Task-based Sparse and Data-sparse Solvers on Top of Runtime Systems

Sparse Days, Toulouse, 2016 June 30-31

HiePACS/RealOpt/Storm/TADaaM
IRIT/Kaust/UCD/UTK/Stanford
INRIA Bordeaux Sud-Ouest

Outline

MORSE

Matrices Over Runtime Systems @ Exascale

INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

Inria
INVENTEURS DU MONDE NUMÉRIQUE

KAUST

University of Colorado Denver

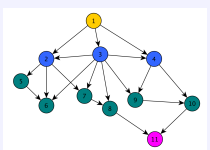
Linear algebra

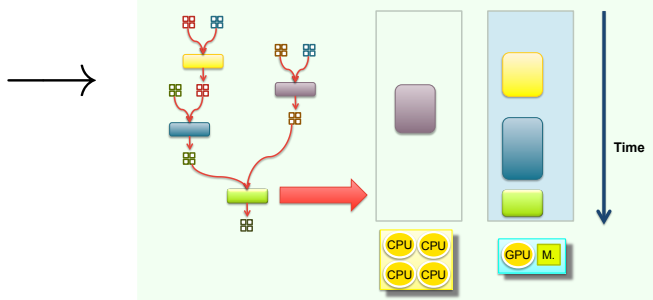
$$AX = B$$

Sequential-Task-Flow

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

Direct Acyclic Graph





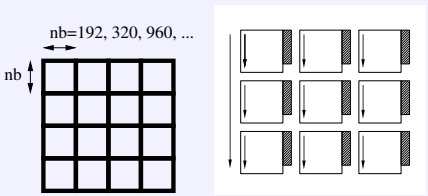
Heterogeneous platforms




The STF model: dense linear algebra case study

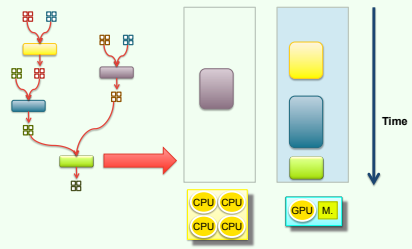
Chameleon = Sequential Task Flow (STF) design of dense linear algebra tiles algorithms (derived from PLASMA) on top of runtime systems

Tile matrix layout



Runtime systems

- QUARK
- StarPU



STF algorithms

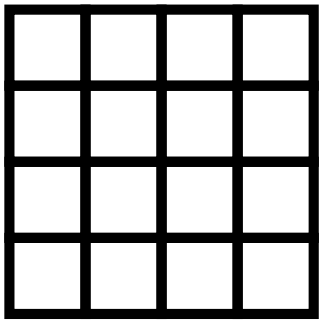
```
for (j = 0; j < N; j++){
  POTRF (A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (A[i][j], A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (A[i][i], A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (A[i][k], A[i][j],
            A[k][j]);
  }
}
```

Optimized kernels

- BLAS, LAPACK
- cuBLAS, MAGMA

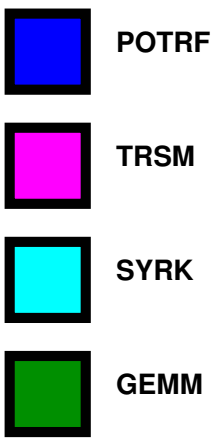
STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent (Inria Storm / CEA)



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__();
```

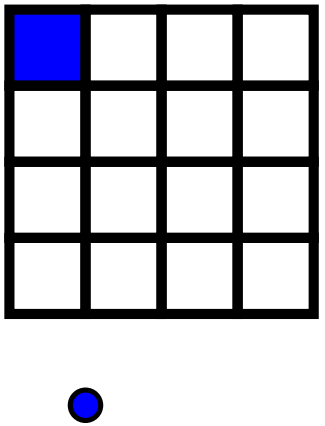


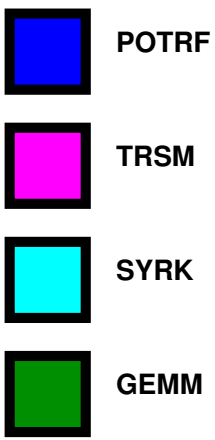
STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent (Inria Storm / CEA)

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__();
```



