MORSE

Matrices Over Runtime Systems @ Exascale

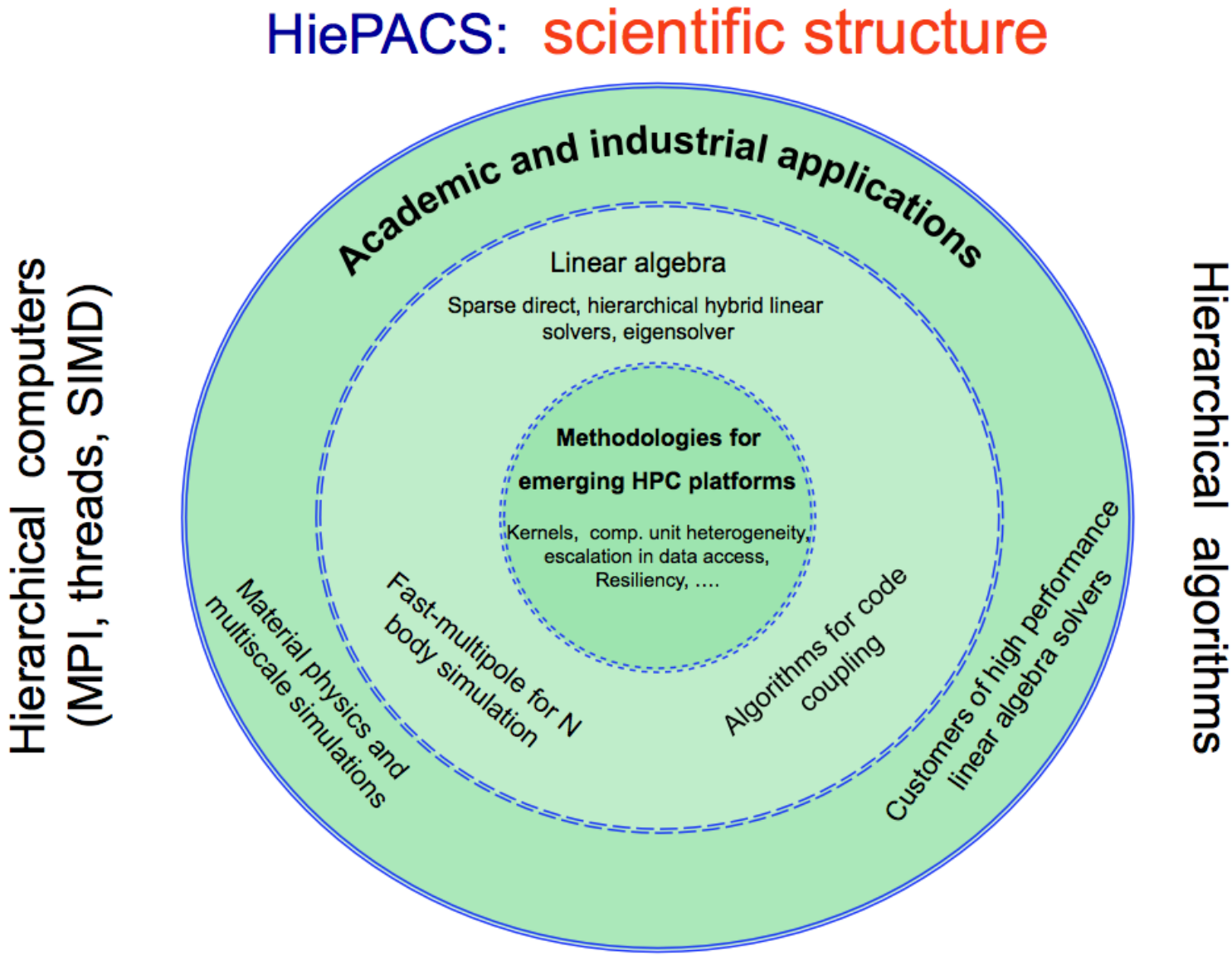
SUD, Frejus, 2016 January 19

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

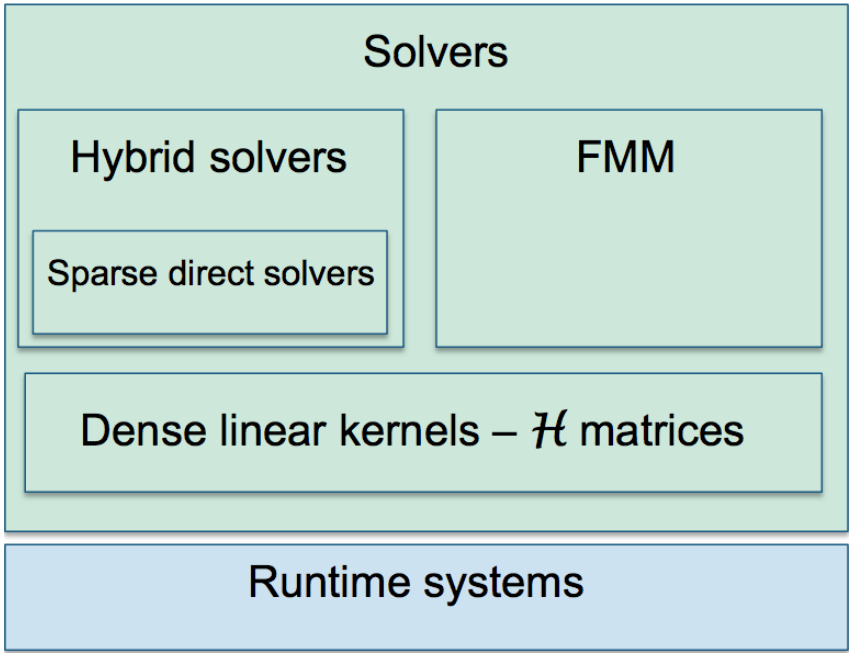
Outline

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



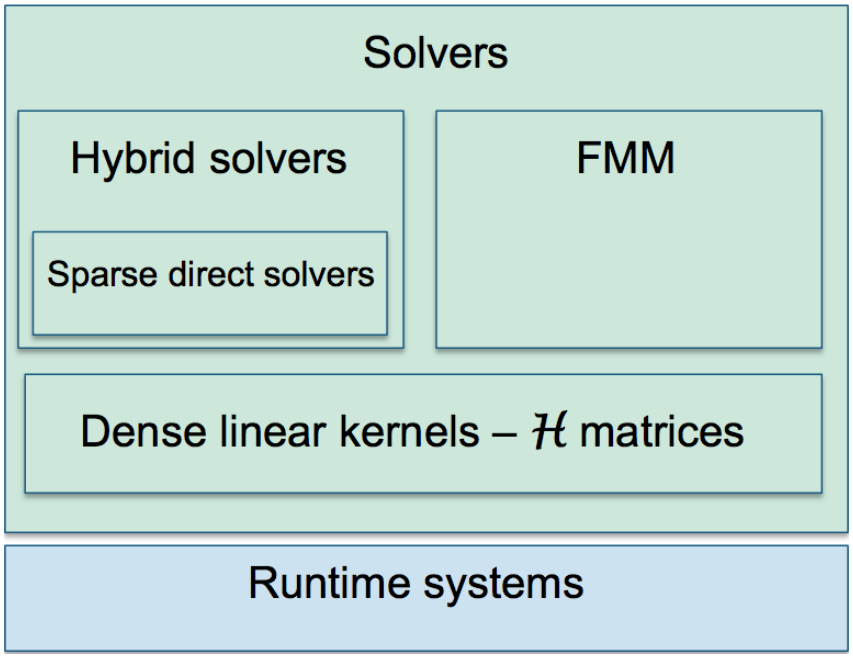
A view of HiePACS solvers



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)

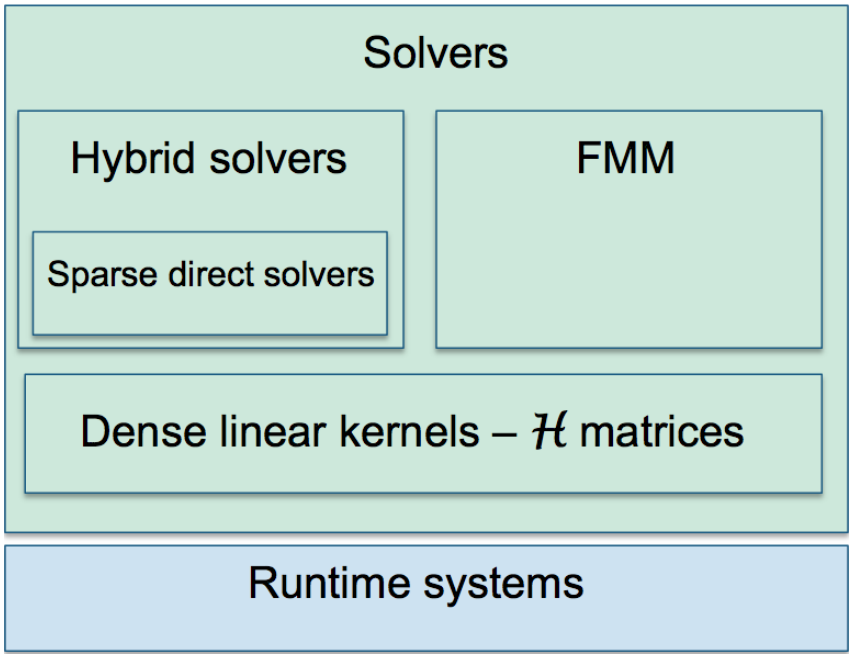
A view of HiePACS solvers



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

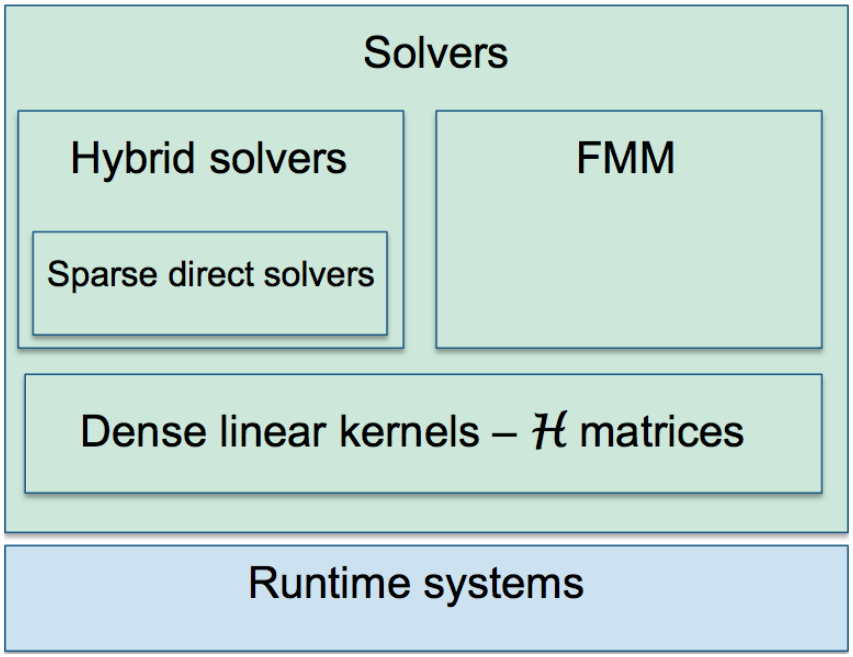
A view of HiePACS solvers



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

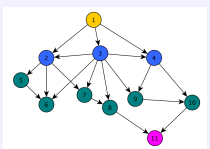
Linear algebra

$$AX = B$$

Sequential-Task-Flow

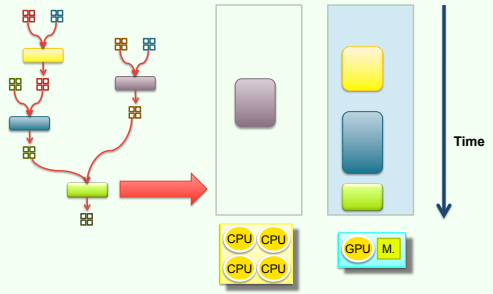
```
for (j = 0; j < N; j++)
  Task (A[j]);
```

Direct Acyclic Graph



→

Runtime systems

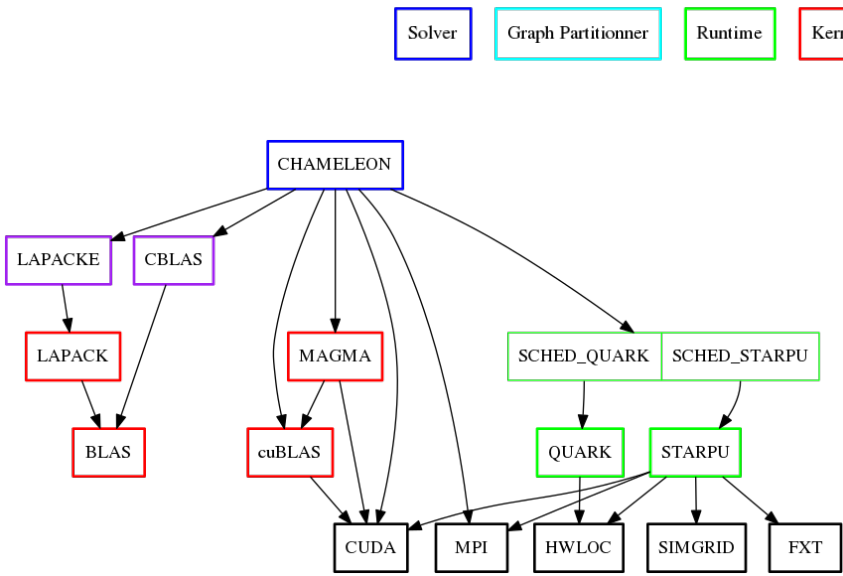


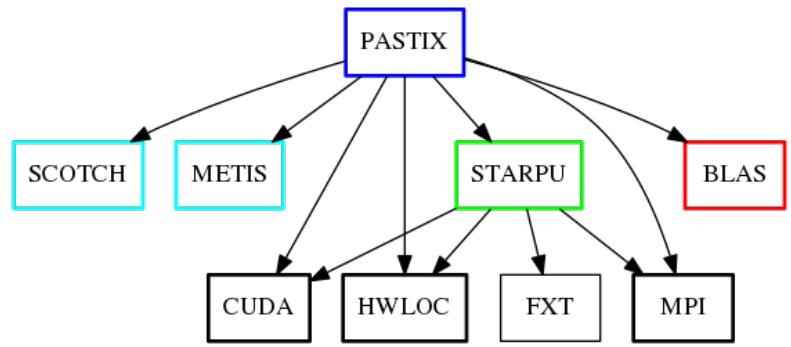
→

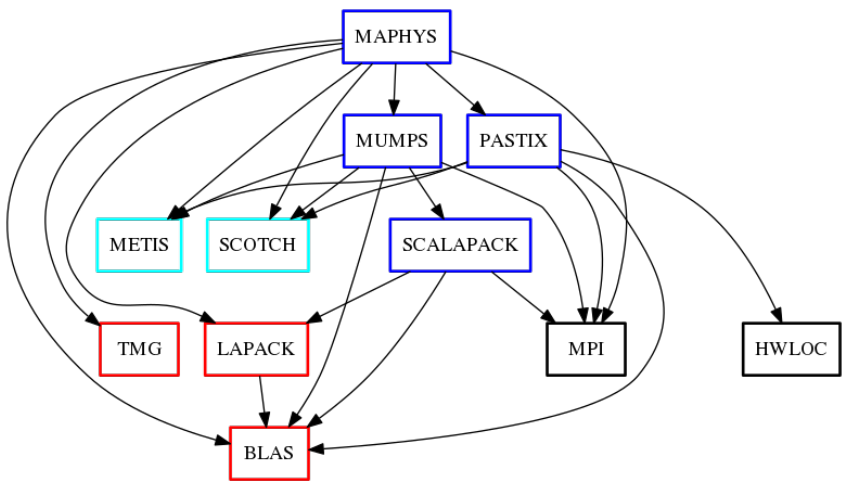
Heterogeneous platforms

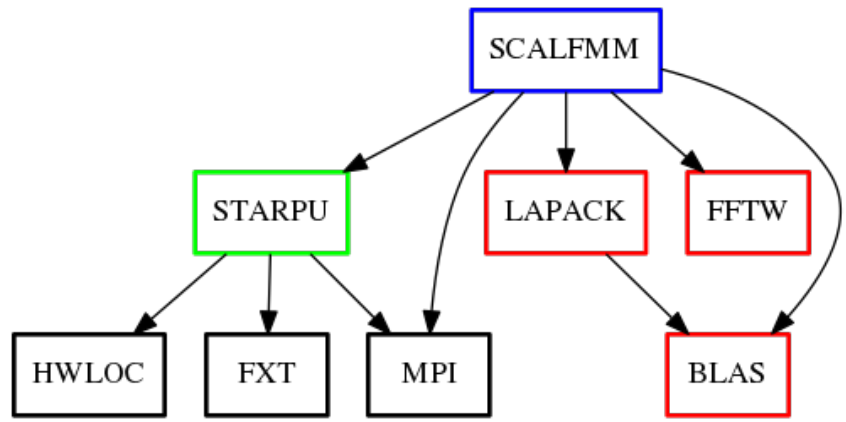



How to deploy complex HPC software stacks?









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
```

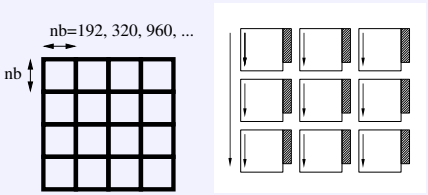
Online tutorial:

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

Case study: dense linear algebra

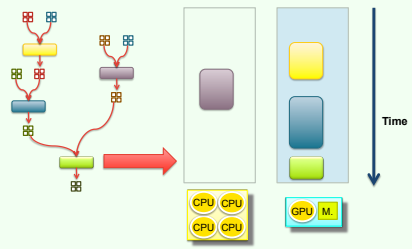
Chameleon = Sequential Task Flow (STF) design of dense linear algebra tiles algorithms on top of runtime systems

Tile matrix layout



Runtime systems

- QUARK
- StarPU



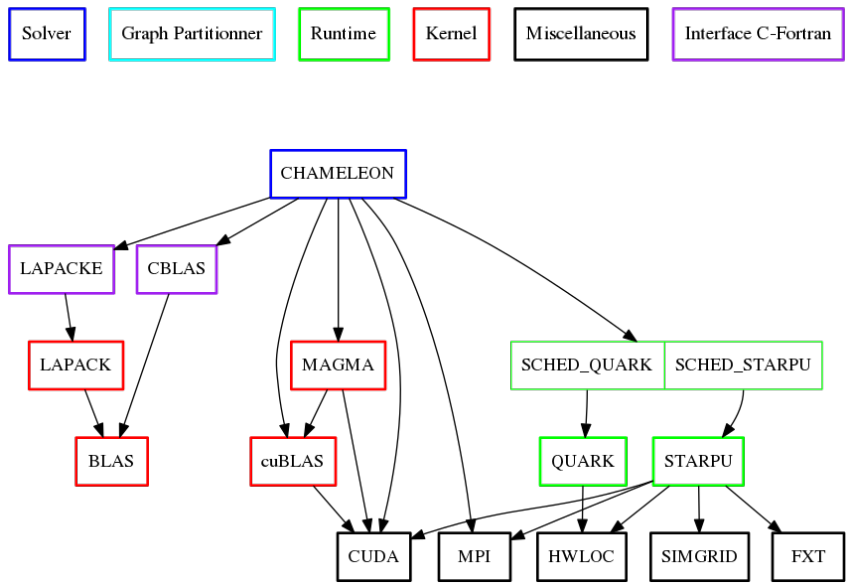
STF algorithms

```
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:

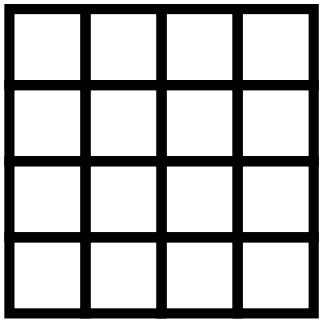
http://morse.gforge.inria.fr/tuto_chameleon/

Outline

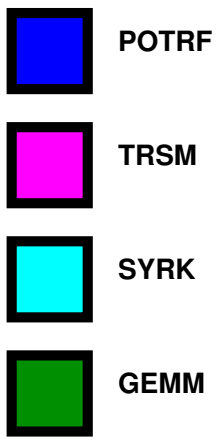
Outline

STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels



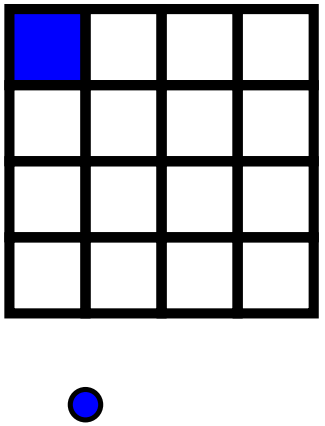
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

