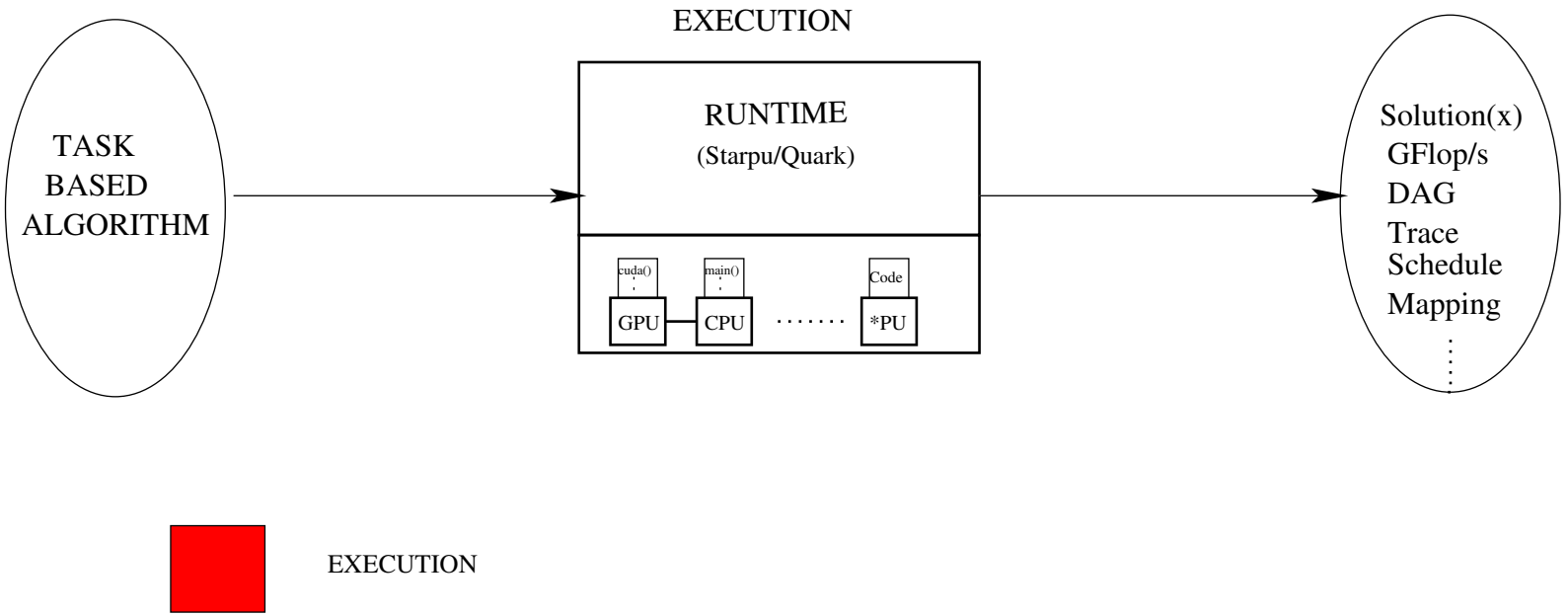
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