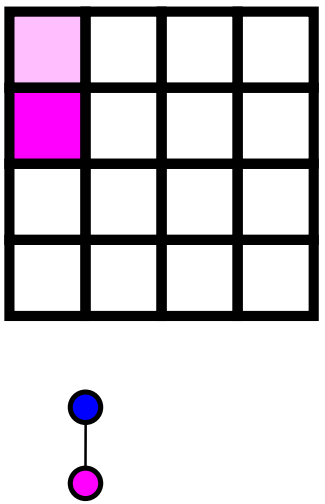


STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent (Inria Storm / CEA)

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__();
```

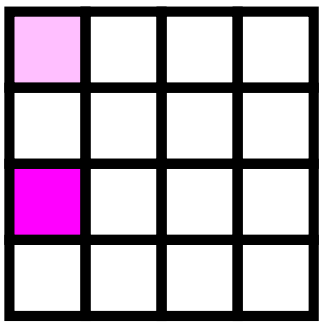


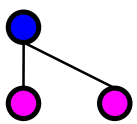
STF Cholesky Algorithm on homogeneous node

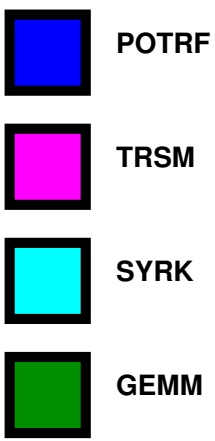
PhD Marc Sergent (Inria Storm / CEA)

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__();
```





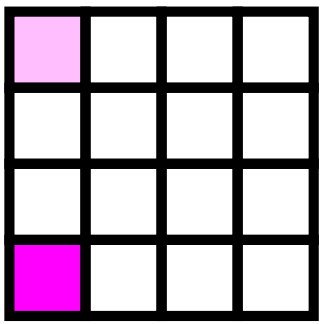


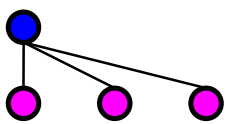
STF Cholesky Algorithm on homogeneous node

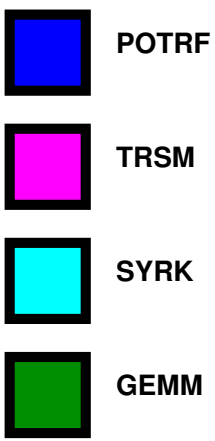
PhD Marc Sergent (Inria Storm / CEA)

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__();
```





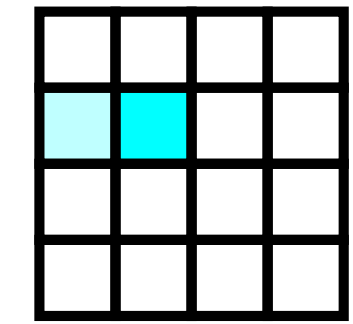


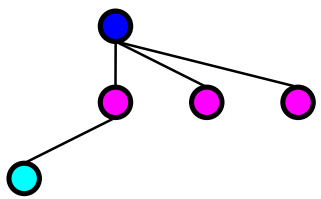
STF Cholesky Algorithm on homogeneous node

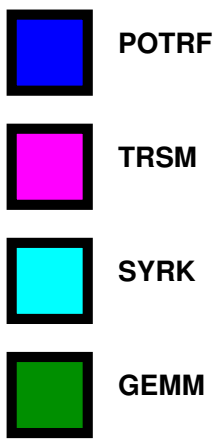
PhD Marc Sergent (Inria Storm / CEA)

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__();
```





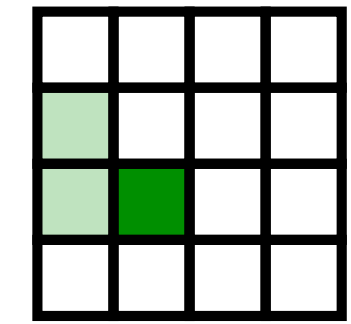


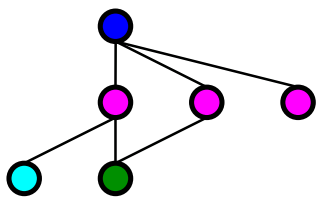
STF Cholesky Algorithm on homogeneous node