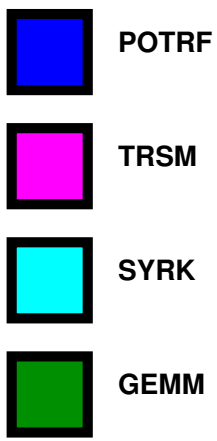


STF Cholesky Algorithm on homogeneous node

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__();
```

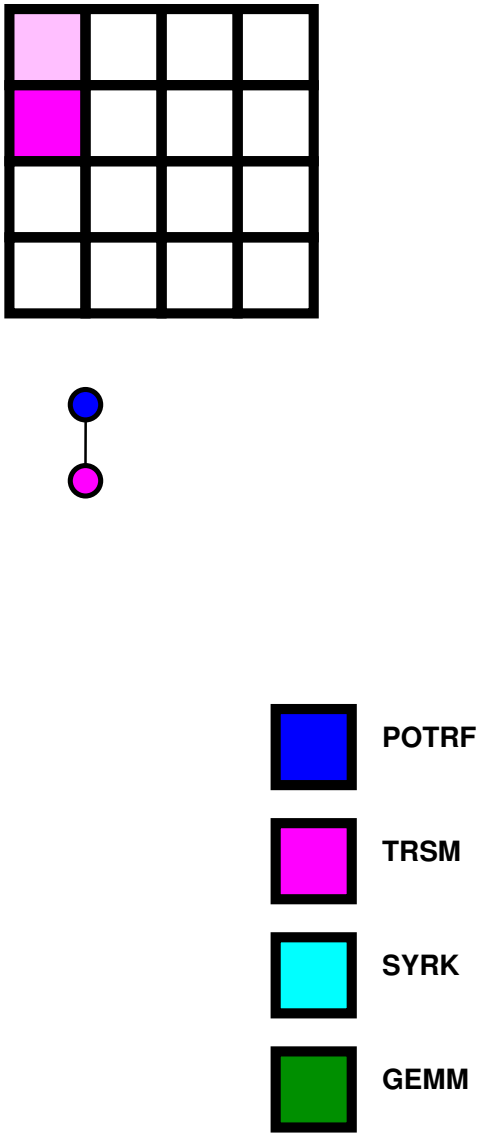




STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

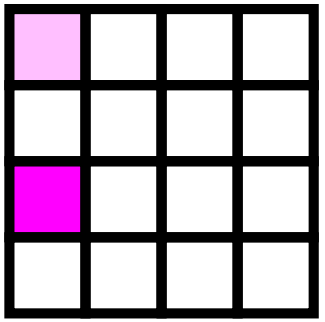
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

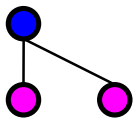


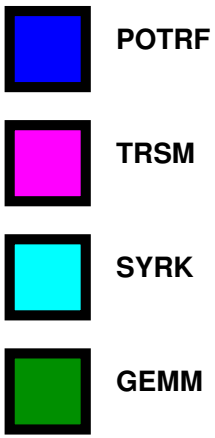
STF Cholesky Algorithm on homogeneous node

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__();
```



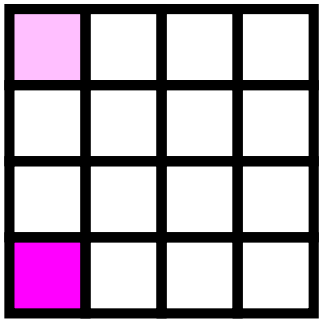


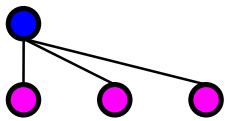


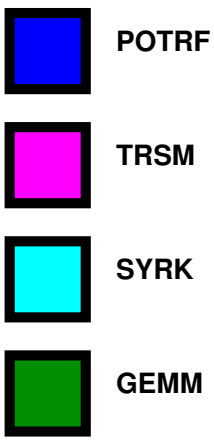
STF Cholesky Algorithm on homogeneous node

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__();
```



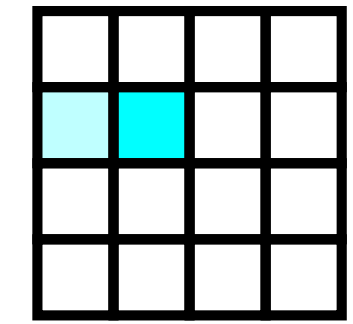


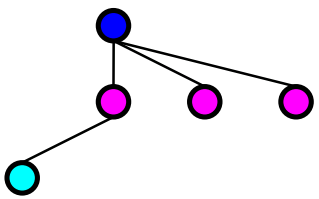


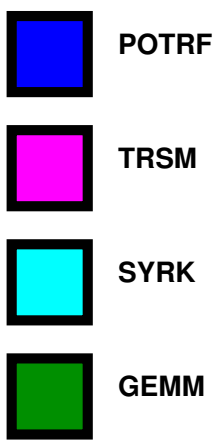
STF Cholesky Algorithm on homogeneous node

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__();
```



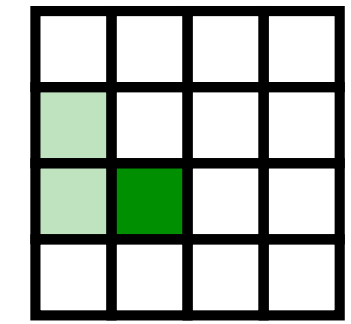


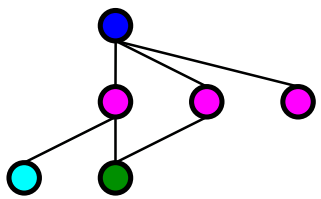


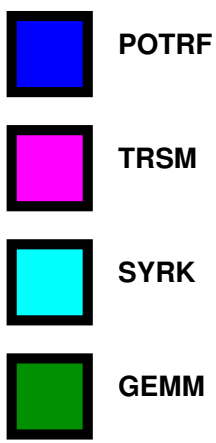
STF Cholesky Algorithm on homogeneous node

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__();
```



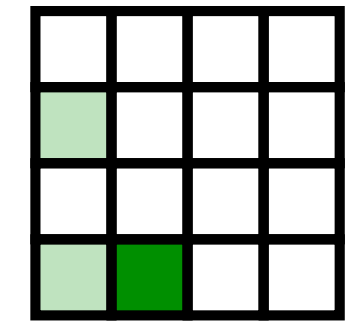


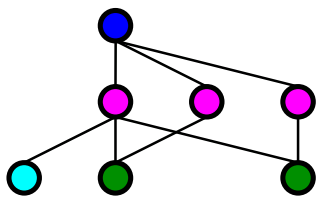


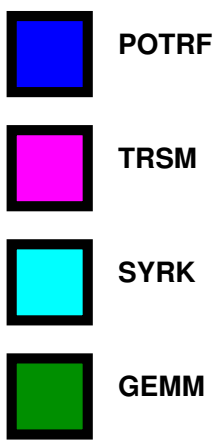
STF Cholesky Algorithm on homogeneous node

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__();
```



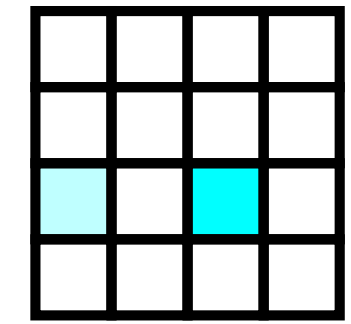


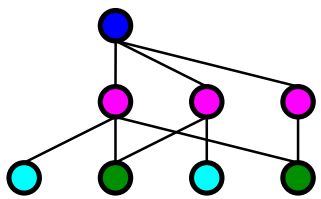


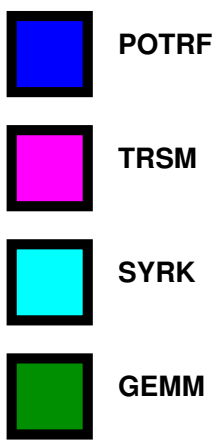
STF Cholesky Algorithm on homogeneous node

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__();
```



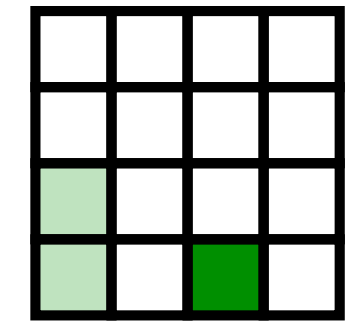


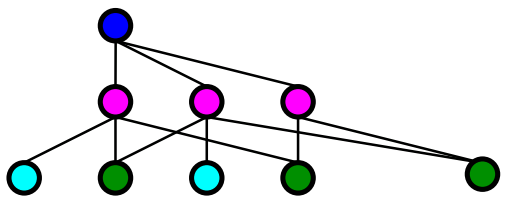


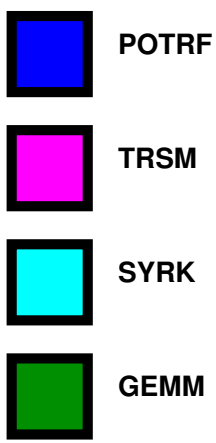
STF Cholesky Algorithm on homogeneous node

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__();
```



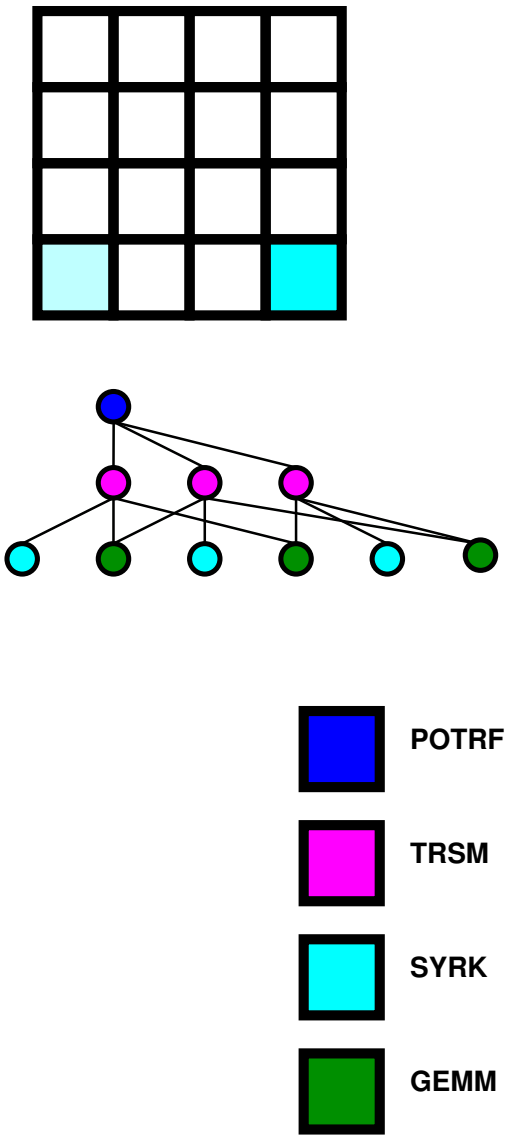




STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

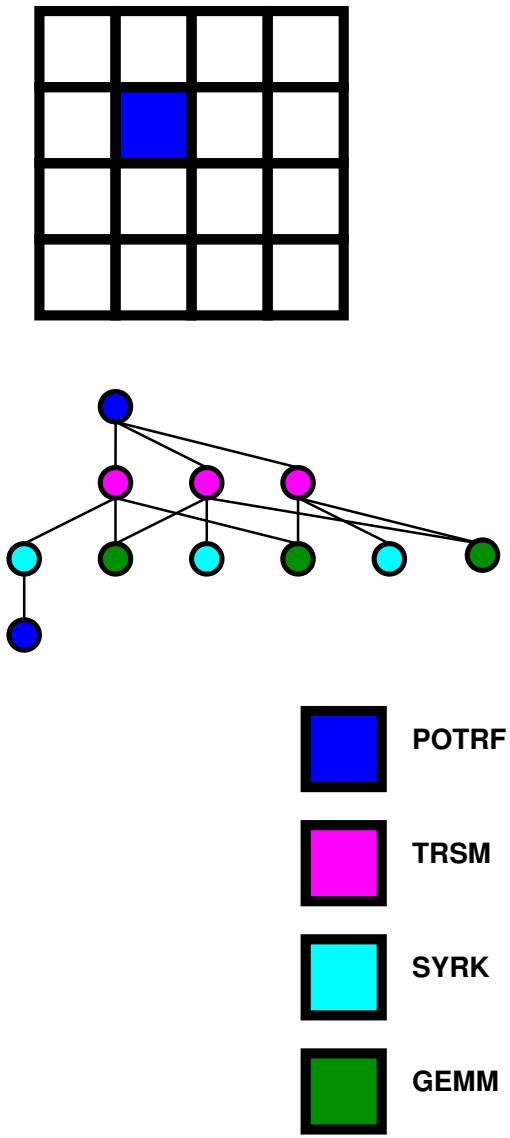
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```



STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

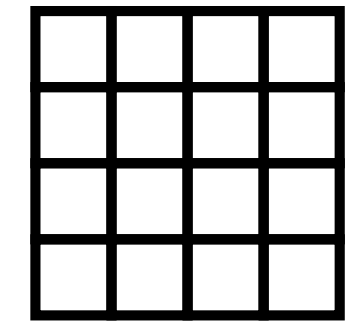
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

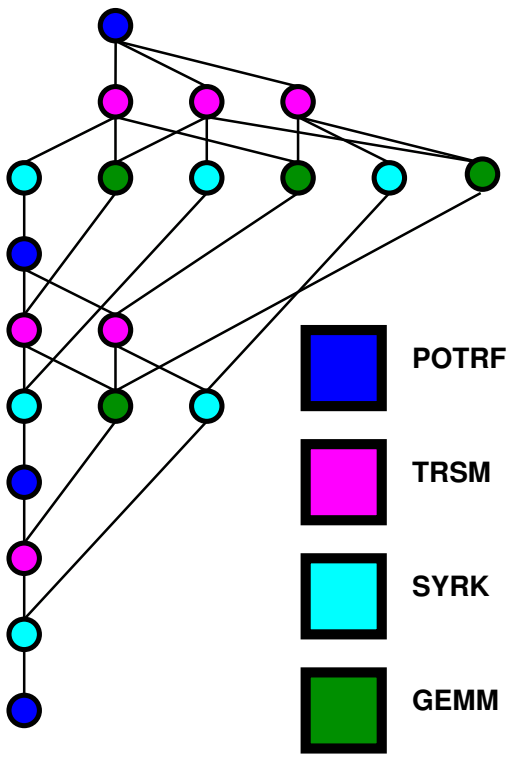


STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

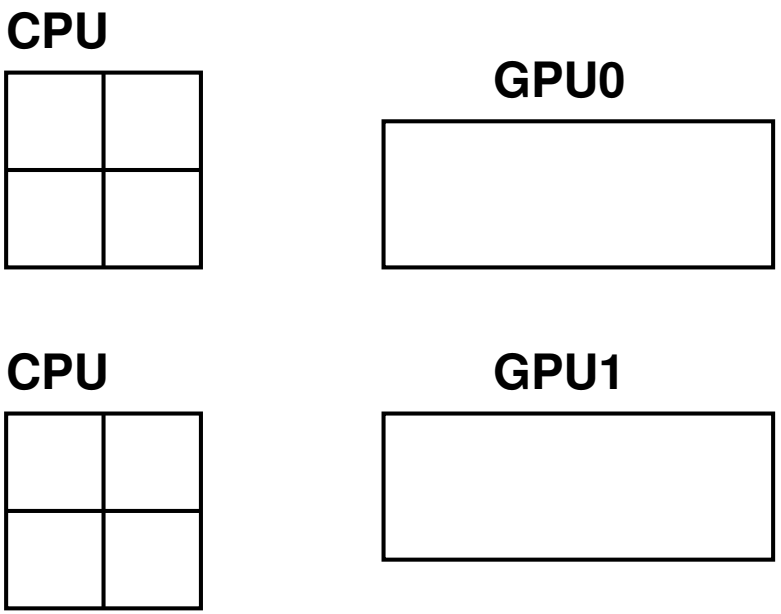
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

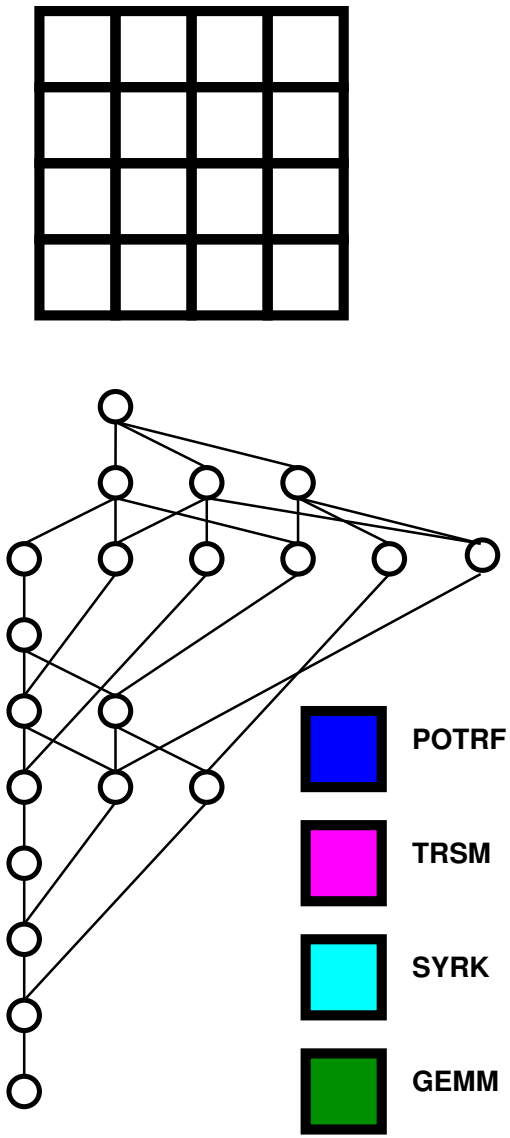




On an heterogeneous node

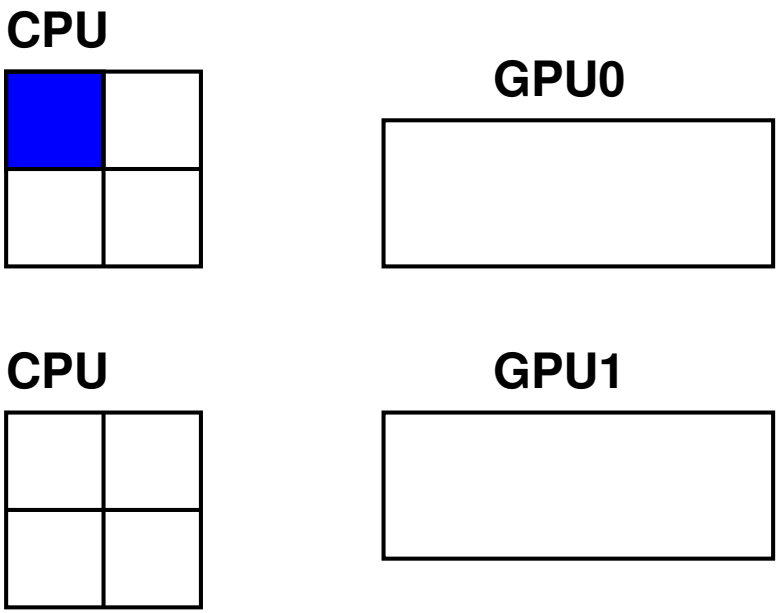
work on tiles $\rightarrow$ CPU + GPU kernels