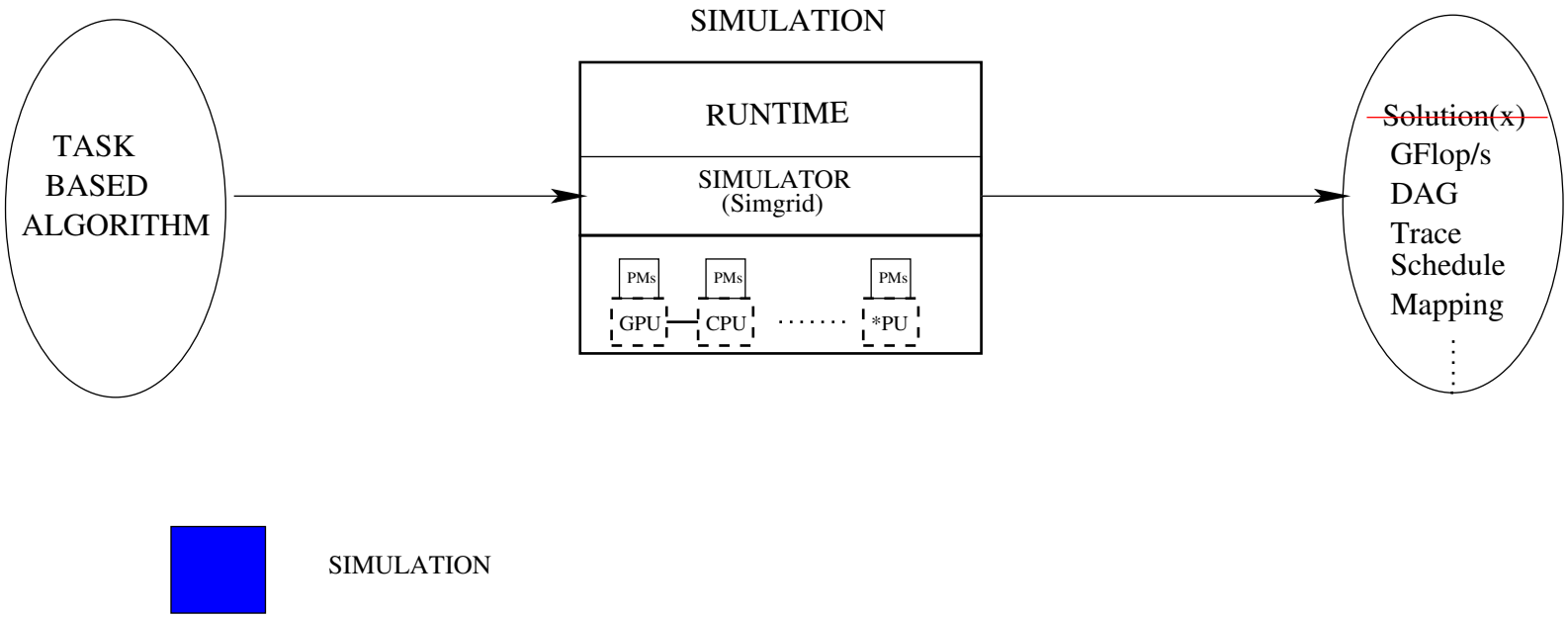
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



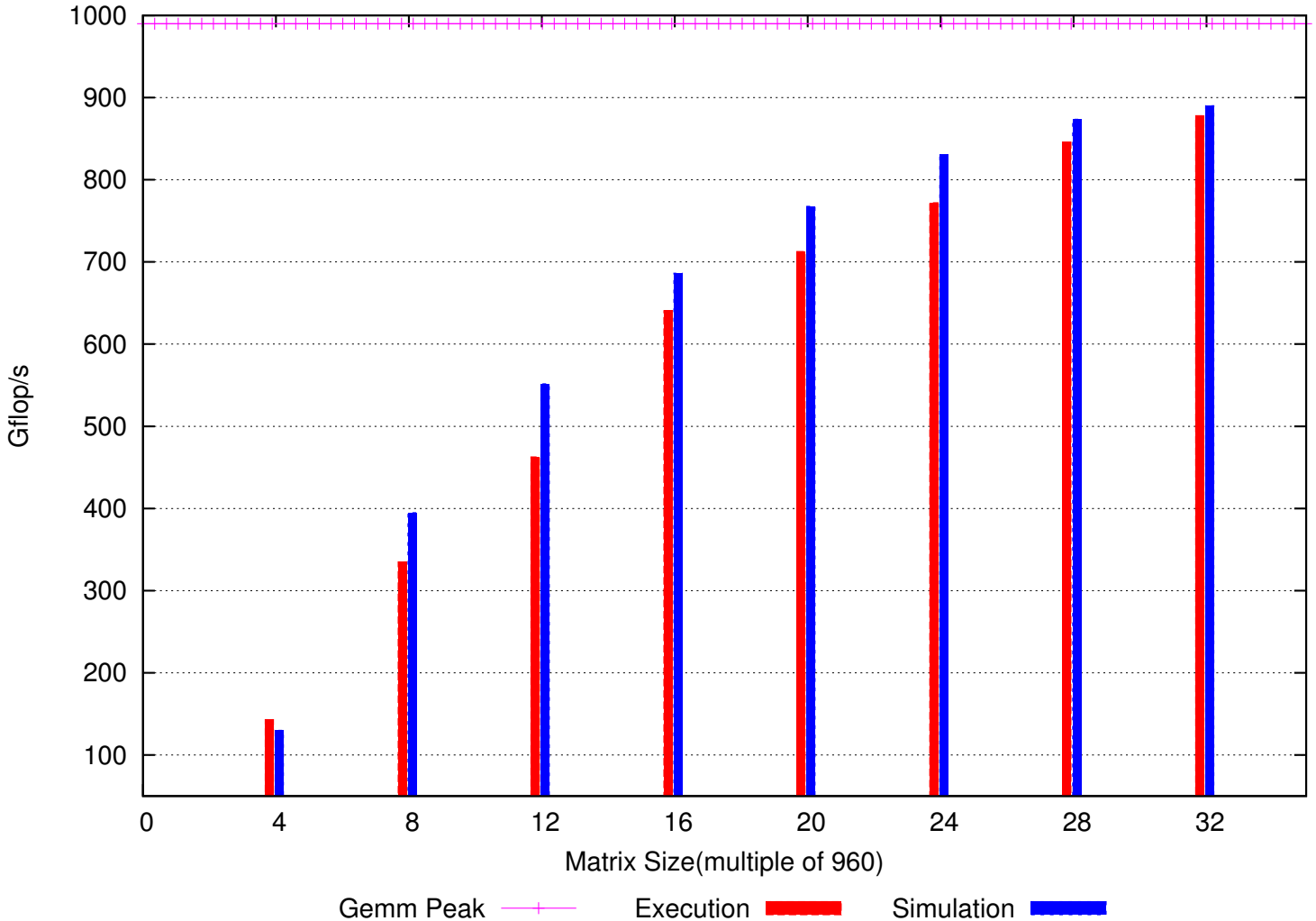
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