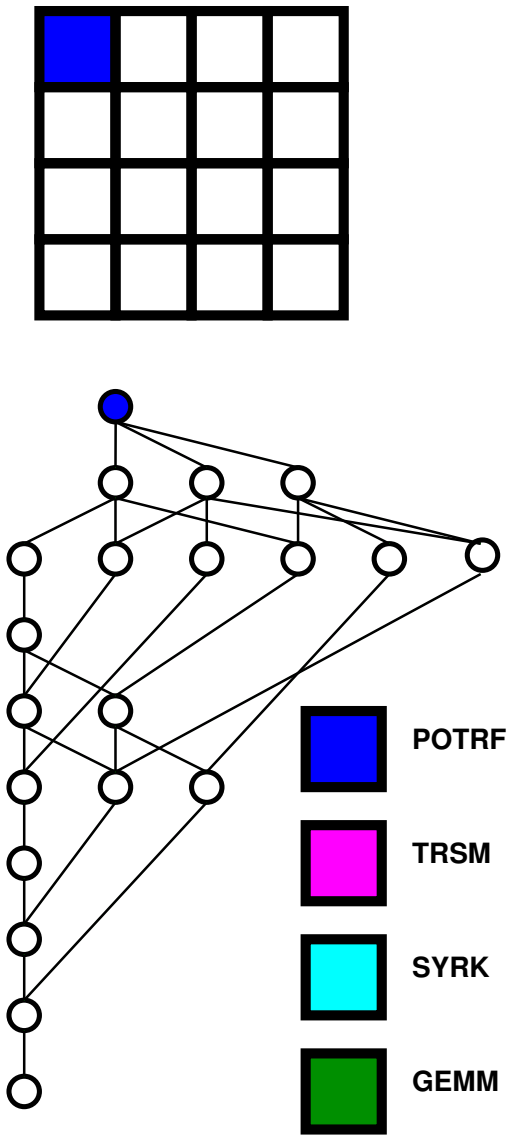




On an heterogeneous node

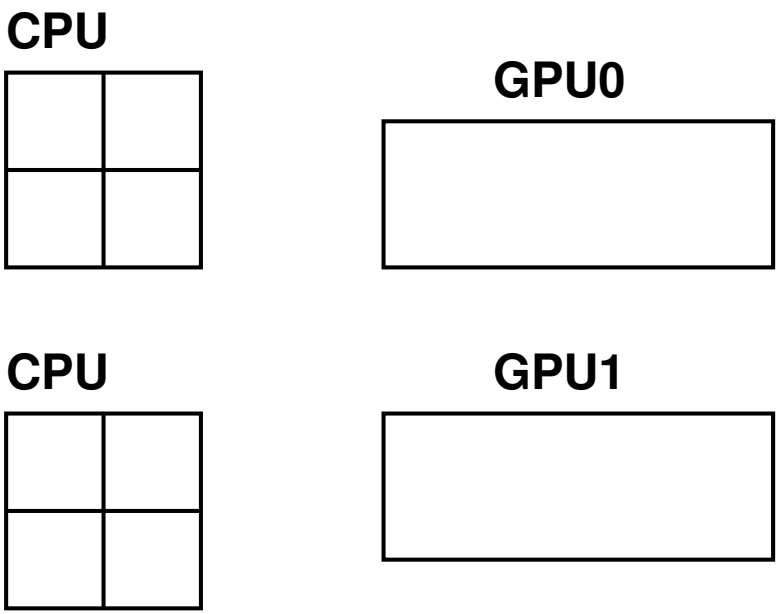
work on tiles $\rightarrow$ CPU + GPU kernels