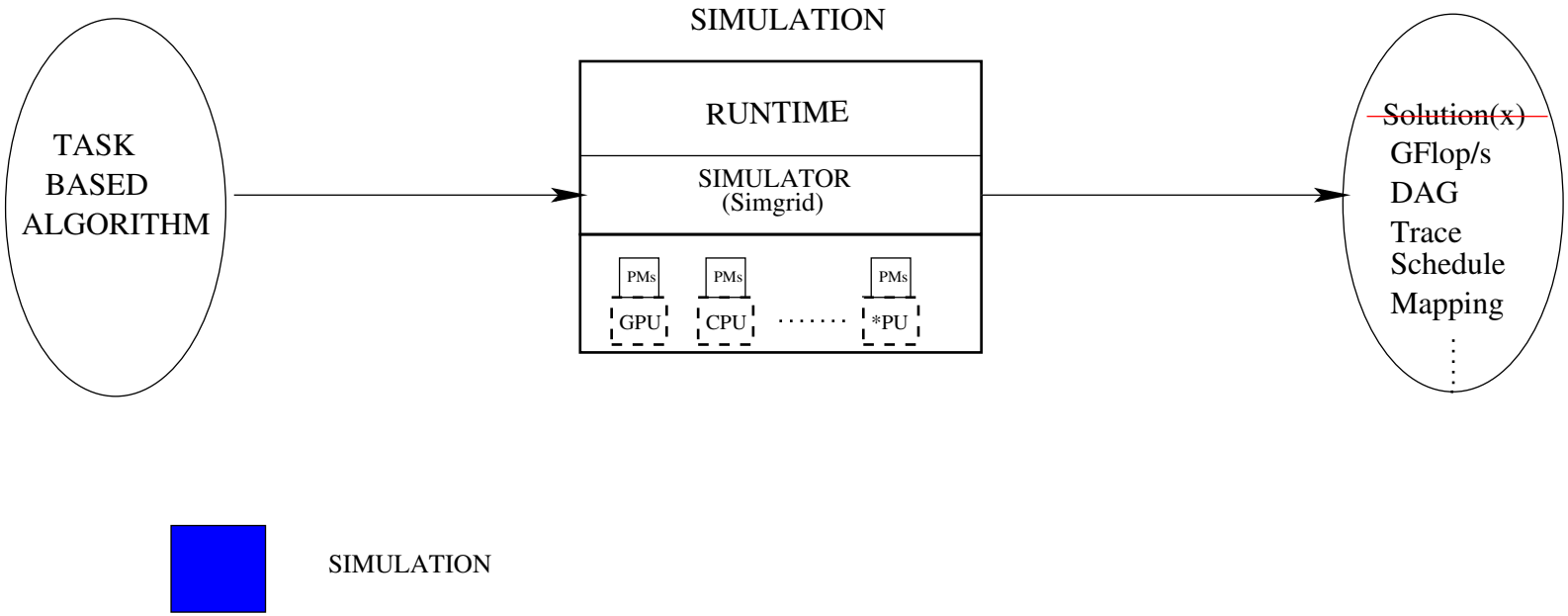
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