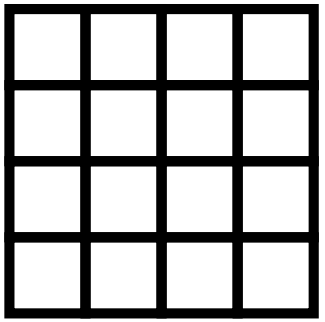


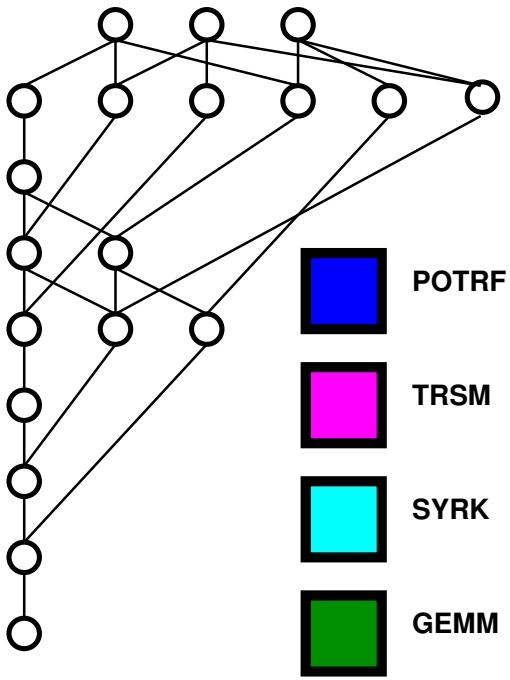


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

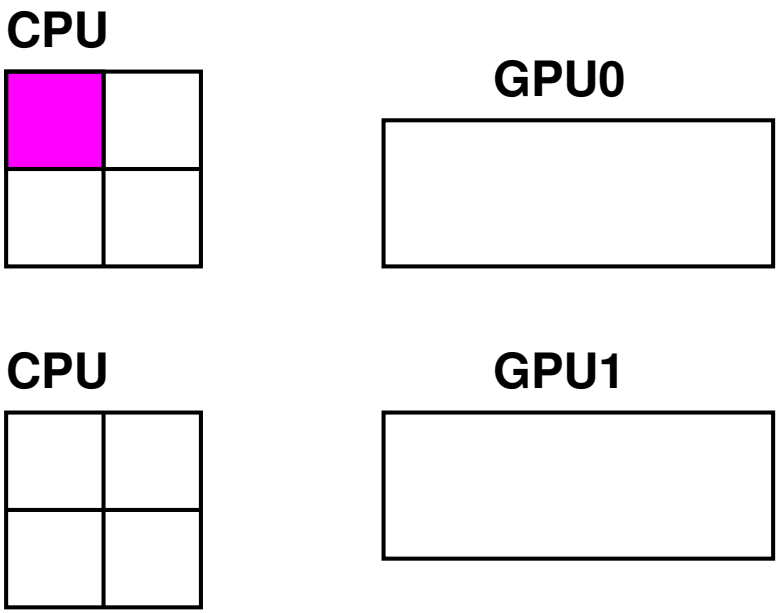


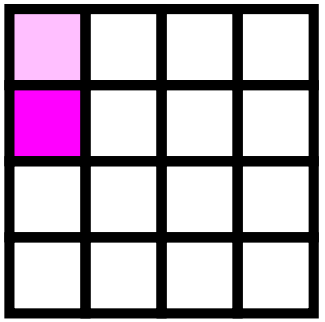


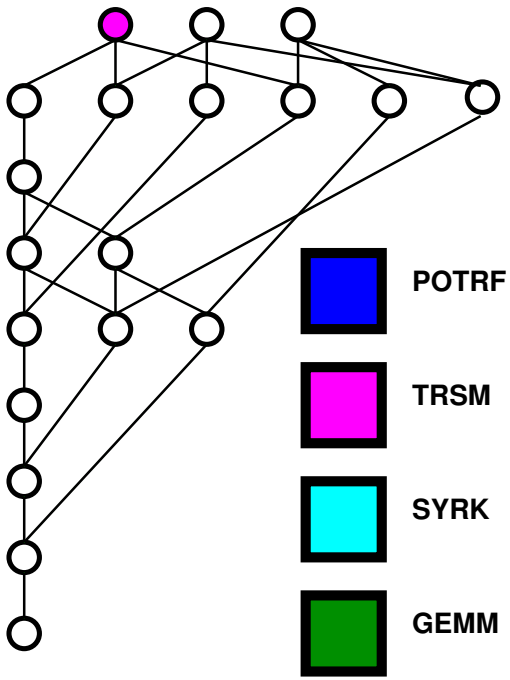


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

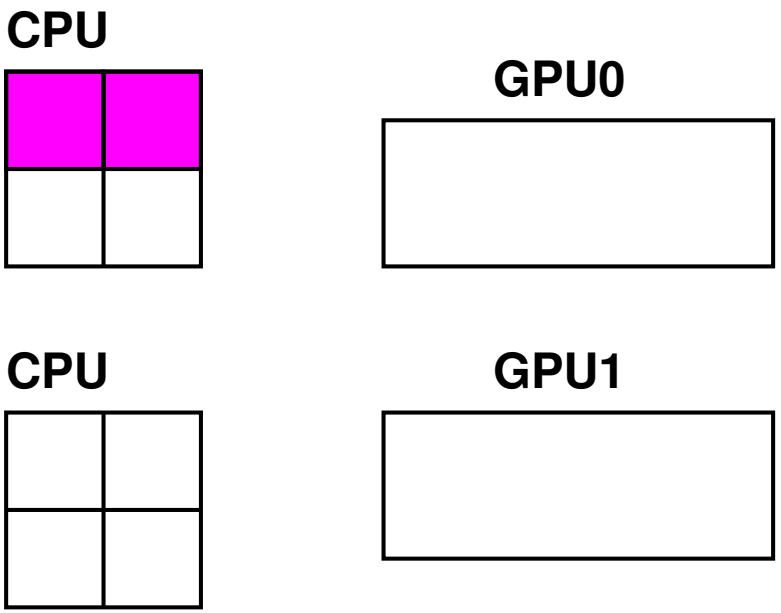


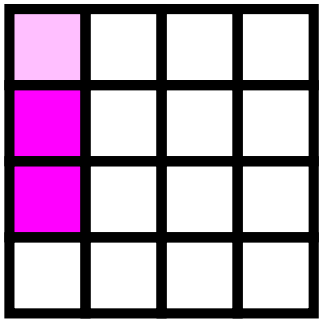


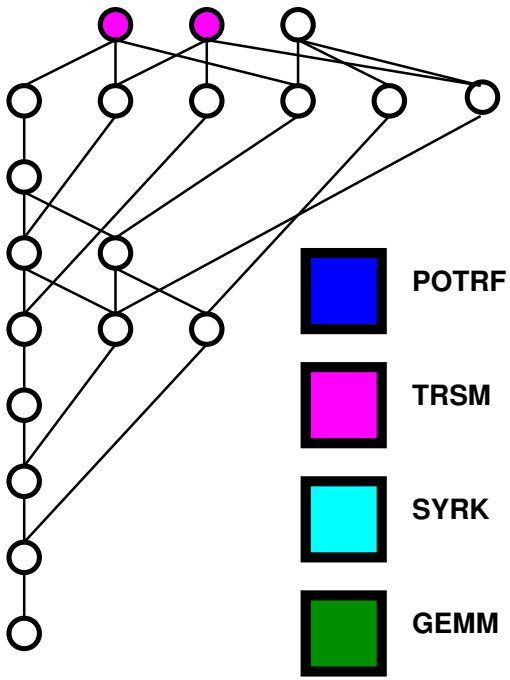


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

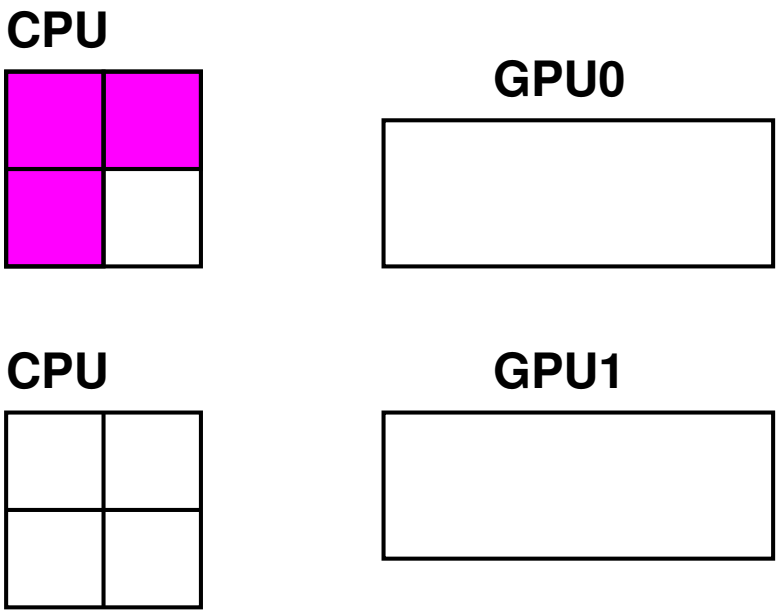


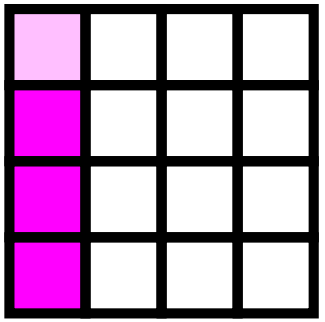


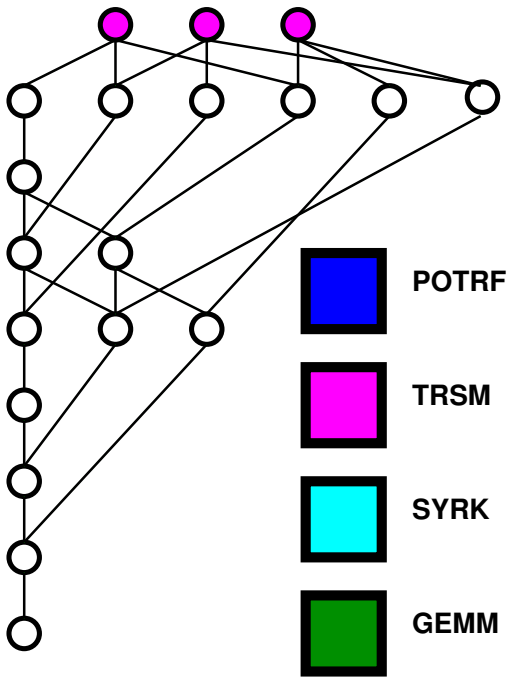


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

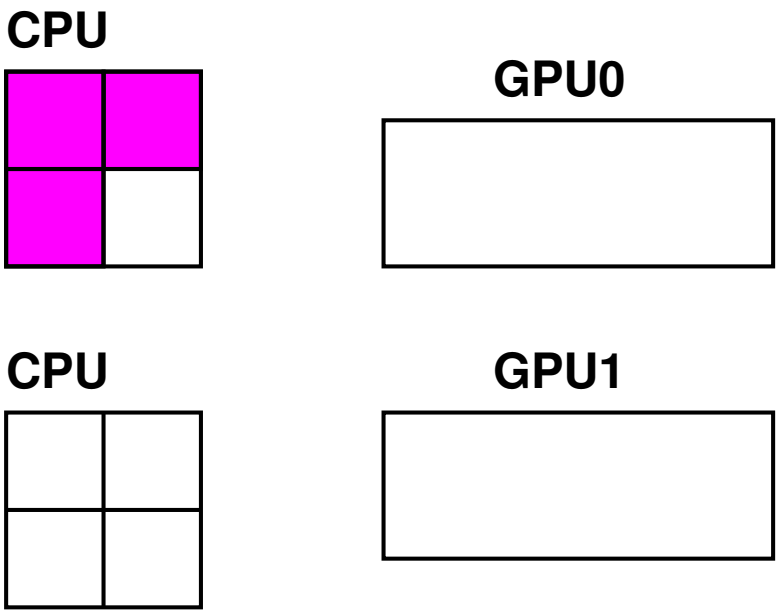


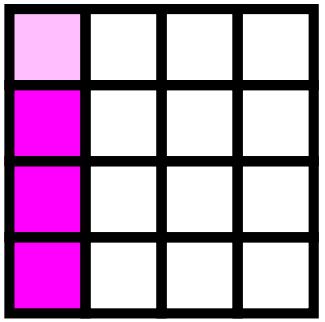


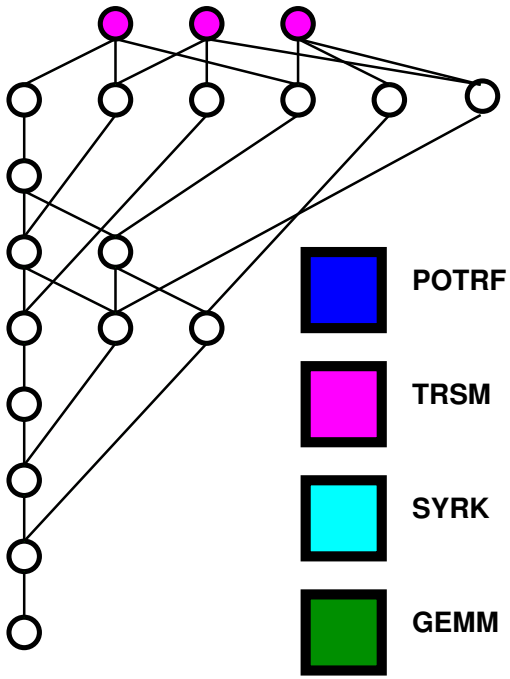


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels



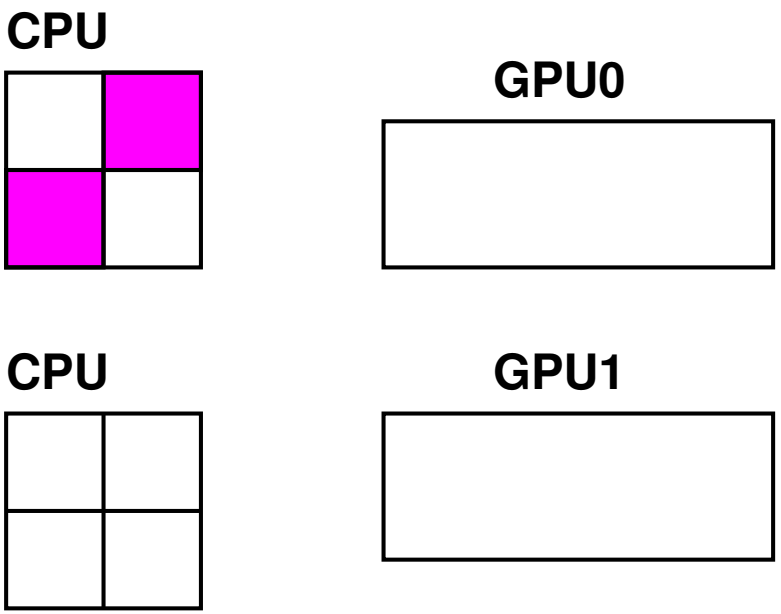


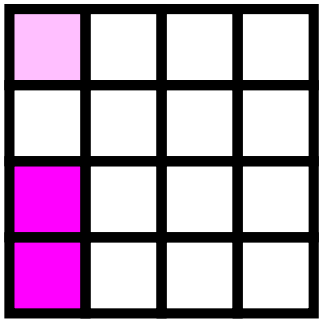


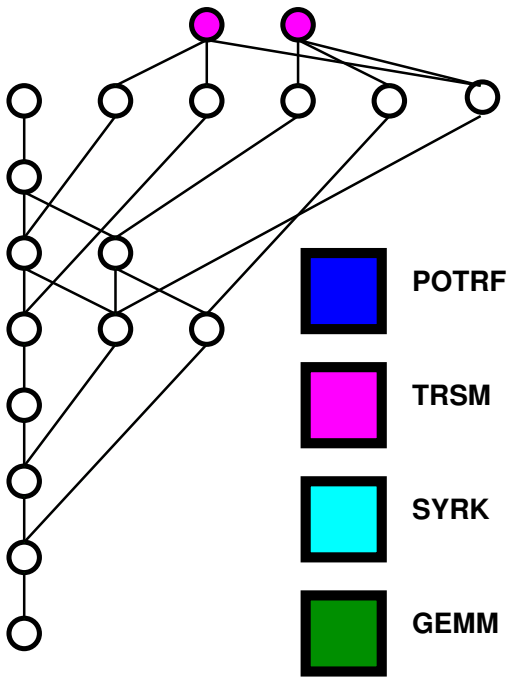
- Handles dependencies

On an heterogeneous node

work on tiles → CPU + GPU kernels



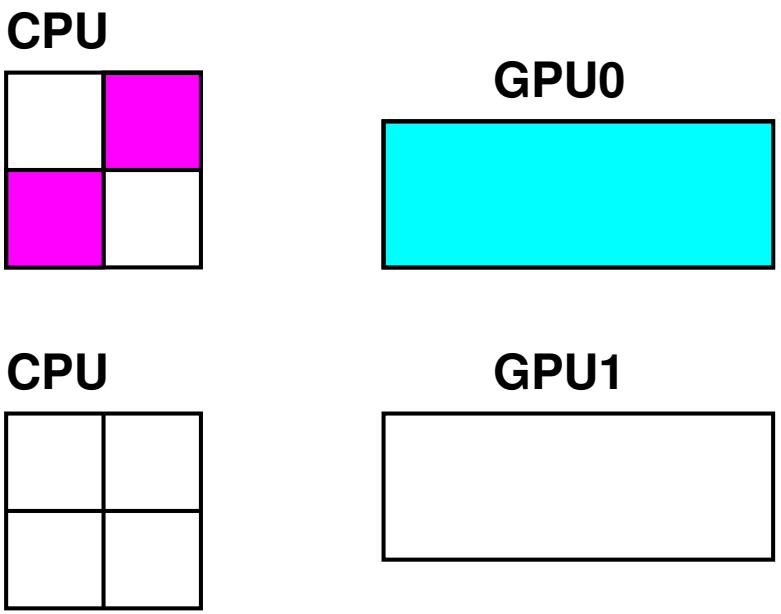


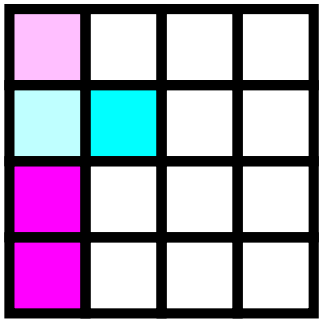


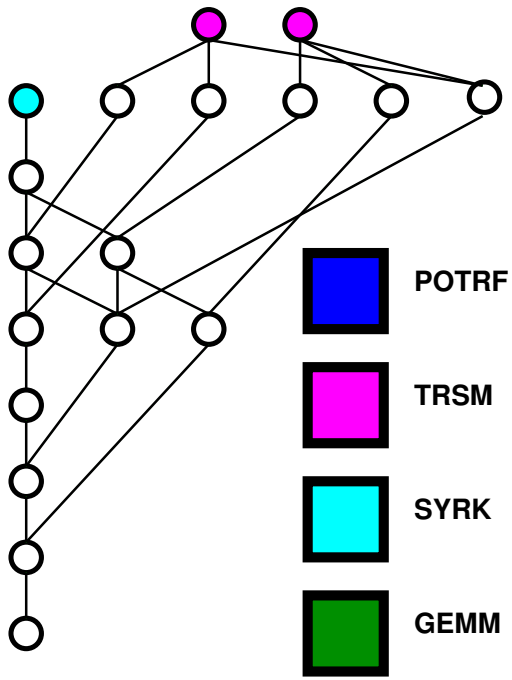
- Handles dependencies

On an heterogeneous node

work on tiles → CPU + GPU kernels



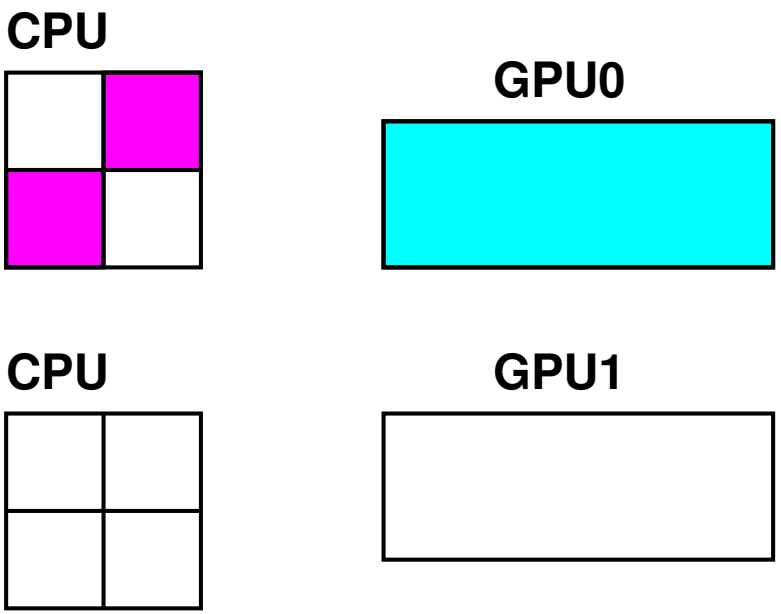


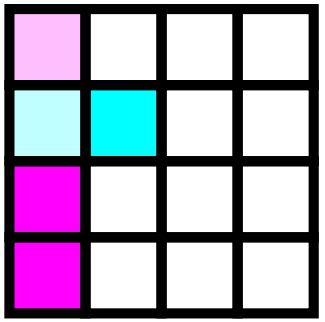


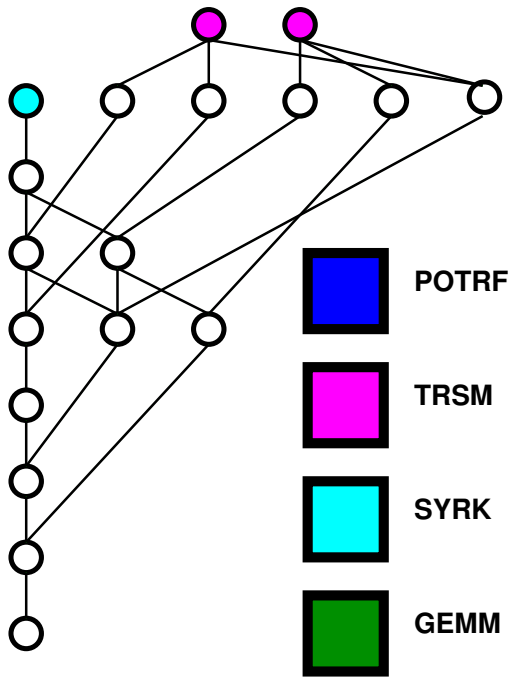
- Handles dependencies

On an heterogeneous node

work on tiles → CPU + GPU kernels



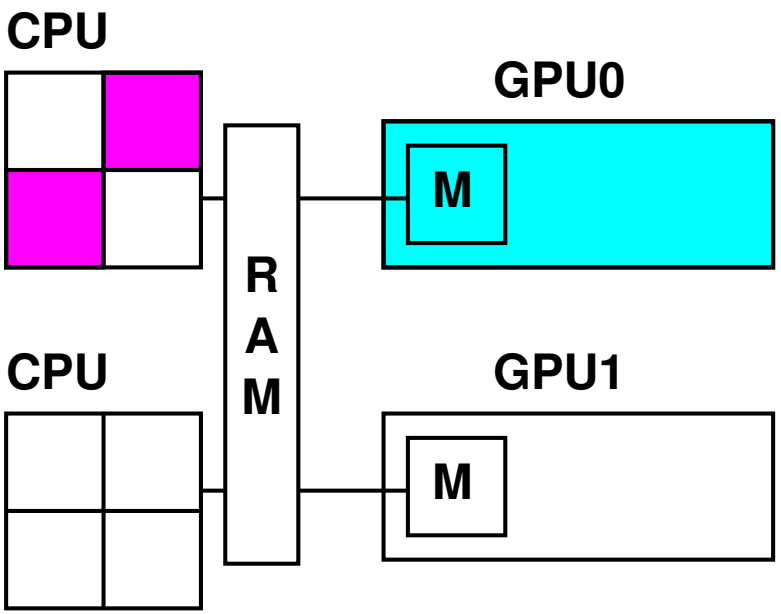


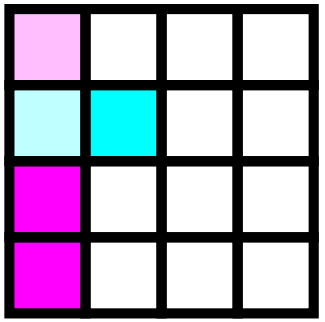


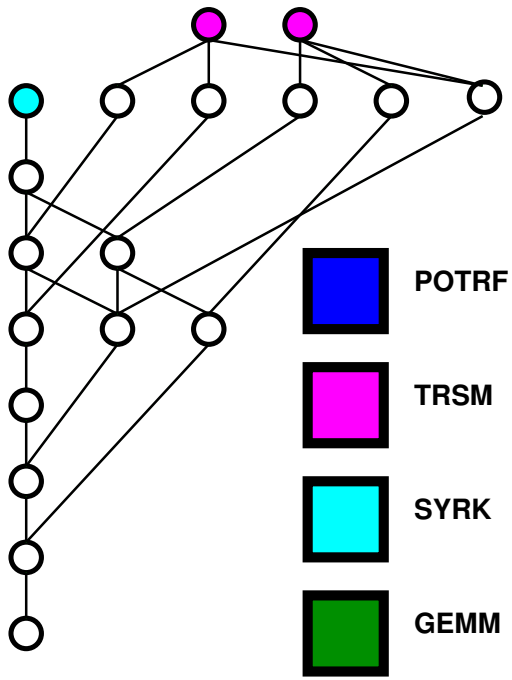
- Handles dependencies
- Handles scheduling

On an heterogeneous node

work on tiles → CPU + GPU kernels



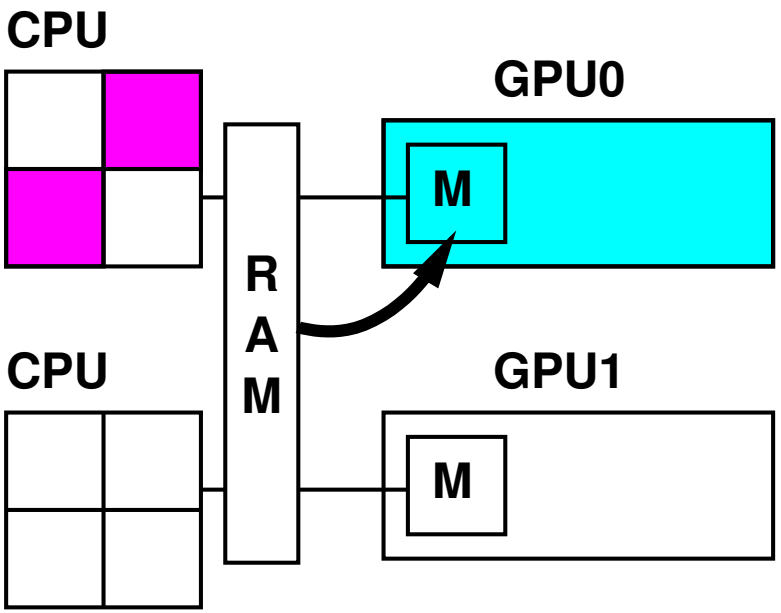


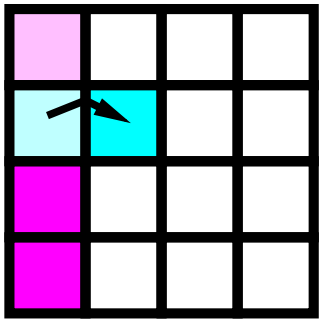


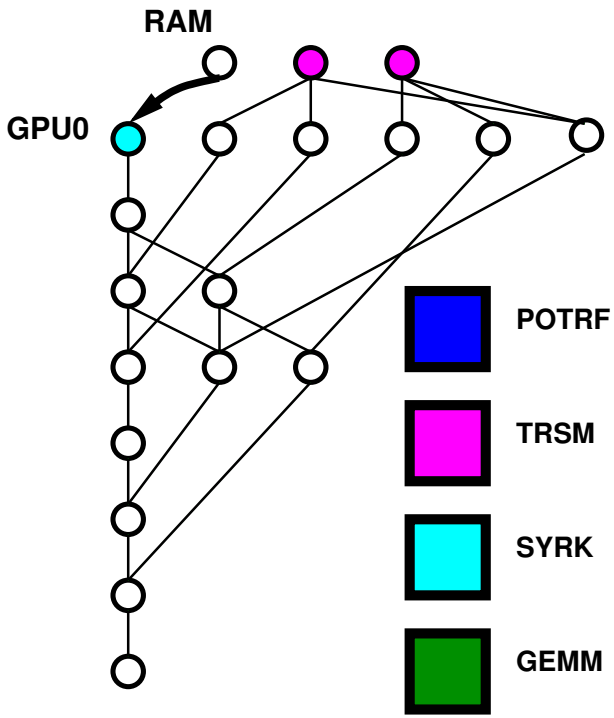
- Handles dependencies
- Handles scheduling

On an heterogeneous node

work on tiles → CPU + GPU kernels





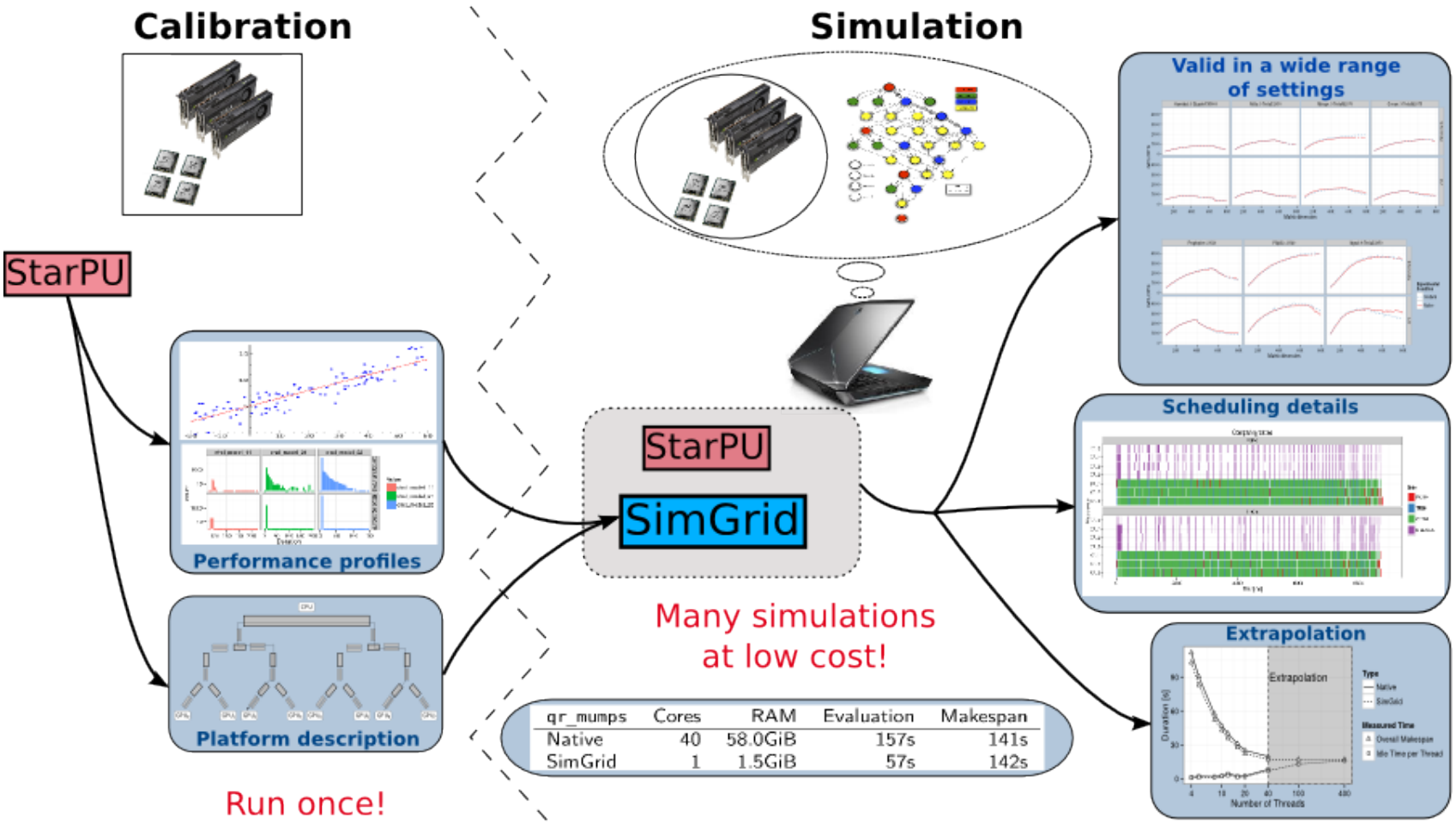


- Handles dependencies
- Handles scheduling
- Handles data consistency

Outline

StarPU-SimGrid in a nutshell

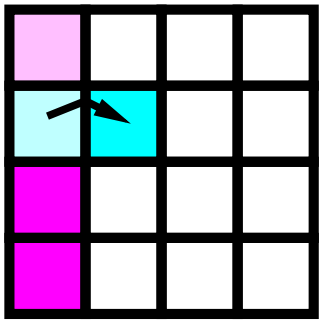
PhD L. Stanisic



Outline

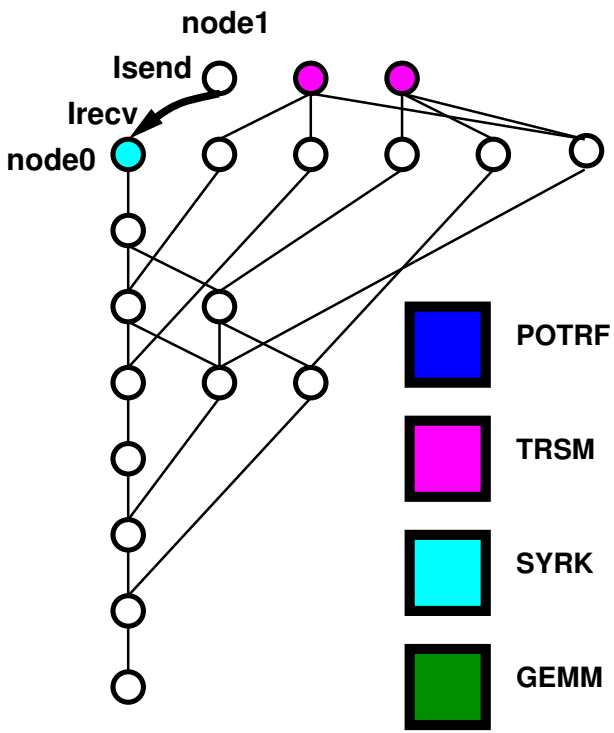
Address scalability

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



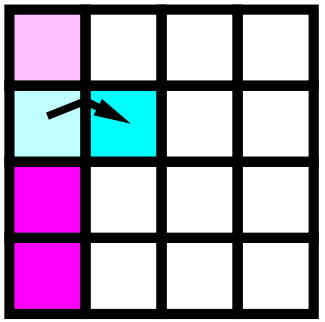
Questions

Existing methods



Address scalability

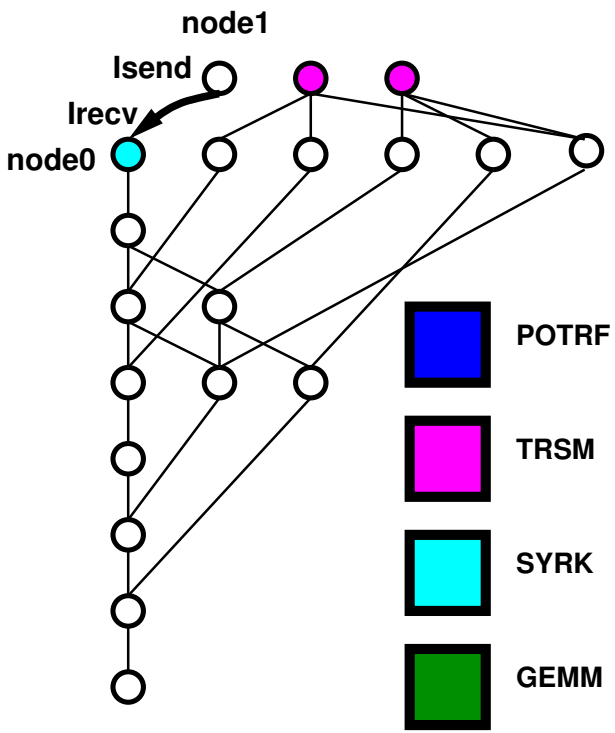
PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?



Questions

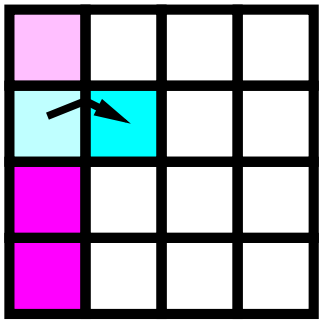
- How to establish the mapping?

Existing methods



Address scalability

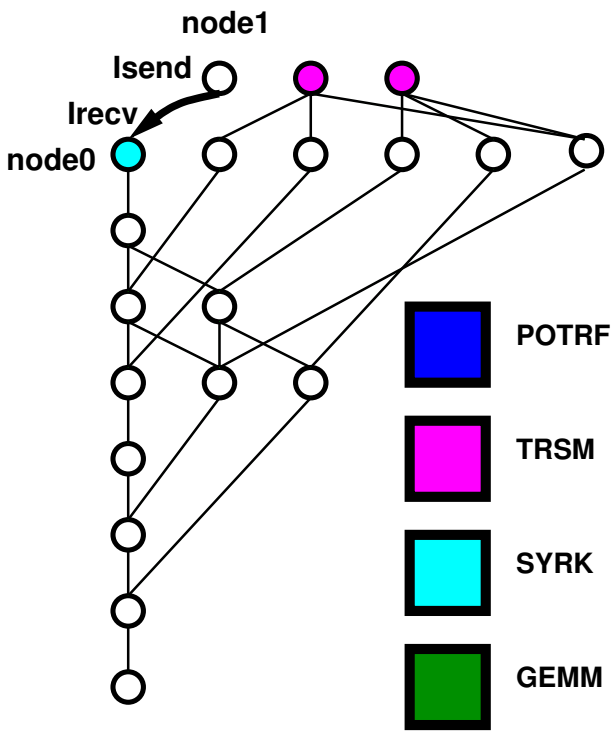
PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?