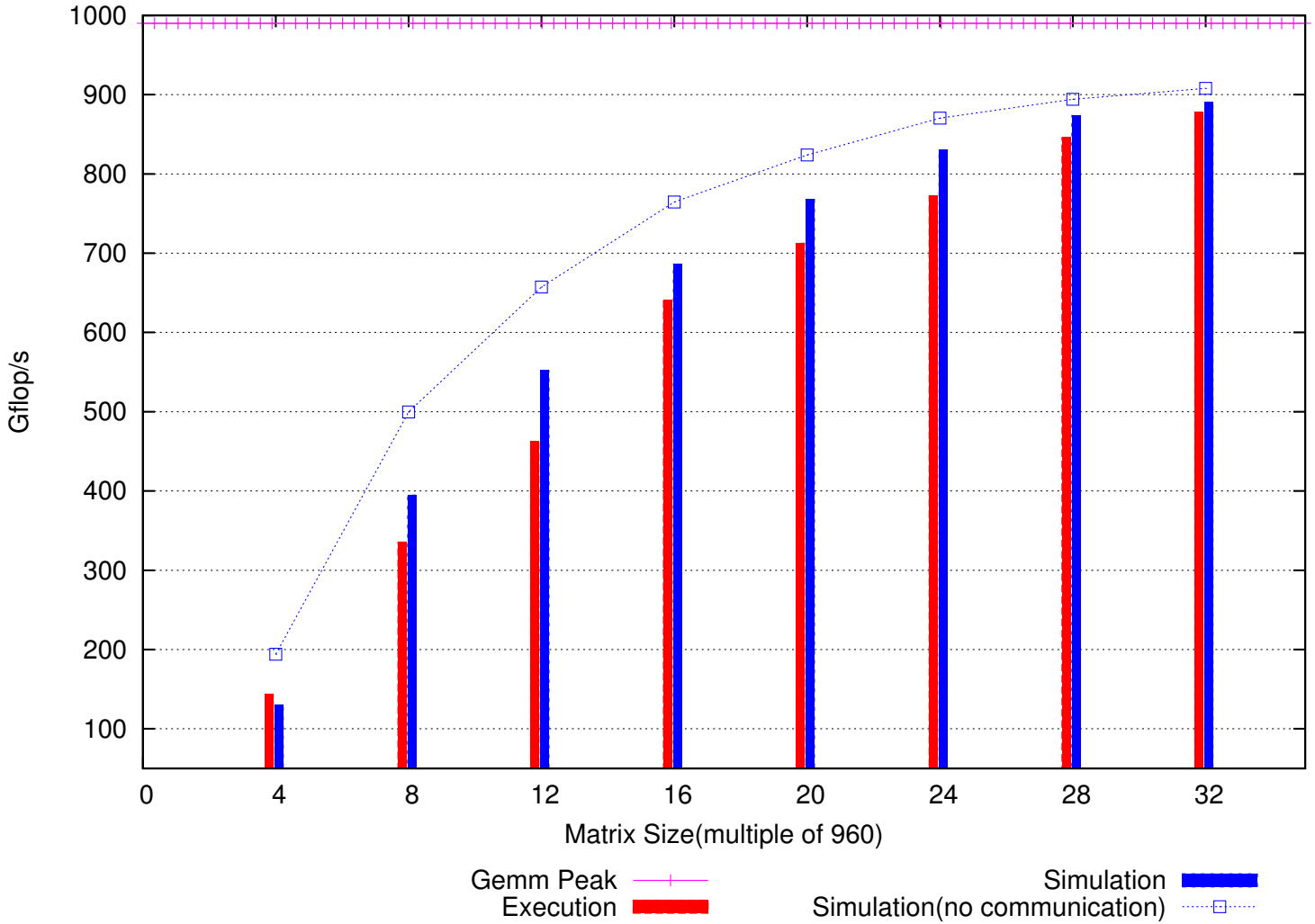
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