Questions

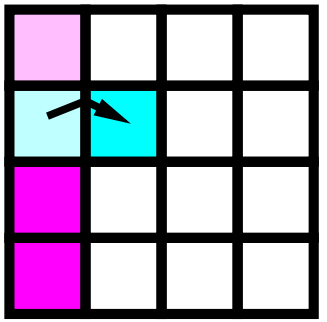
- How to establish the mapping?
- How to manage communications?

Existing methods



Address scalability

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

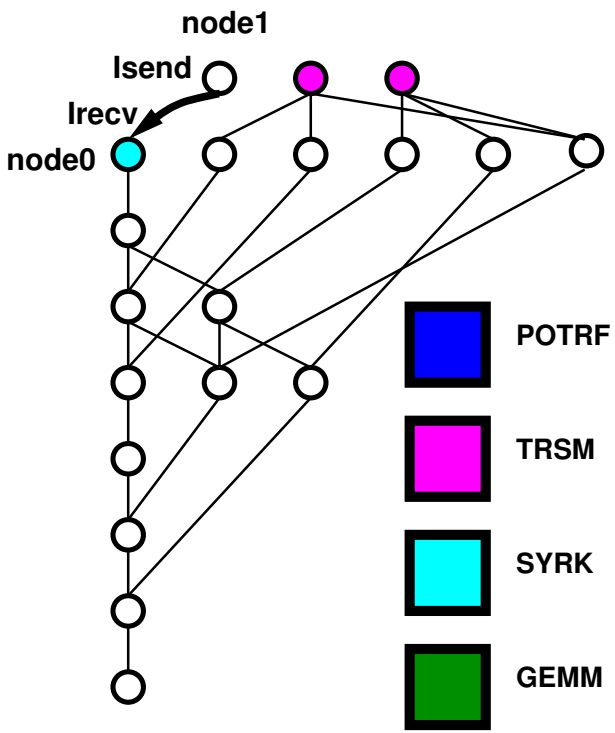


Questions

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

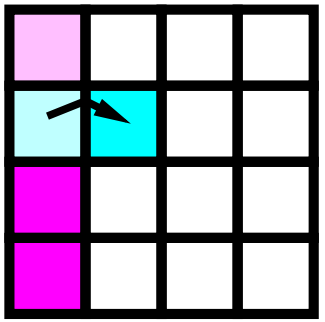
Existing methods

- Explicit MPI communications tasks



Address scalability

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

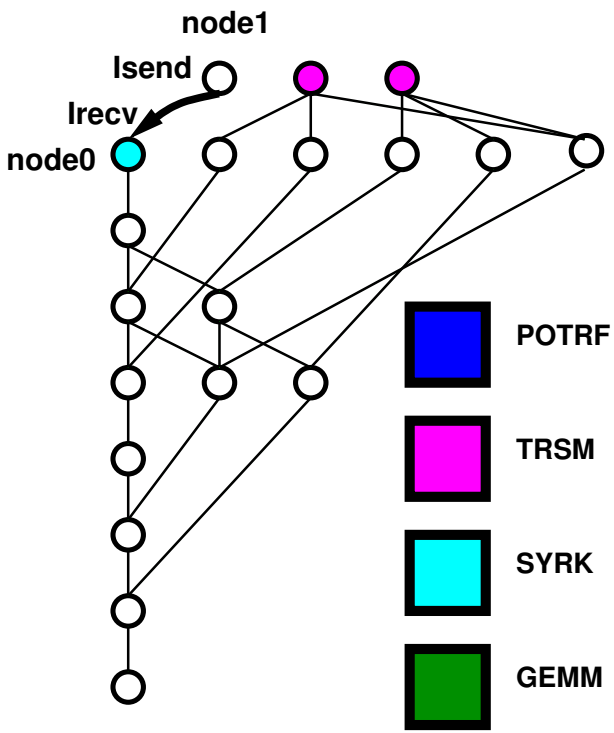


Questions

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

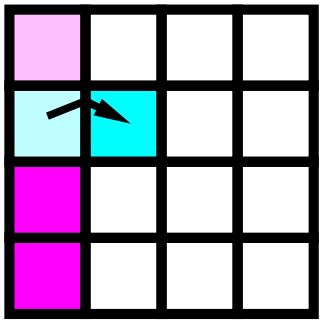
Existing methods

- Explicit MPI communications tasks
- PTG model (PaRSEC)



Address scalability

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

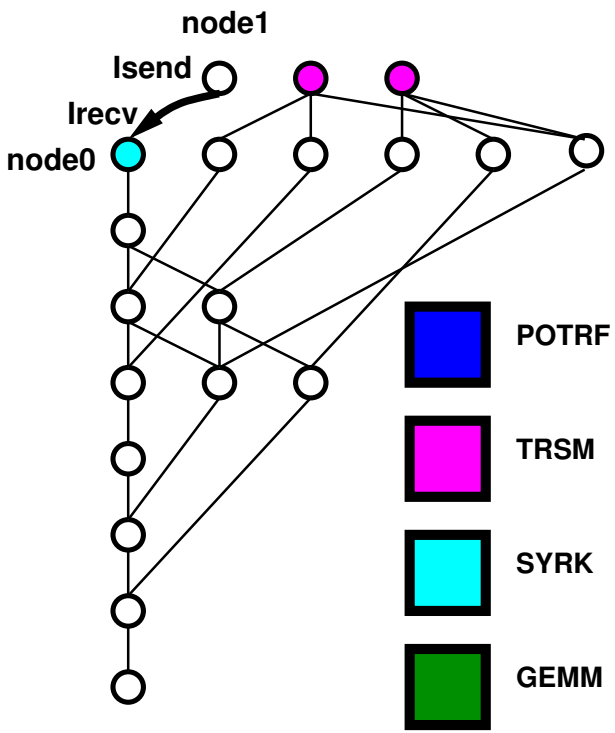


Questions

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

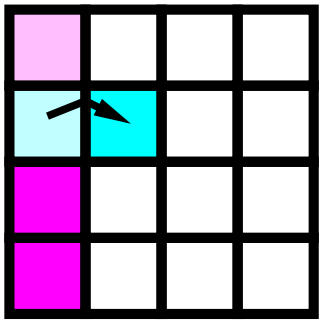
Existing methods

- 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?

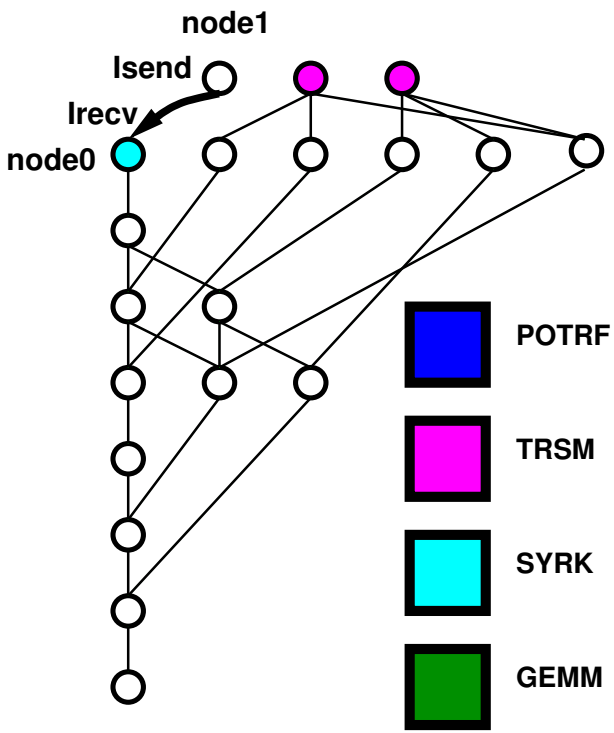


Questions

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

Existing methods

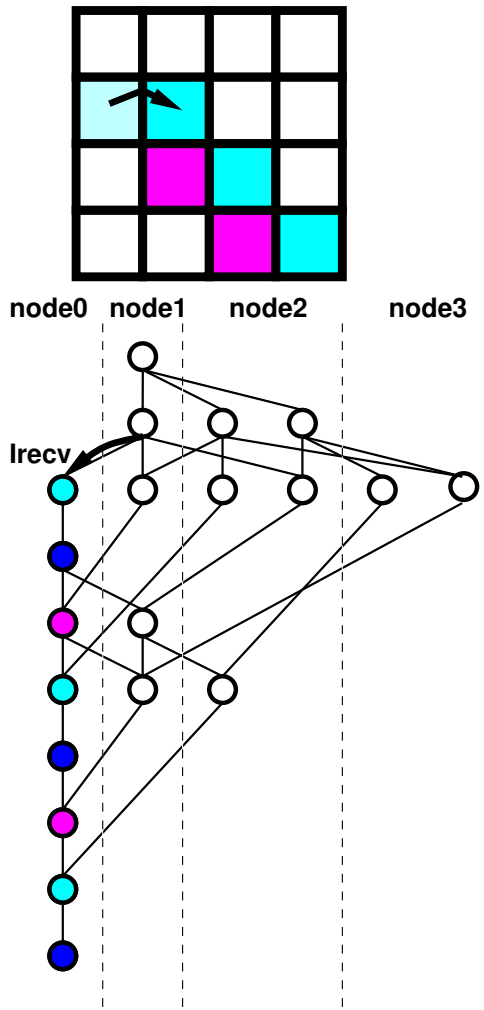
- Explicit MPI communications tasks
- PTG model (PaRSEC)
- STF model - Master/Slave (clusterSS)
- STF model - Replicated unrolling (StarPU)



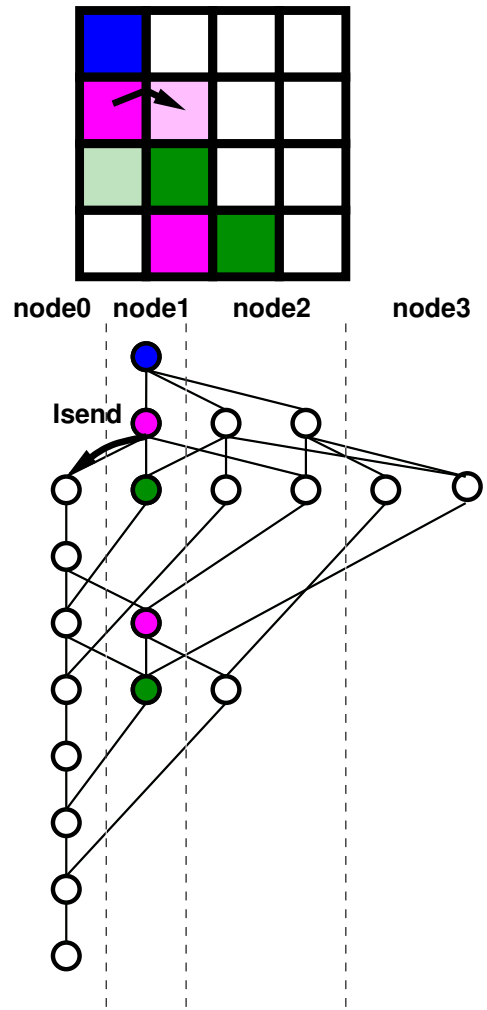
Data transfers between nodes

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



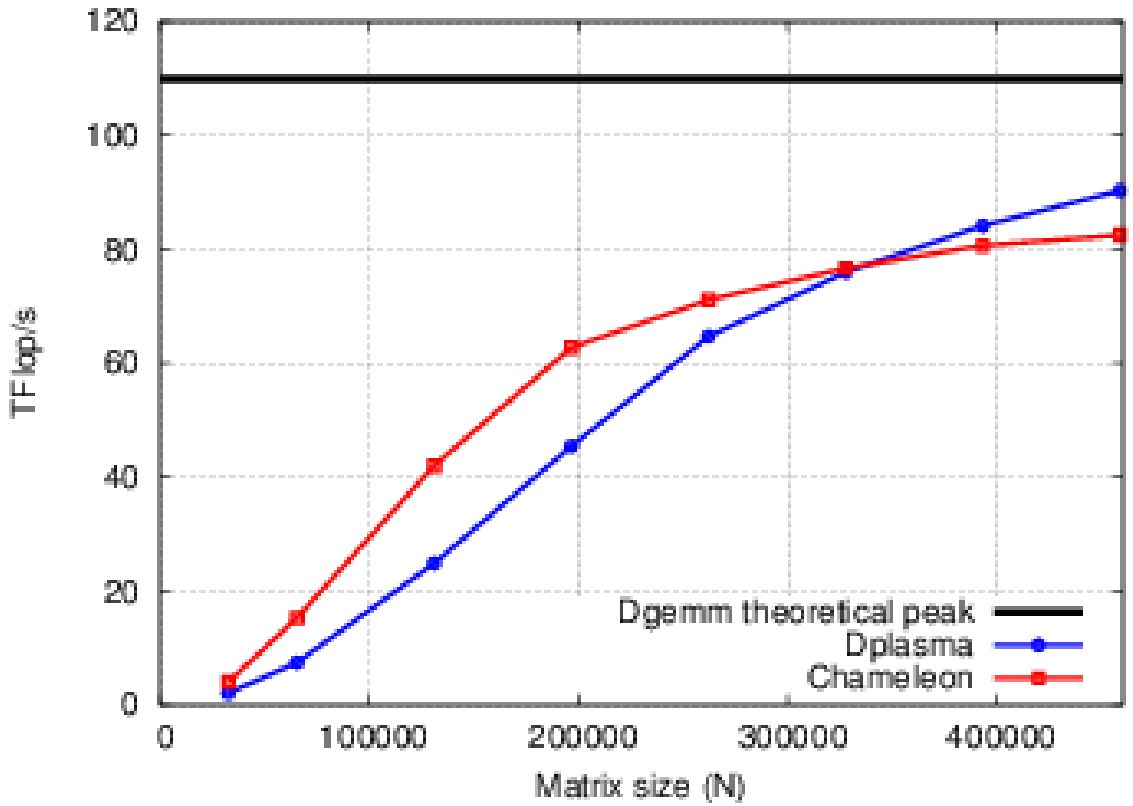
Node 0 execution



Node 1 execution

Performance

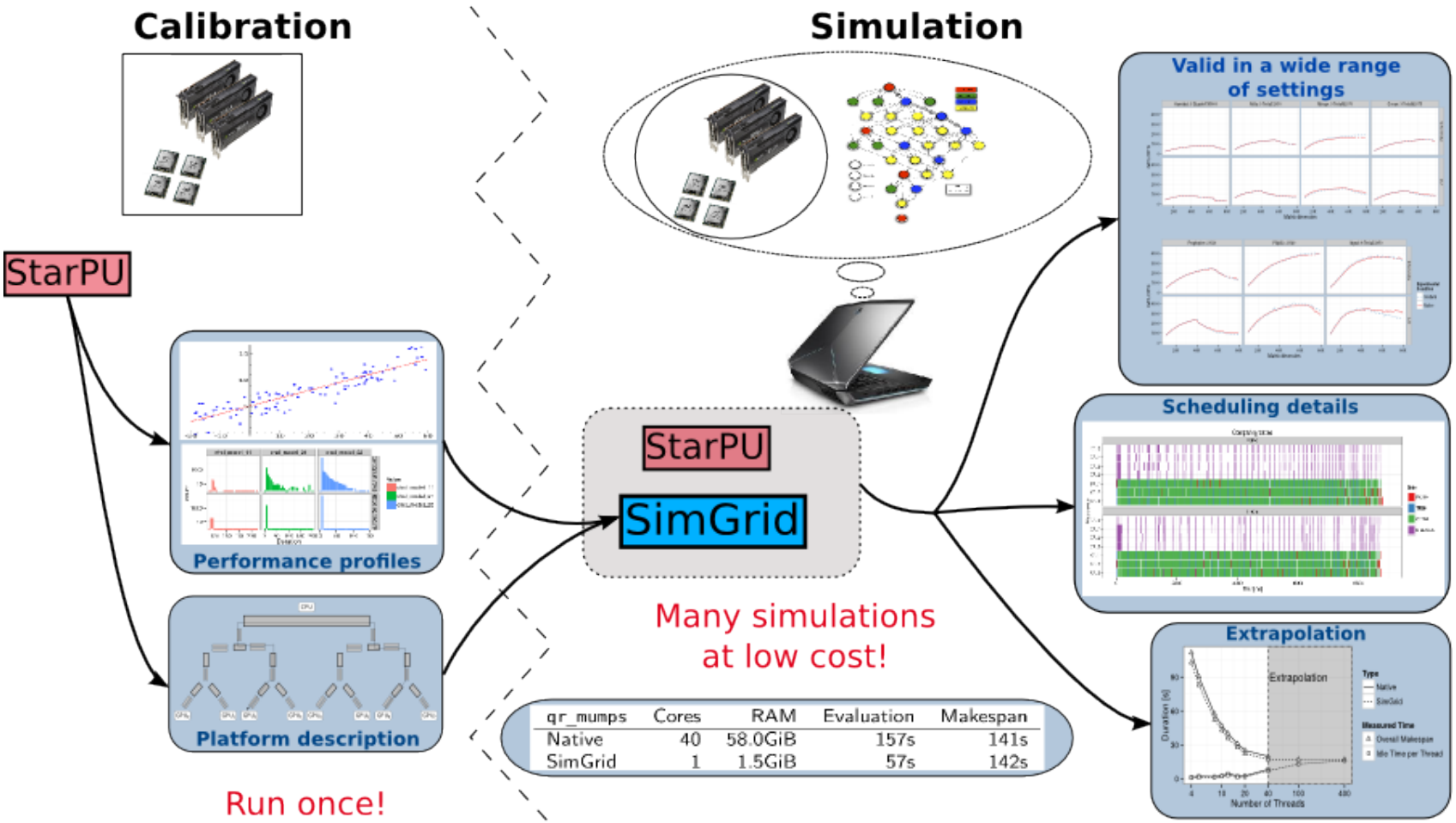
- 144 TERA-100 Hybrid nodes
  - collaboration with CEA-CESTA
- CPU: 2 Quad-core Xeon E5620 (per node)
- GPU: 2 NVIDIA Tesla M2090 (per node)



Outline

StarPU-SimGrid in a nutshell ...