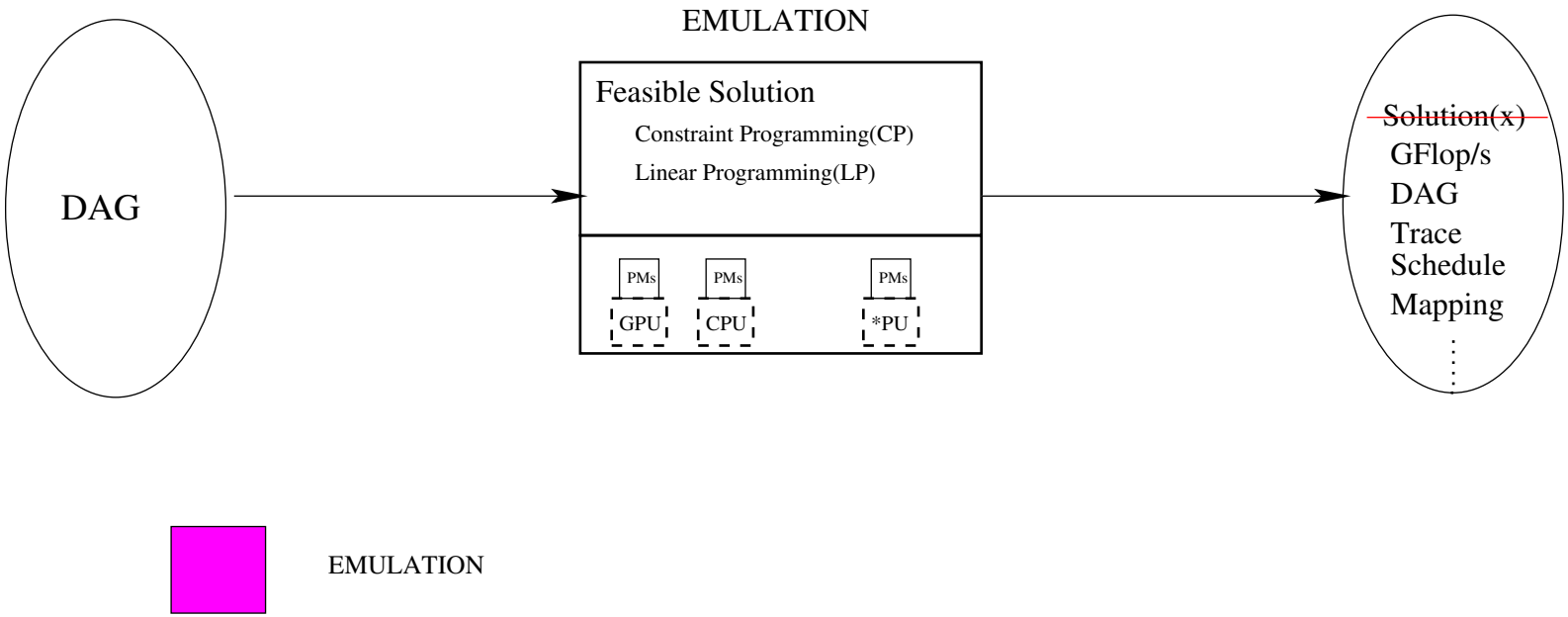
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



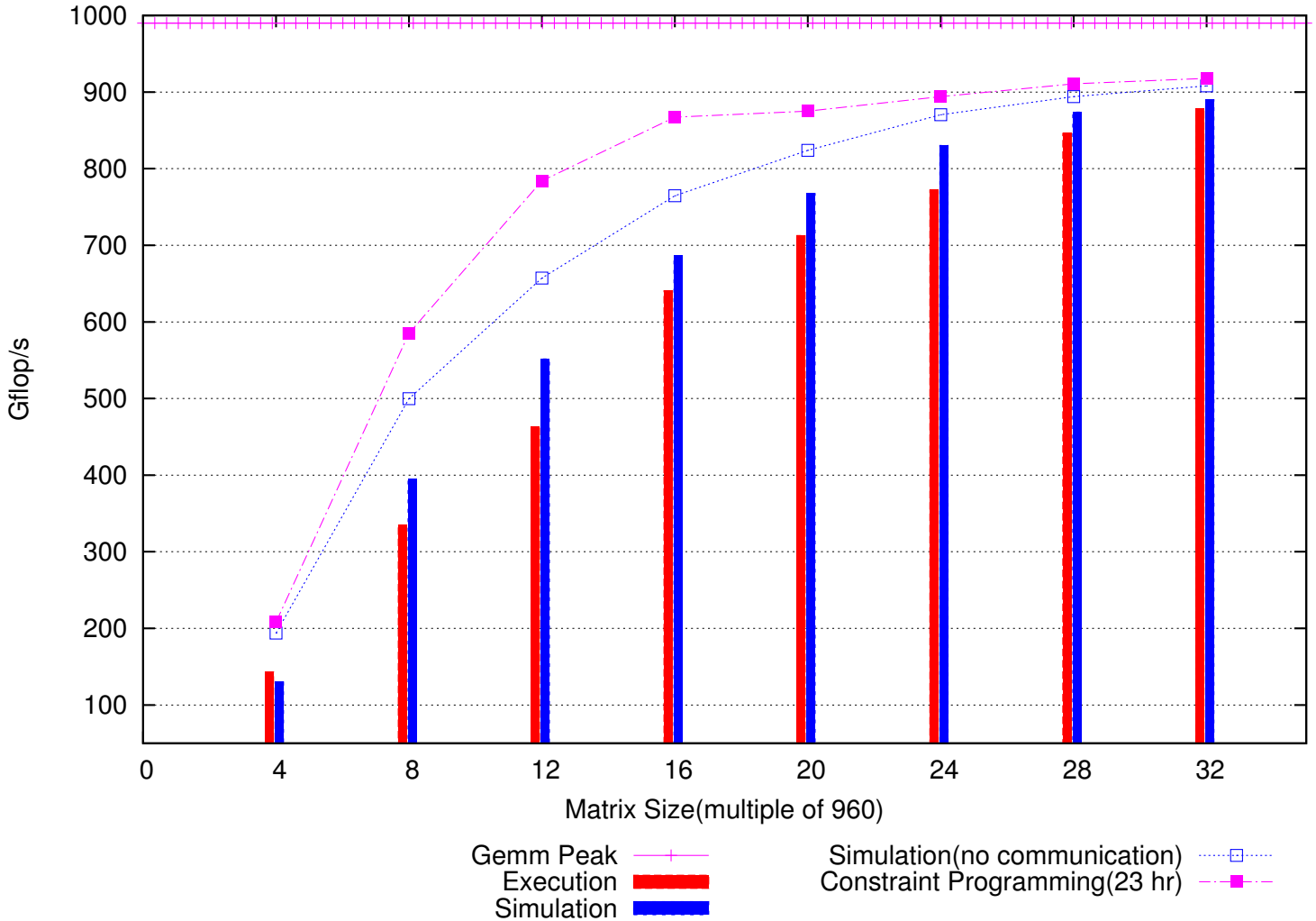
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