Postdoc L. Stanisic



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

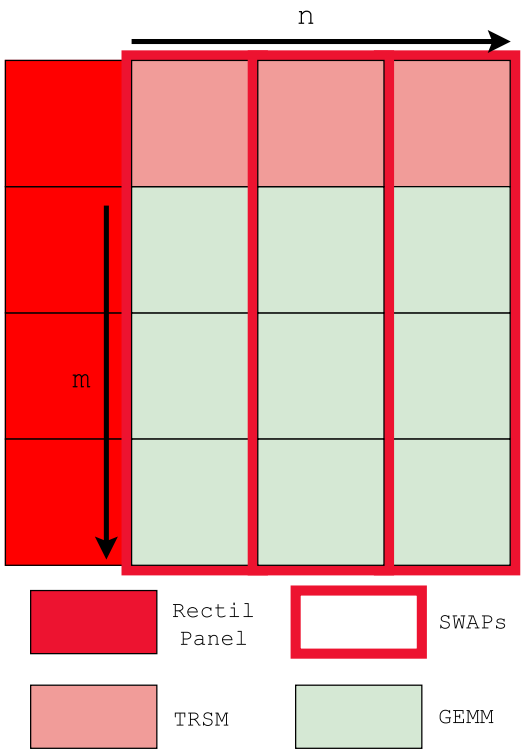
Outline

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
```

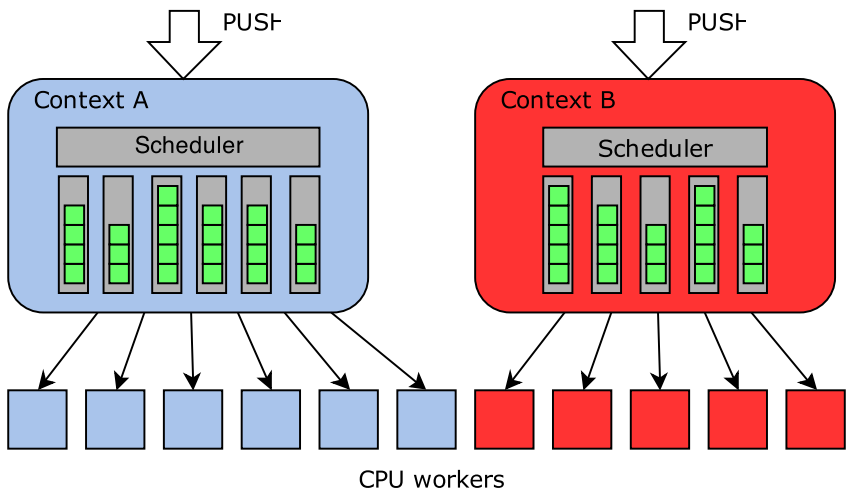
For step k=0 with MT=4



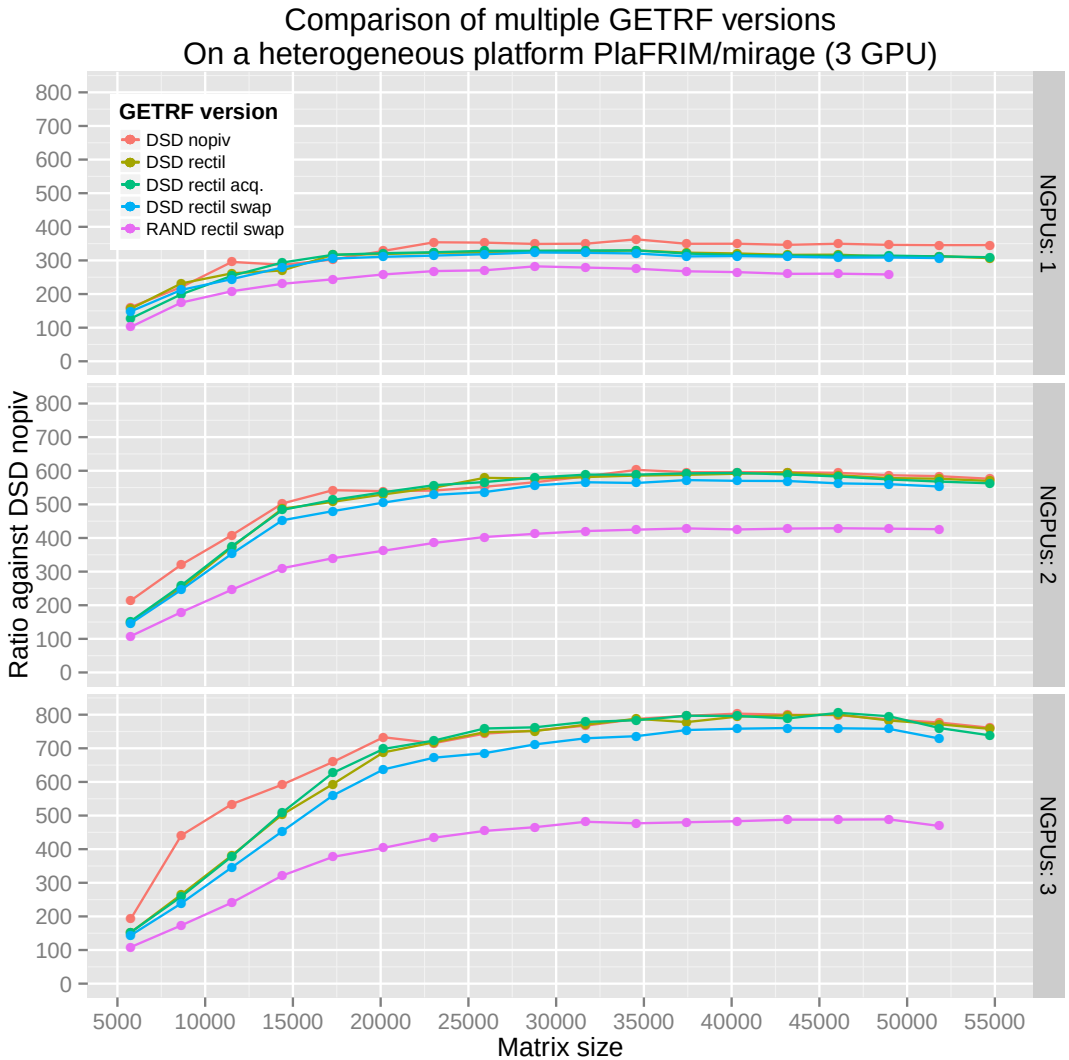
StarPU contexts

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)



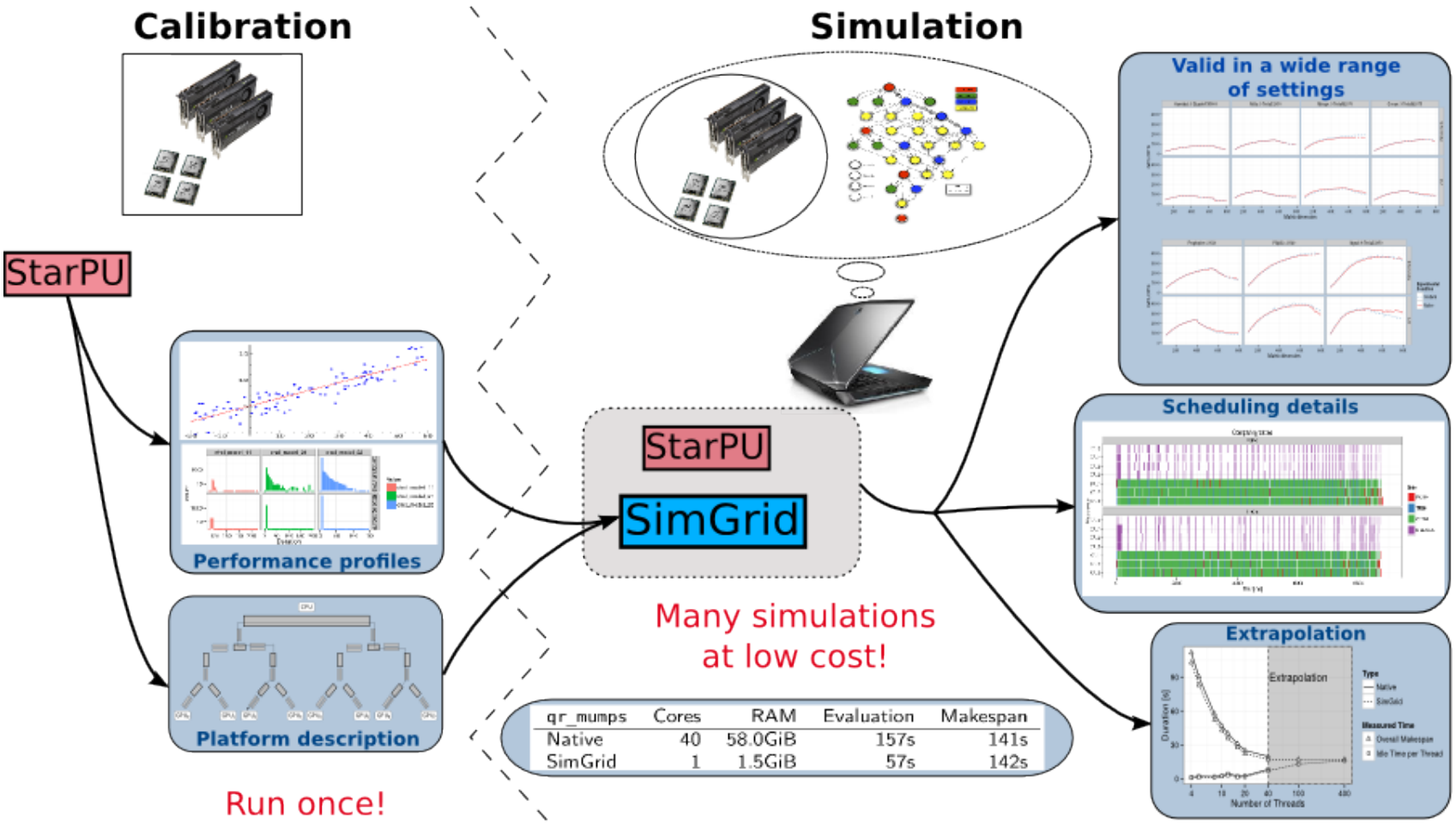
Performance results - Plafrim Mirage node



Outline

StarPU-SimGrid in a nutshell ...

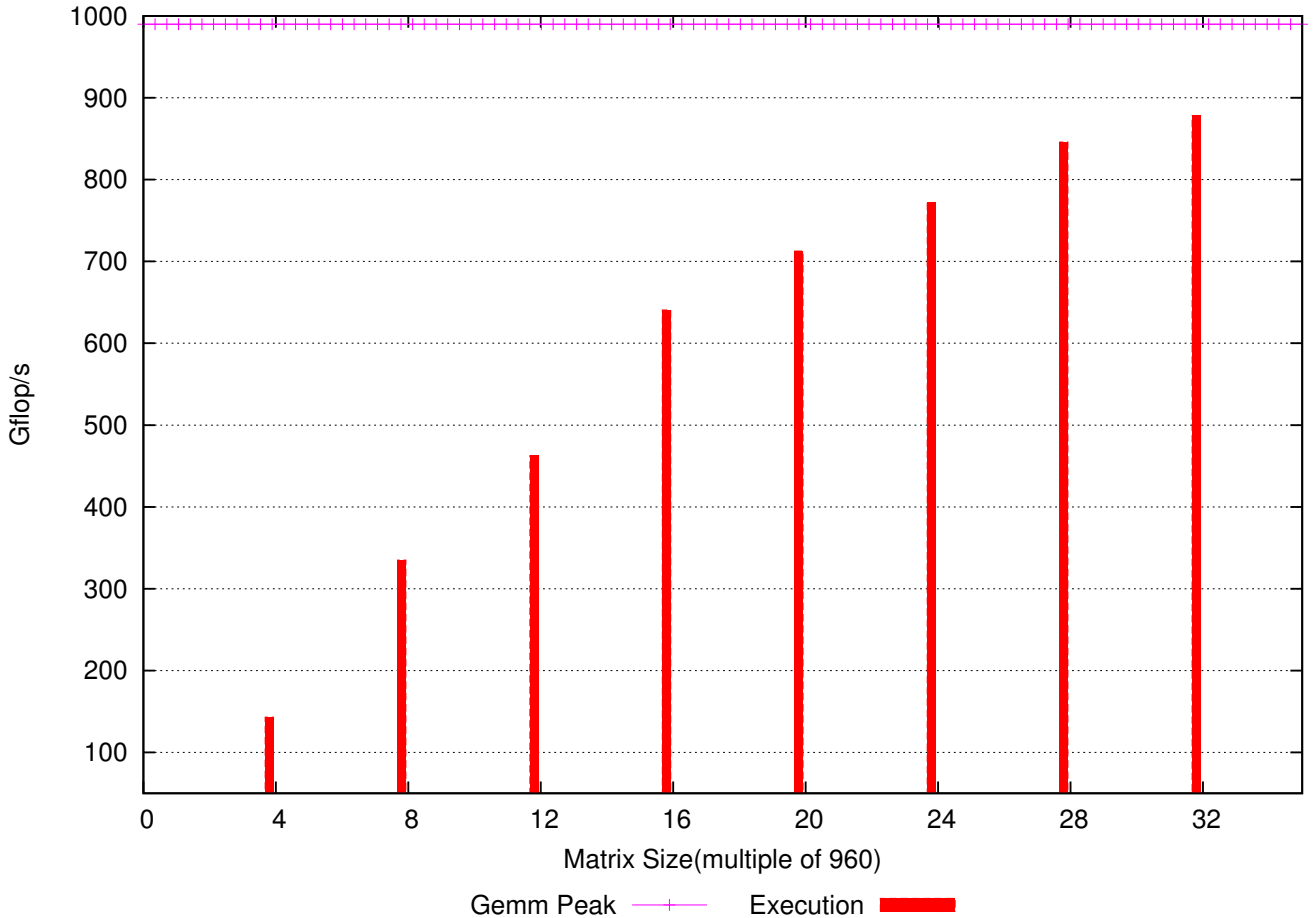
Postdoc L. Stanisic and PhD T. Cojean



Outline

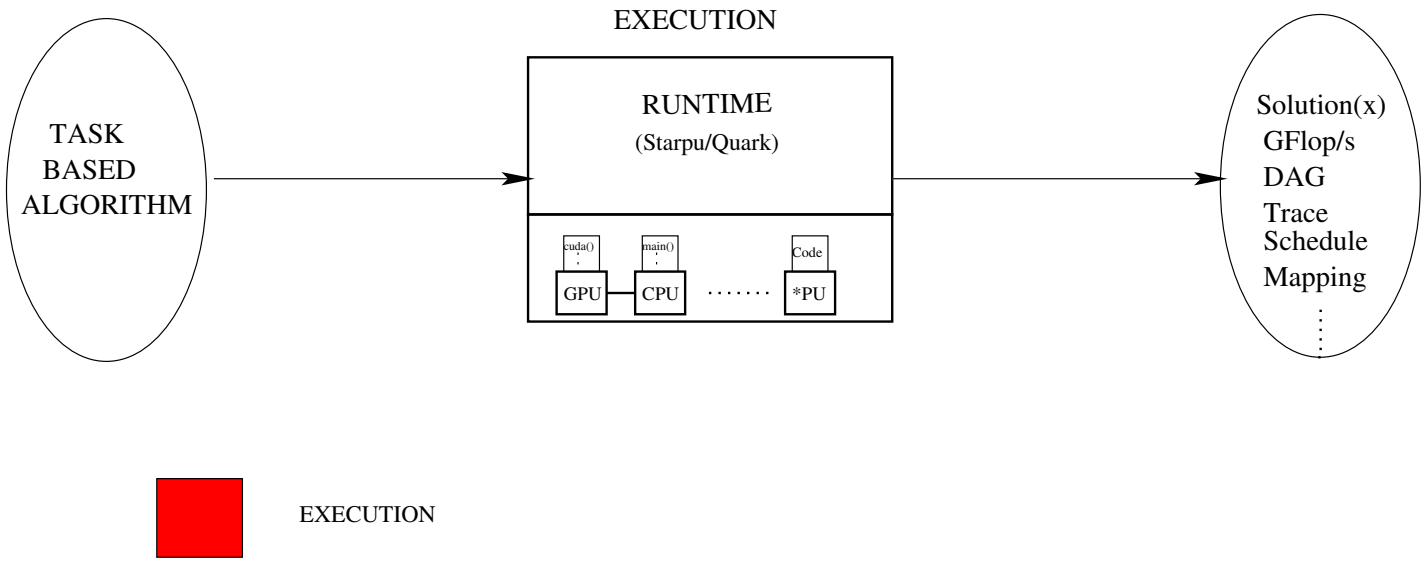
Improving Cholesky through theoretical analysis

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



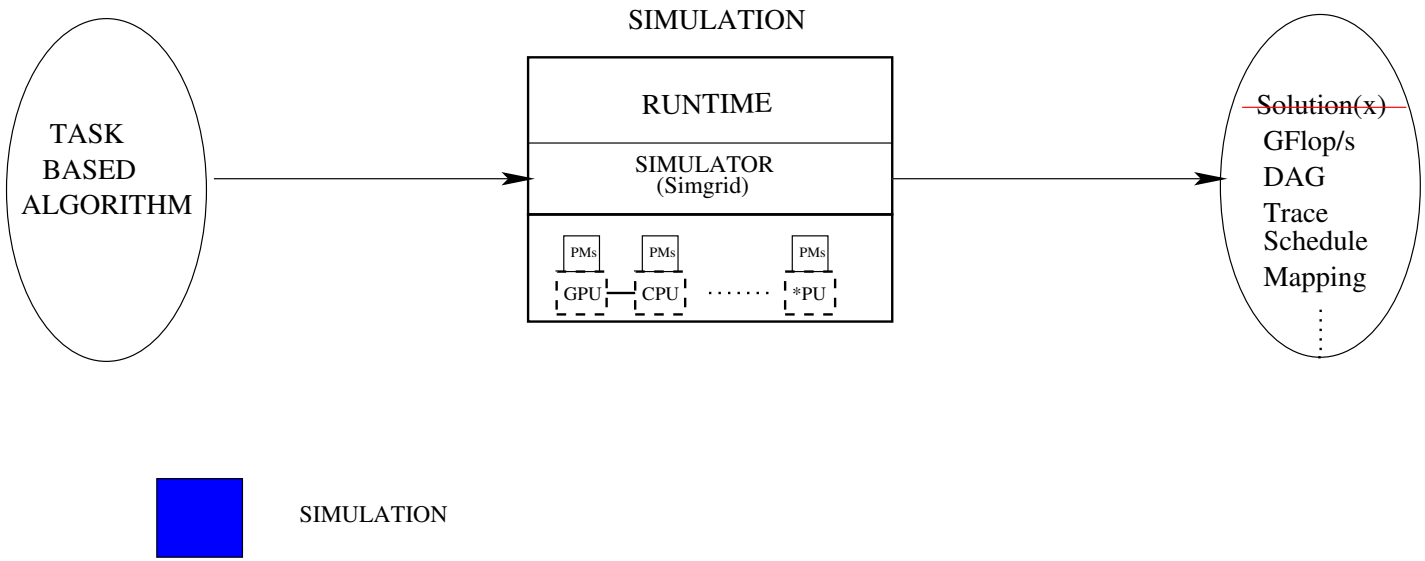
Improving Cholesky through theoretical analysis

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



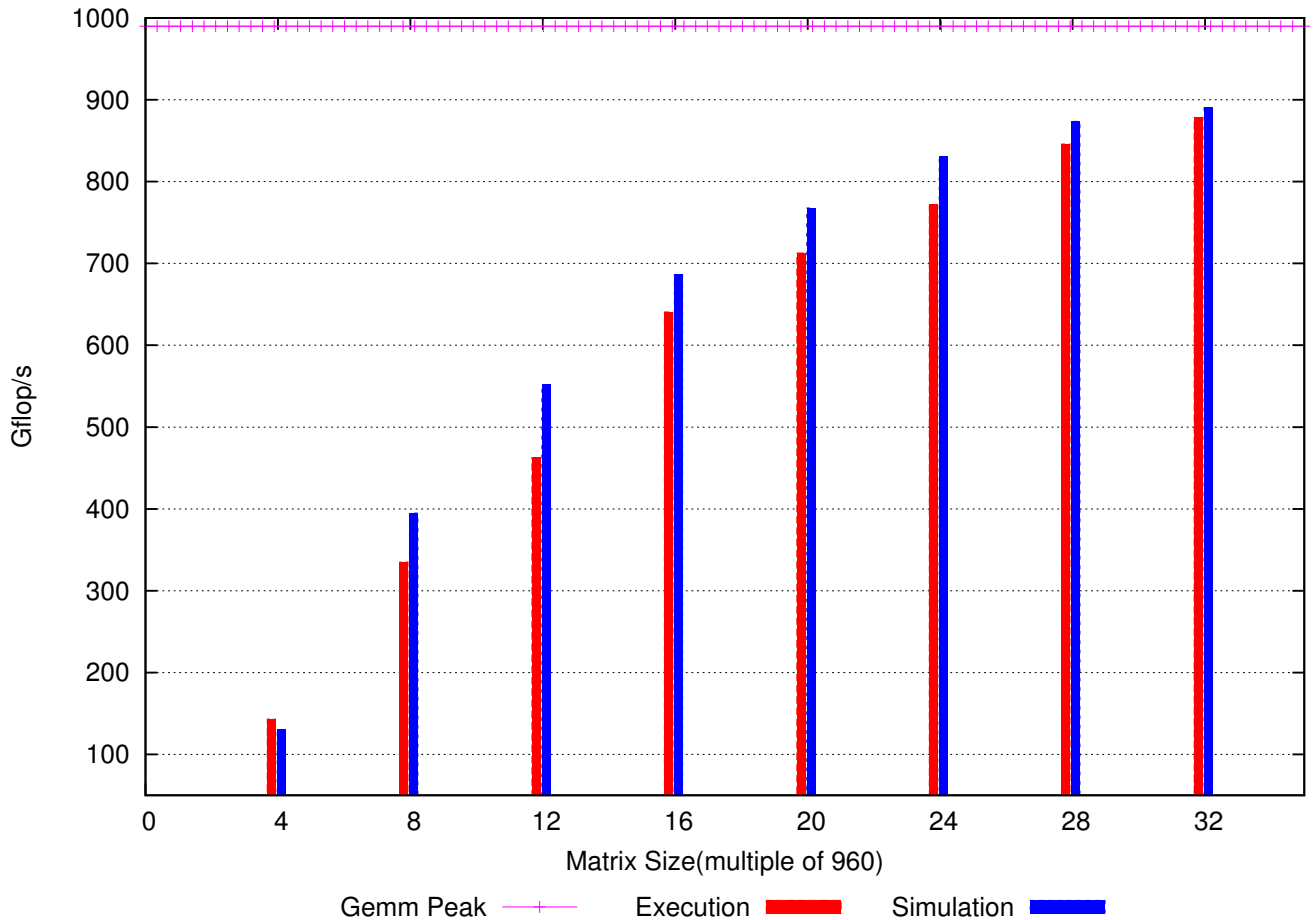
Improving Cholesky through theoretical analysis

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



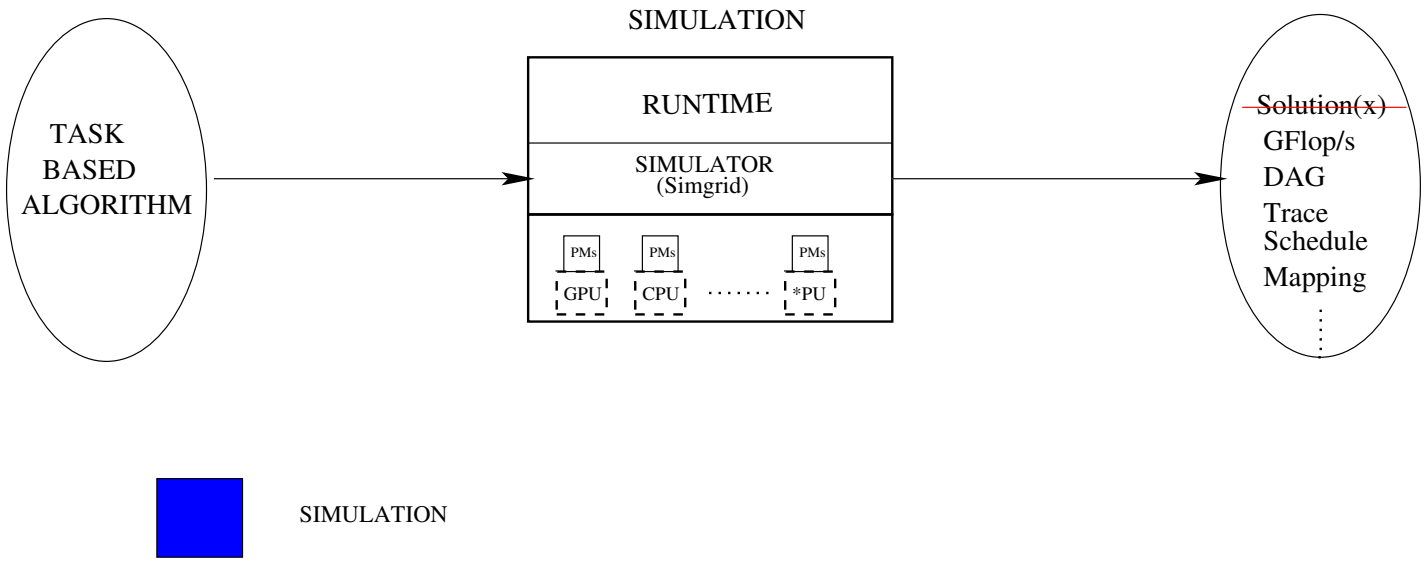
Improving Cholesky through theoretical analysis

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



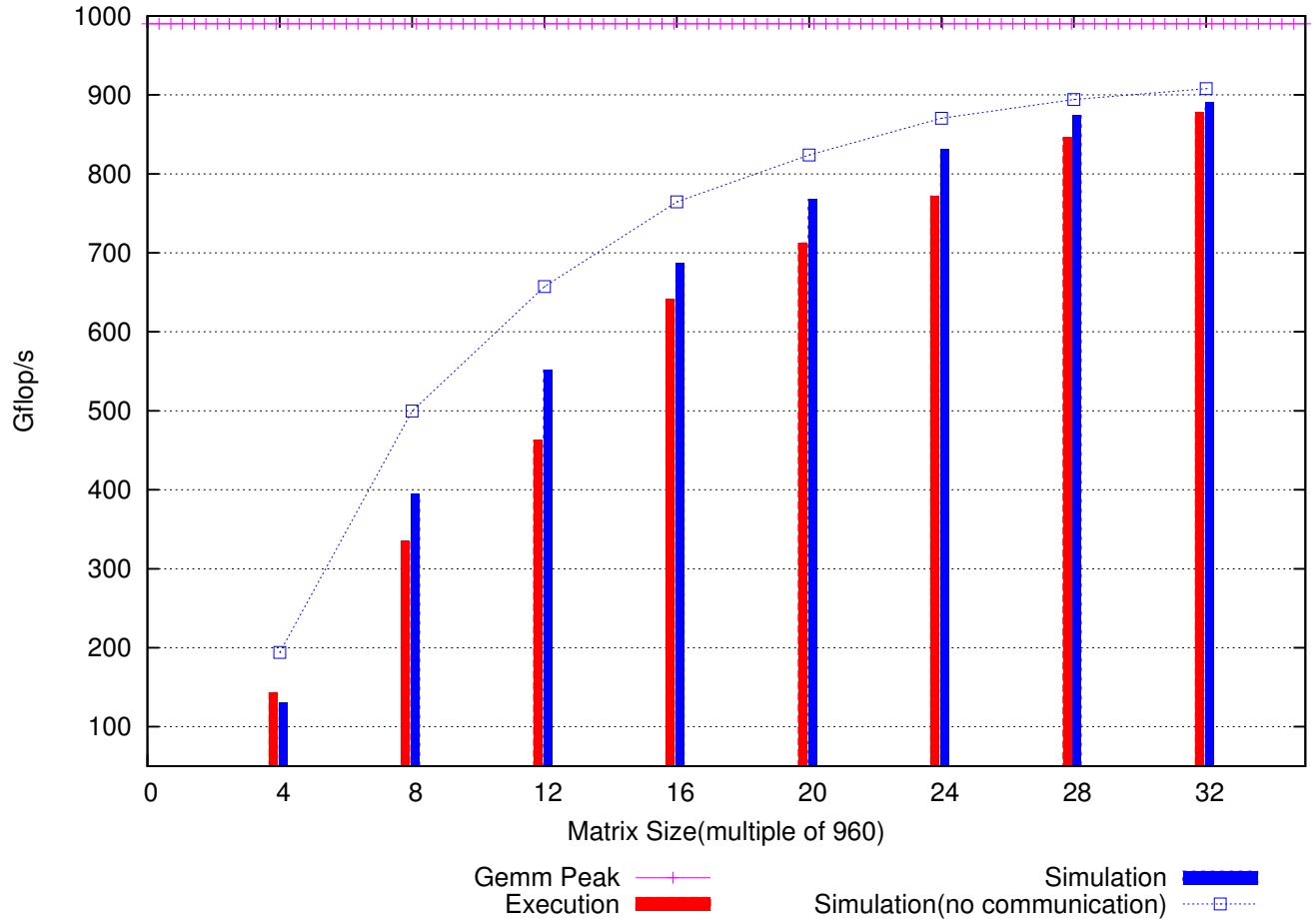
Improving Cholesky through theoretical analysis

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



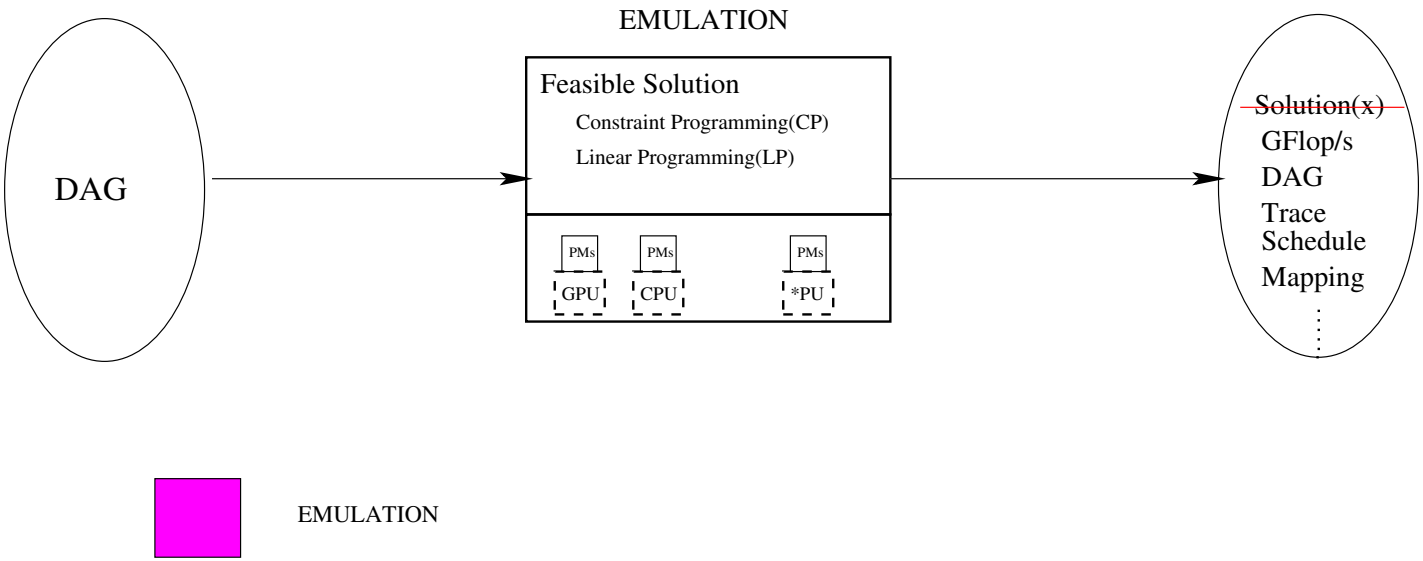
Improving Cholesky through theoretical analysis

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



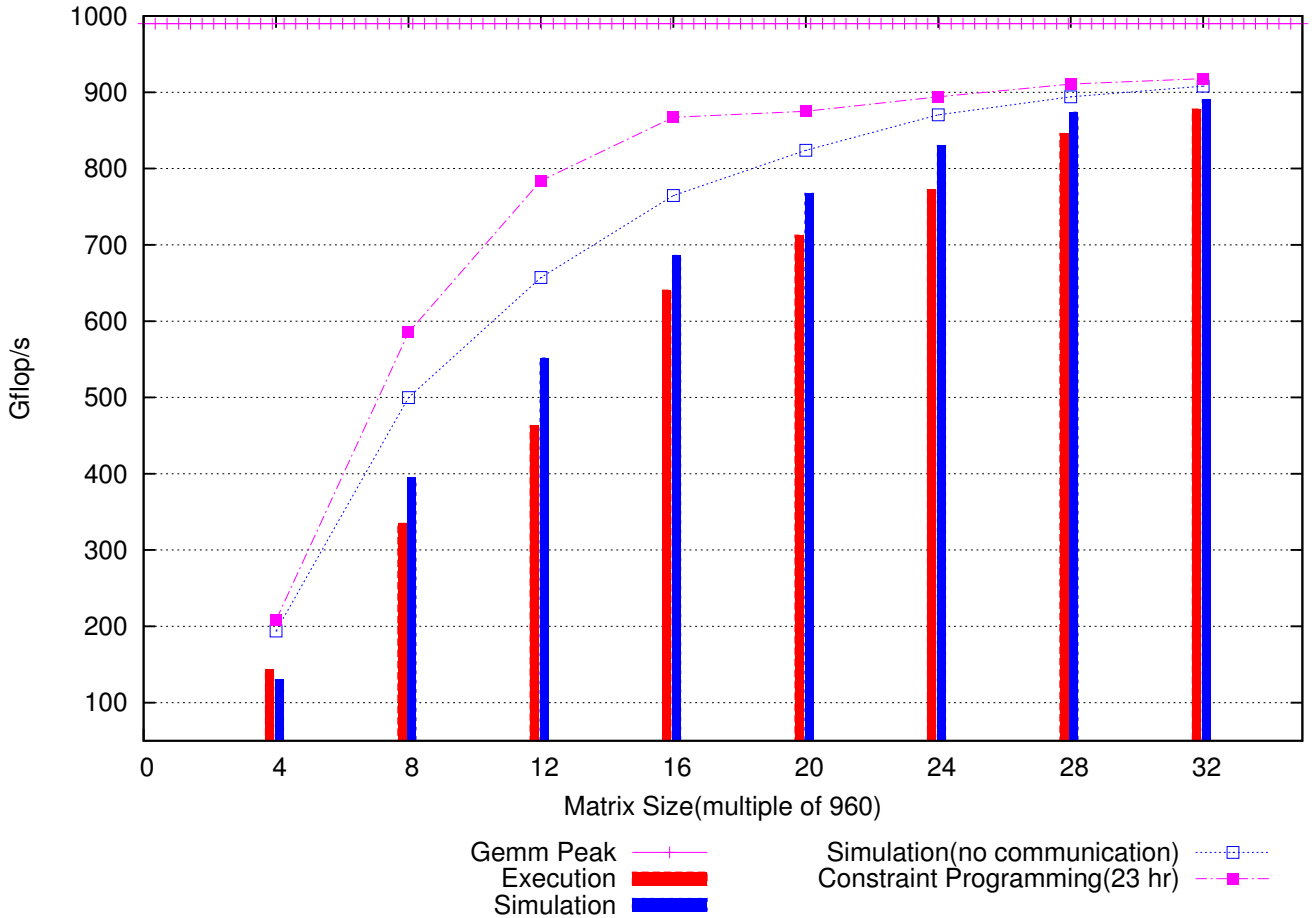
Improving Cholesky through theoretical analysis

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



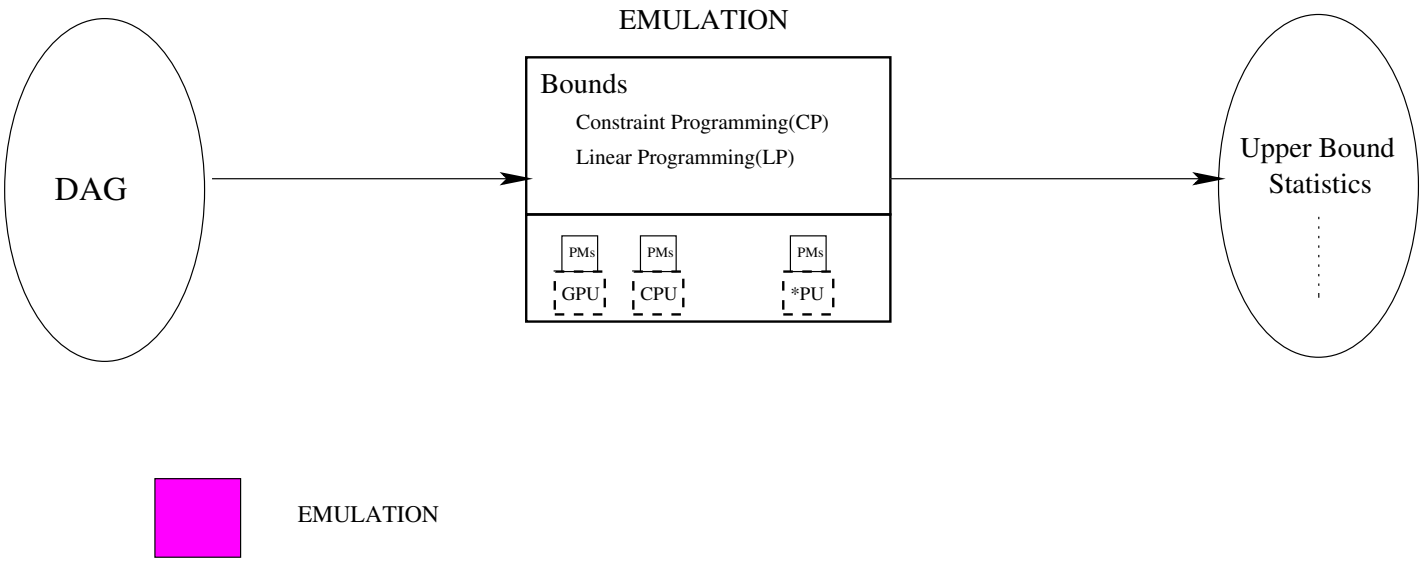
Improving Cholesky through theoretical analysis

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



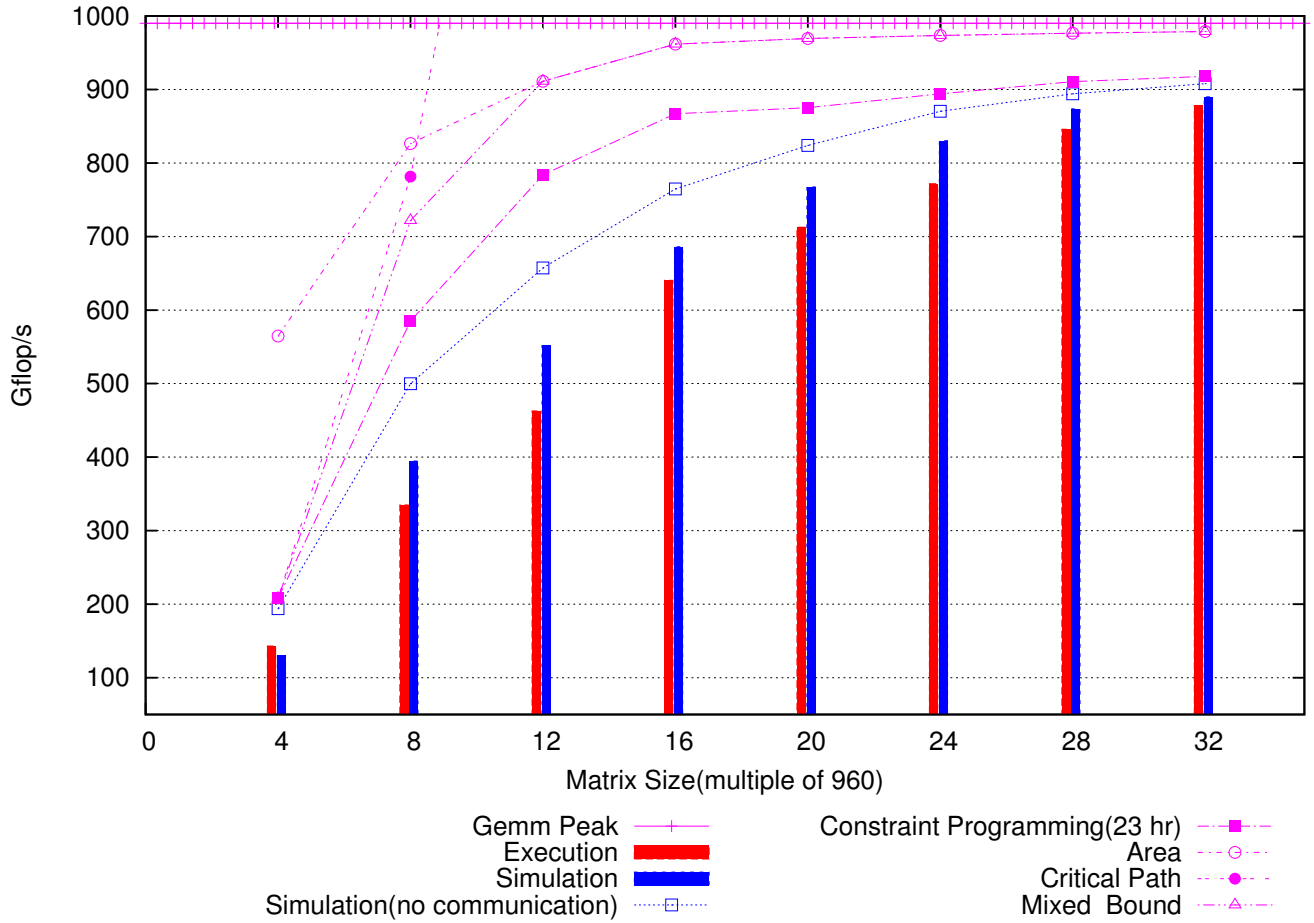
Improving Cholesky through theoretical analysis

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



Improving Cholesky through theoretical analysis

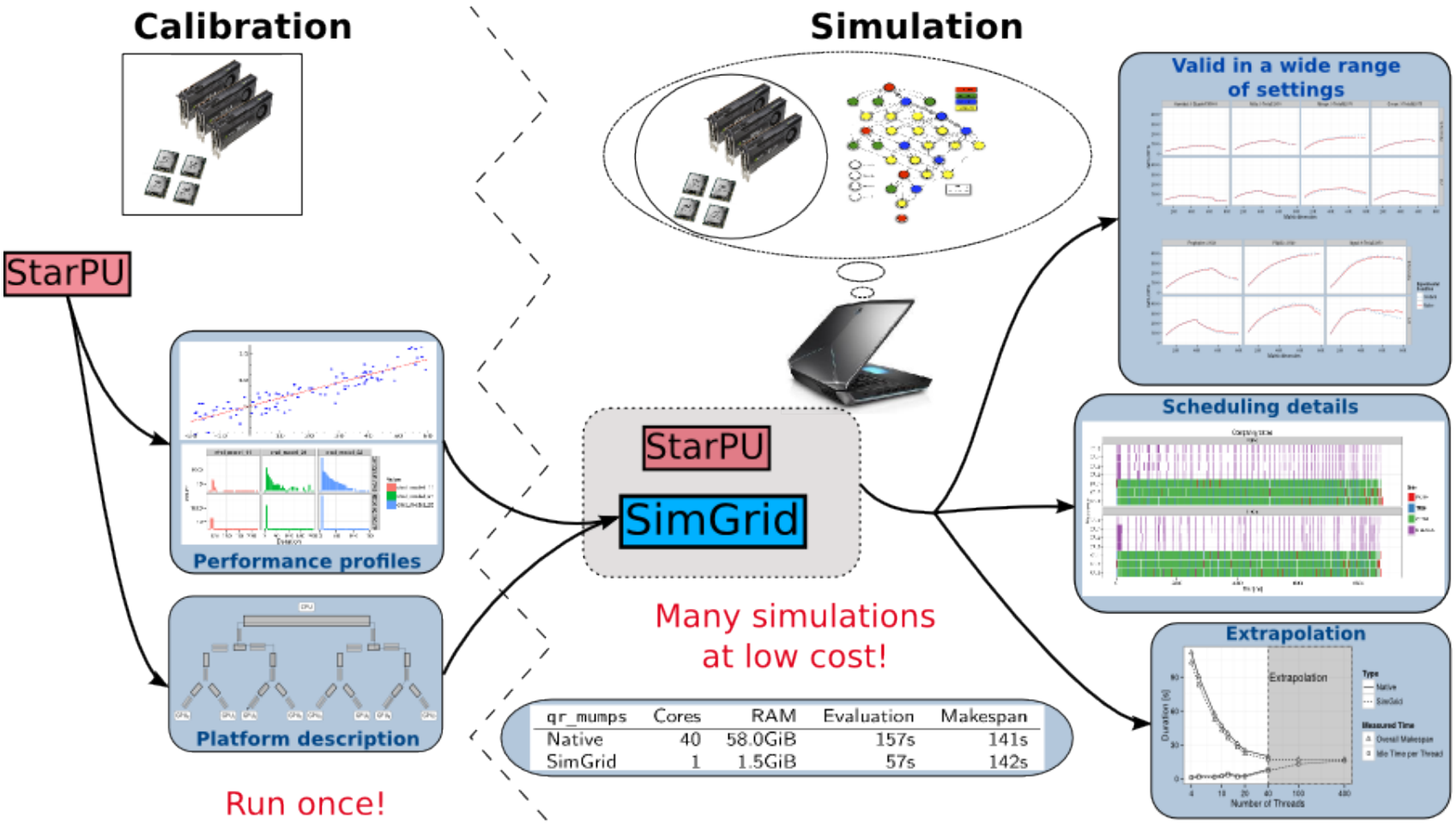
PhD. S. Kumar (HiePACS / Realopt / Runtime)



Outline

StarPU-SimGrid in a nutshell ...

PhD S. Kumar



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

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

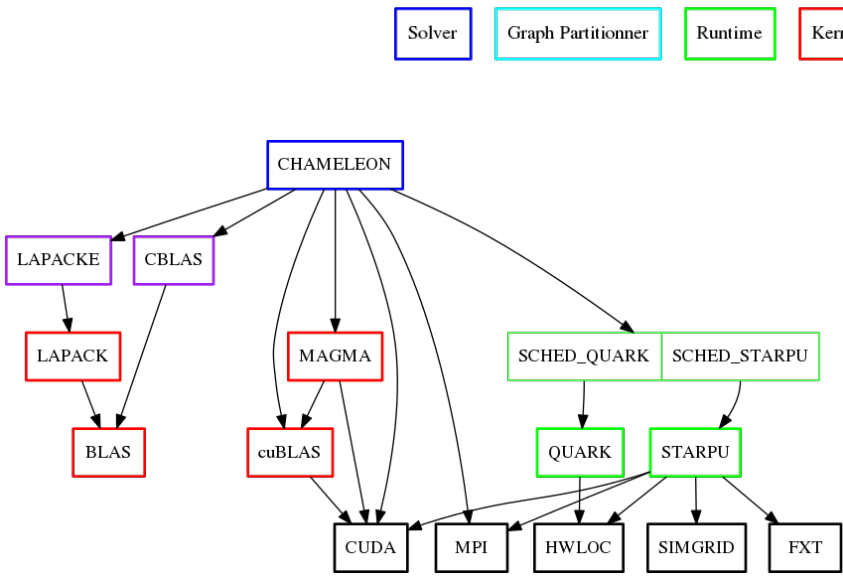
Conclusion and perspectives (2/2)

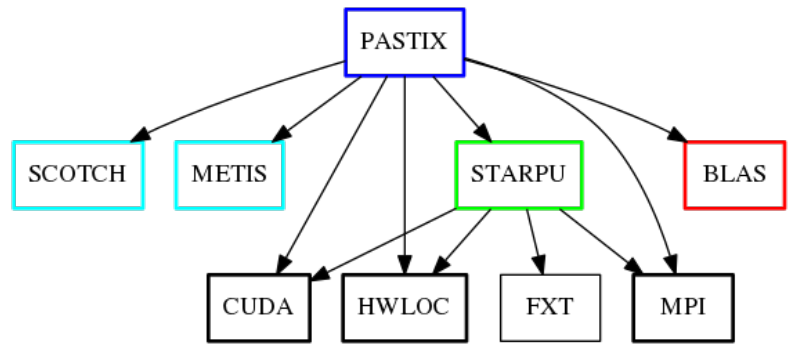