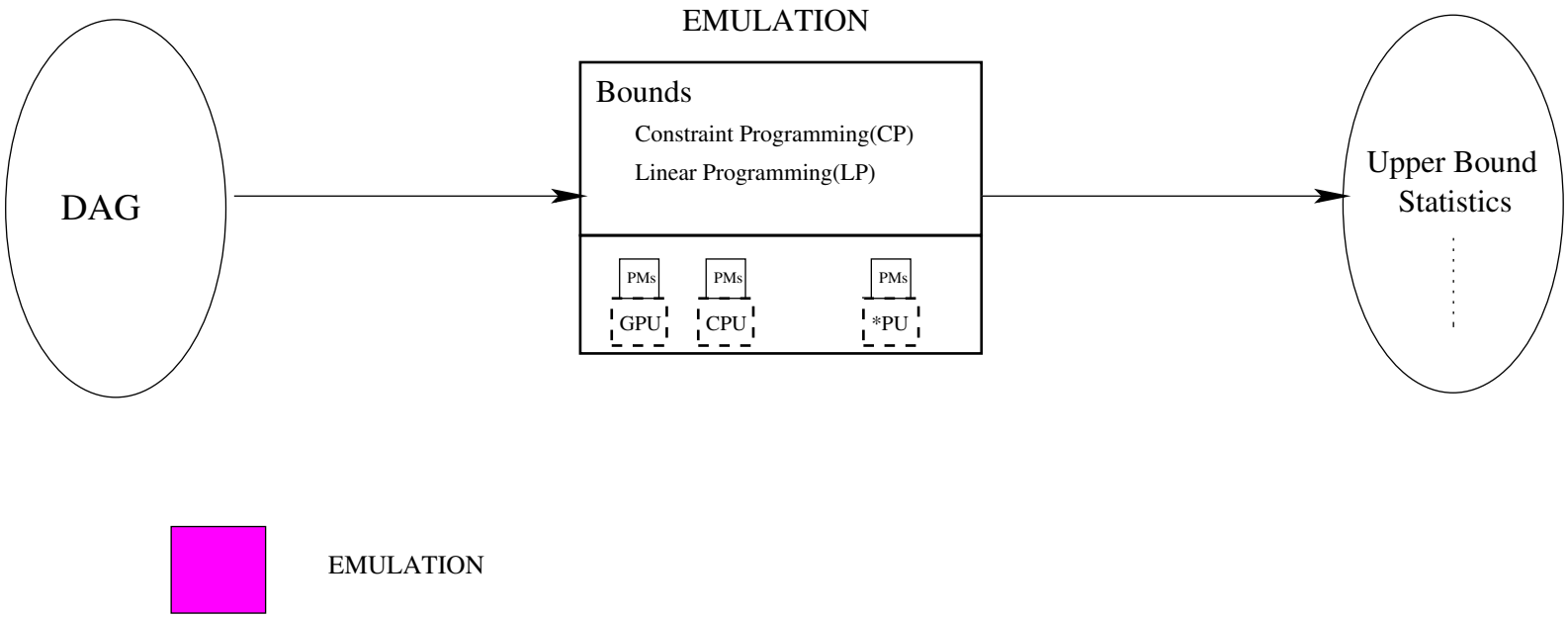
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



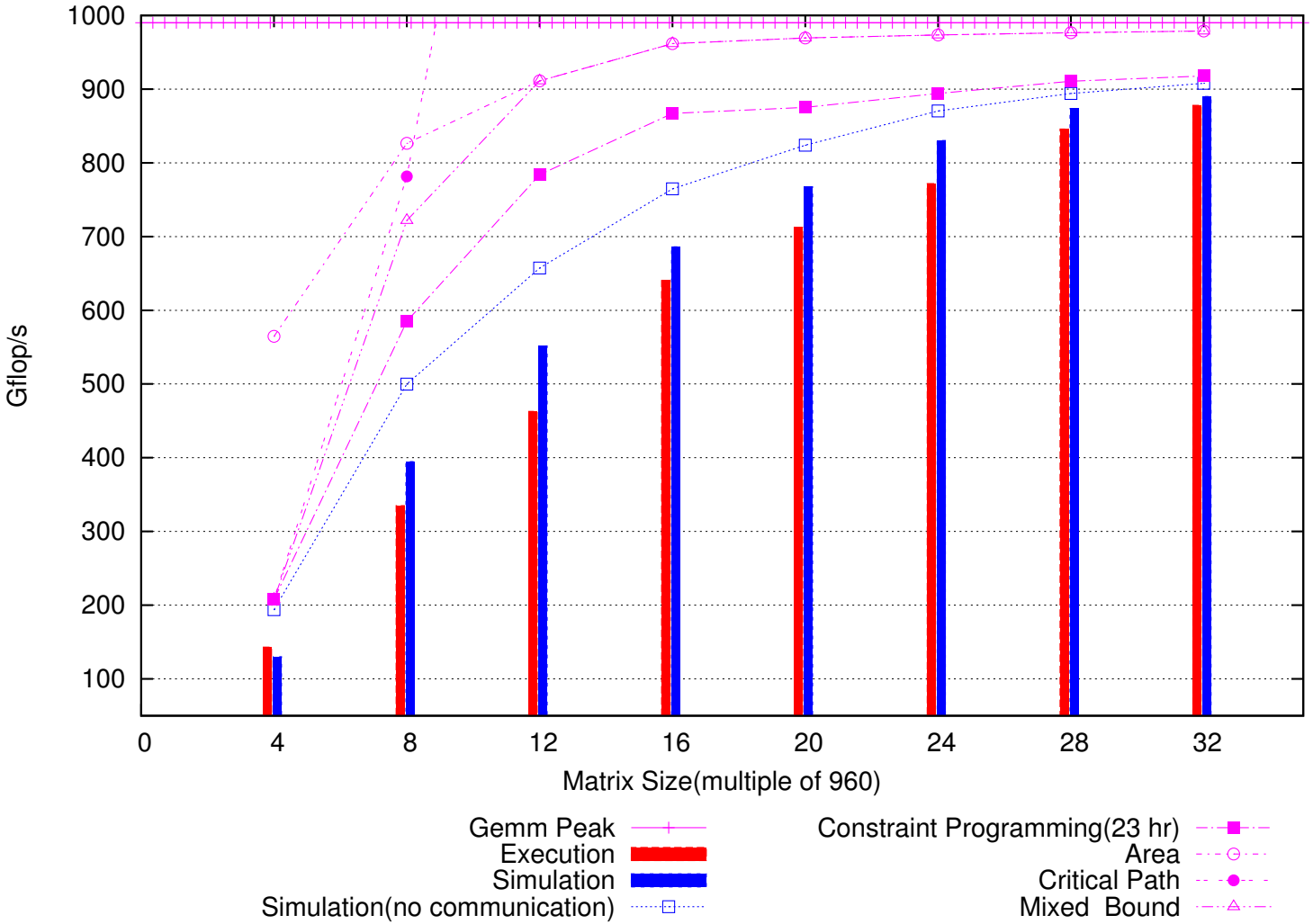
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)



Outline

Look-ahead scheduling

PhD S. Kumar

1. Stop execution for an important decision
2. Launch multiple parallel simulation
3. Get simulated makespans
4. Continue execution with the fastest solution

- Simple code based only on DAG of tasks
- Using StarPU-SimGrid (communications taken into account)
  - Simple forking of SimGrid process should work

Current status : started looking at the changes needed for the StarPU code

Outline

Conclusion and perspectives (1/2)

Beyond the solver stack:

- ANR Solhar
  - qr_mumps - PhD F. Lopez (N7 - IRIT)
  - PaStiX - PhD X. Lacoste (HiePACS)
- DIP project - PhDs L. Boillot & S. Nakov (HiePACS / Magique 3D)
- Flusepa - PhD J.-M. Couteyen (HiePACS / Airbus)
- Boltzmann transport equation - PhD S. Moustafa (EDF / HiePACS)
- Aerosol - PhD Damien Genet (Bacchus / HiePACS)
- FastLA associate team - Inria / LBNL / Stanford

Conclusion and perspectives (2/2)

AT MORSE (2011-2013)

- Numerical algorithms - HiePACS - UTK - UCD - KAUST
- Runtime systems - Runtime - UTK
- Automatic parallelization - UTK

Conclusion and perspectives (2/2)

ANR Solhar (2013-2016)

- Numerical algorithms - HiePACS - IRIT - Roma
- Runtime systems - Runtime
- Scheduling - Realopt - Roma

Conclusion and perspectives (2/2)

AT MORSE (2014-2016)

- Numerical algorithms - HiePACS - UTK - UCD - KAUST
- Runtime systems - Runtime - UTK
- Automatic parallelization - UTK
- Scheduling - Realopt

Conclusion and perspectives (2/2)

Achieving a software stack running @ exascale

- Numerical algorithms - HiePACS - UTK - UCD - KAUST - IRIT - Roma
- Runtime systems - Storm - UTK
- Automatic parallelization - Corse - UTK
- Scheduling - Realopt - Roma
- Communication - Tadaam
- Simulation and reproducibility - Polaris
- Resilience - HiePACS - UTK
- Autotuning - Polytechnique Montreal