Achieving a software stack running @ exascale

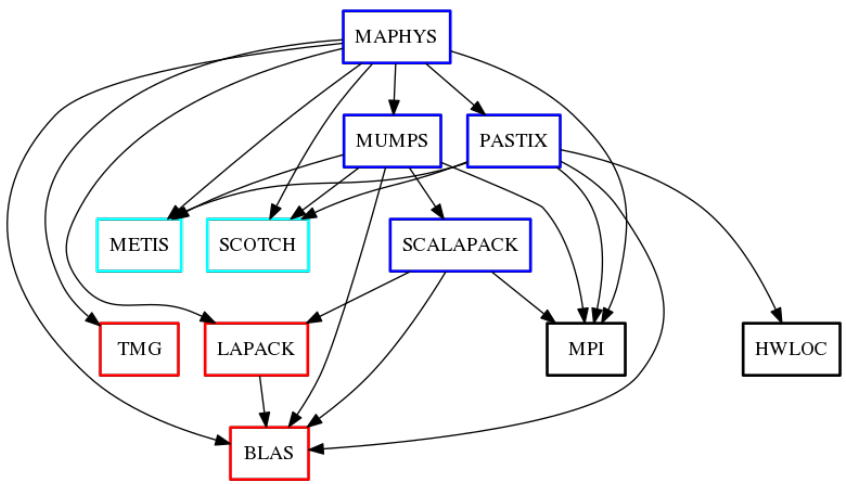
- Numerical algorithms - HiePACS - UTK - UCD - KAUST - IRIT - Roma
- Runtime systems - Storm - UTK
- Automatic parallelization - Corse - UTK
- Scheduling - Realopt - Roma
- Communication - Tadaam
- Simulation and reproducibility - Polaris
- Resilience - HiePACS - UTK
- Autotuning - Polytechnique Montreal

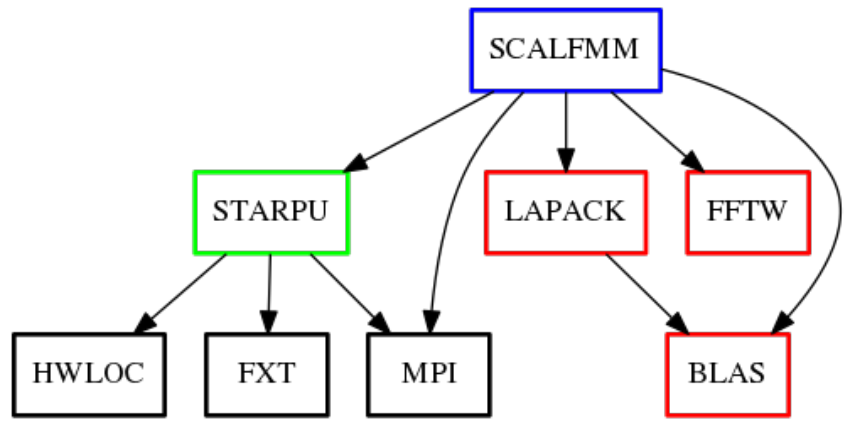
Outline

How to deploy complex HPC software stacks?









Common situations

You rock: install all the stack by yourself !

- specialists on all the stack are rare
- problems of compatibility between versions
- takes a lot of time

You are a numerician not comfortable with software building

- ask someone else to do it
- use pre-installed versions (binary packages, modules)
  - problem: only a couple of versions exist

Wish list

Two kinds of users

1. Top-level users want the best version:
   - a default build with best options to get perfomances regarding the platform
2. A specialist wants to have the lead:
   - on the components he operates on
     - flexibility to set his version
   - but may not care about many dependencies
     - automatic choice of best options

Requirements

- A simple process to install a default version
- A flexible way to choose build variants
  - choose compiler, software versions
  - enable components, *e.g.* MPI : YES/NO
  - build options, *e.g.* `--enable-debug`
- Be able to install it on a remote cluster
  - no root permissions
  - no internet access (not necessarily)

Existing toolboxes

- PETSc
  - scientific library for solving PDEs in parallel MPI+Threads
  - wrappers to external solvers (partitionners, linear algebra, ...)
  - custom python scripts to activate packages
    - detection mode or download+install a web release, great !
    - detection problems, fixed versions to download
- Trilinos
  - similar to PETSc, maybe even broader scope
  - embed one precise version of solvers
  - no tool to install missing third party libraries

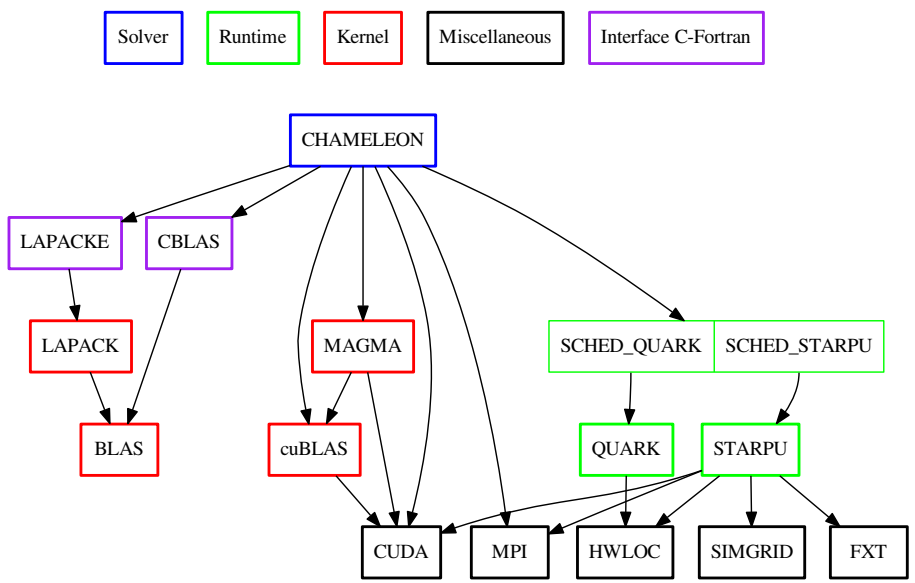
$\Rightarrow$ **no competitive tool to install dependencies**

Outline

Common build-install process for solvers

- use of CMake with similar options
- rely on the same detection of the system and libraries
  - recursive system of CMake Finds
  - if your application depends on Chameleon, in CMake:

```
find_package(CHAMELEON COMPONENTS STARPU MPI CUDA MAGMA FXT)
```



Common build-install process for solvers

- use of CMake with similar options
- rely on the same detection of the system and libraries
  - recursive system of CMake Finds
  - if your application depends on Chameleon, in CMake:

```
find_package(CHAMELEON COMPONENTS STARPU MPI CUDA MAGMA FXT)
```

List of available `find_package` in Morse:

solvers	chameleon, magma, mumps, pastix, plasma, scalapack
runtimes	quark, parsec, starpu
kernels	(c)blas, lapack(e), fftw
misc	(par)metis, (pt)scotch, hwloc, fxt, eztrace

Available online:

https://scm.gforge.inria.fr/anonscm/svn/morse/trunk/morse_distrib/cmake_modules/morse/find

Outline

State of the art: tool to distribute the software stack

- we do not want to reinvent the wheel *i.e.* use an existing solution:
  - Dpkg, 0install, Gub, Guix/Nix, Easybuild, ...
- classical package managers cannot meet our requirements
  - no root permissions, build variants easy to give, a mode to handle non open software (Intel MKL, nvidia CUDA)
- **Spack** a custom tool to install HPC libraries will be used

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
```

Spack weaknesses

- Not so mature $\Rightarrow$ bugs, not robust enough?
- Detection mode is missing ... but will be integrated soon
  - positive exchanges with the main developer
  - reactive to answer
  - we feel that they have the same needs

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 tutorials:

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

http://morse.gforge.inria.fr/tuto_chameleon/

MORSE provides packages to automatically install libraries and its dependencies with  Spack

Dense linear solvers

Chameleon, MAGMA, PLASMA, ScaLAPACK

Sparse linear solvers

HIPS, MaPHyS, MUMPS, PaStiX, qr_mumps, SuiteSparse

Fast Multipole Method solvers

ScalFMM

Runtime systems

QUARK, ParSEC, StarPU

Kernels

(C)BLAS (MKL, Netlib, OpenBlas), LAPACK(E), FFTW

Miscellaneous

(Par)METIS, (PT-)SCOTCH, hwloc, MPI, CUDA, SimGrid, EZTrace

Morse in Spack: continuous integration

https://ci.inria.fr/morse/

	ub 14-04 i386	ub 14-04 amd64	centos 63 amd64	OS X 10.9
cblas	✔	✔	✔	✔
chameleon	✔	✔	✔	✔
eztrace	✔	✔	✘	✘
fftw	✔	✔	✔	✔
fxt	✔	✔	✔	✔
hwloc	✔	✔	✔	✔
maphys	✔	✔	✔	✔
metis	✔	✔	✔	✔
mpich	✔	✔	✔	✔
mumps	✔	✔	✔	✔
netlib-blas	✔	✔	✔	✔
netlib-lapack	✔	✔	✔	✔
openmpi	✔	✔	✔	✔

Morse in Spack: continuous integration

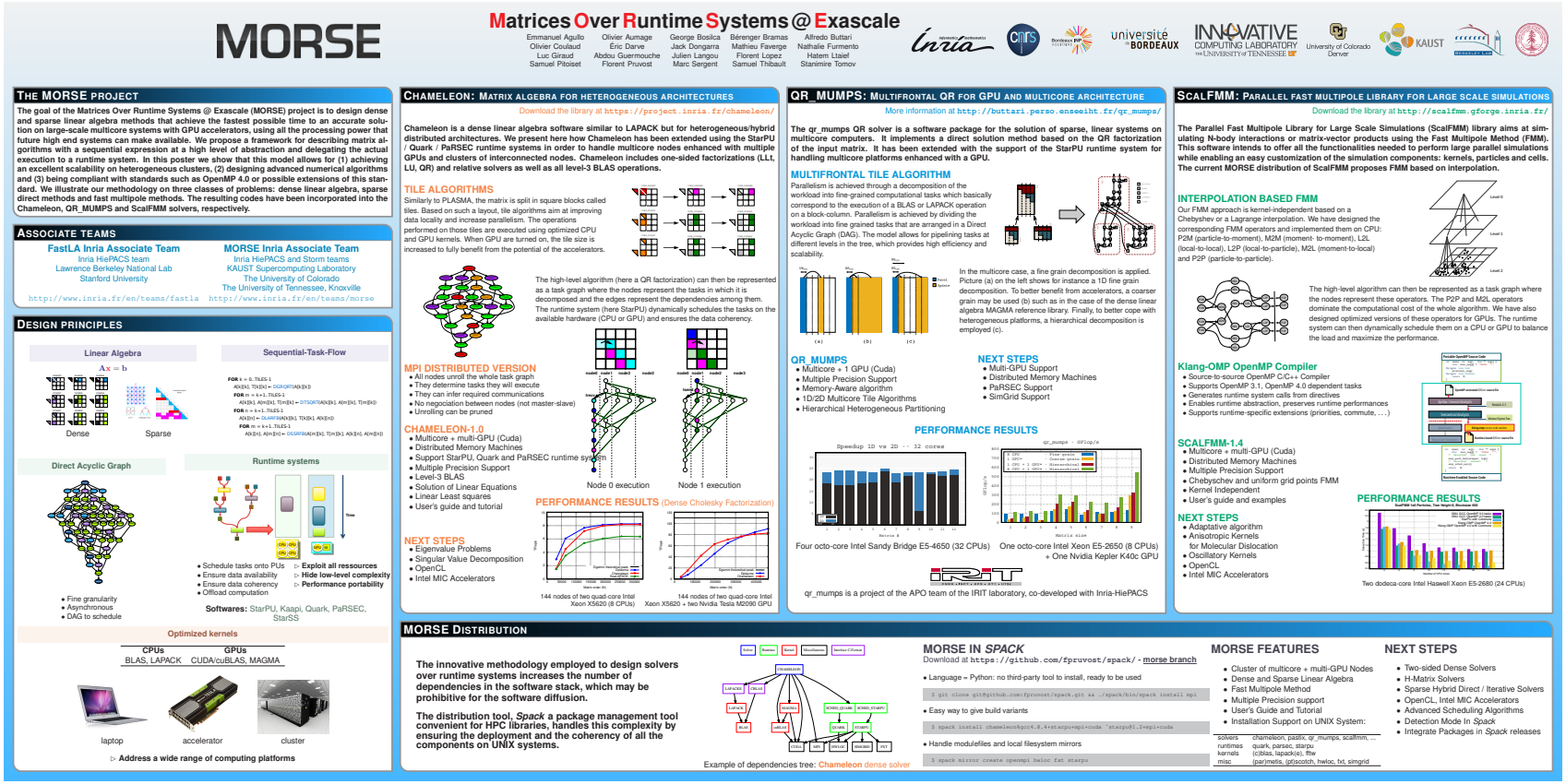
https://ci.inria.fr/morse/

	ub 14-04 i386	ub 14-04 amd64	centos 63 amd64	OS X 10.9
papi	✔	✔	✘	✘
parmetis	✔	✔	✔	✔
parsec	✔	✔	✔	✔
pastix	✔	✔	✔	✔
plasma	✔	✔	✔	✔
qr_mumps	✔	✔	✘	✘
quark	✔	✔	✔	✔
scalapack	✔	✔	✔	✔
scalfmm	✔	✔		✔
scotch	✔	✔	✔	✔
simgrid	✔	✔	✘	✔
starpu	✔	✔	✔	✔
suitesparse	✔	✔	✔	✔
vite	✔	✔	✘	✘

SC15 poster session

To get an updated view of Morse project:

https://gforge.inria.fr/frs/download.php/file/35233/poster-morse-sc15.pdf



Outline

Deliver a V0 of the solver stack distribution:

- Add some missing MPI packages
- Communication:
  - Documentation: prerequisites, features, limitations, compatibility issues
  - Website

Task-based solvers for users - an open question

Unify task-based solvers

- Installation of solvers is a step
  - common way to install the solver stacks
- What about their integration, usage in upper-level programs
- Is it possible to factorize something between solvers
  - Chameleon, Hips, MaPHyS, PaStiX, ScalFMM
    - change some solvers API, or
    - drive existing solvers with an intermediate layer (converters)
  - tutorial, documentation, vocabulary?