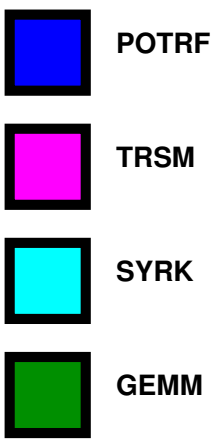
PhD Marc Sergent (Inria Storm / CEA)

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__();
```





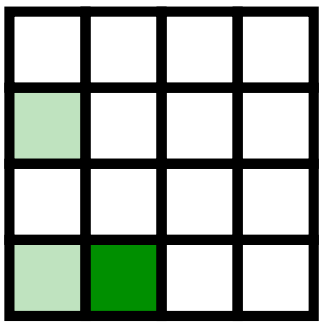


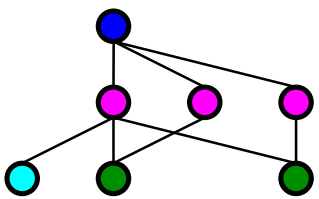
STF Cholesky Algorithm on homogeneous node

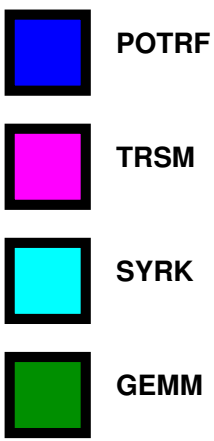
PhD Marc Sergent (Inria Storm / CEA)

work on tiles $\rightarrow$ 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__();
```





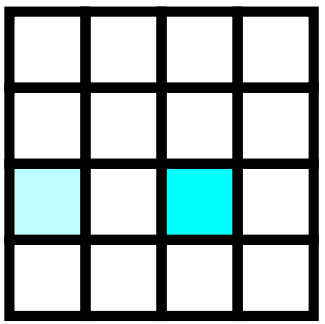


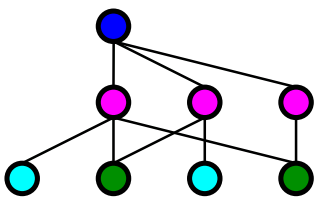
STF Cholesky Algorithm on homogeneous node

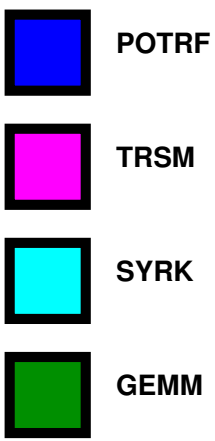
PhD Marc Sergent (Inria Storm / CEA)

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__();
```





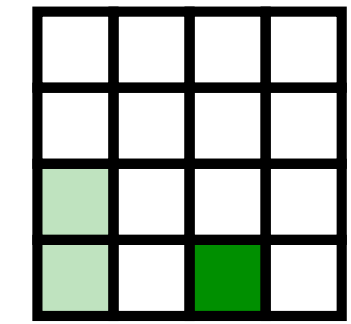


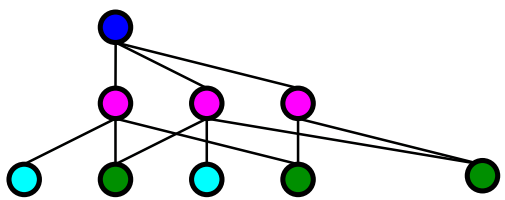
STF Cholesky Algorithm on homogeneous node

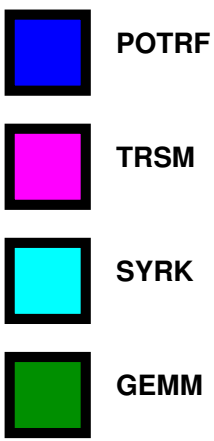
PhD Marc Sergent (Inria Storm / CEA)

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__();
```





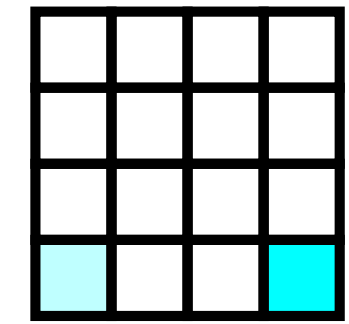


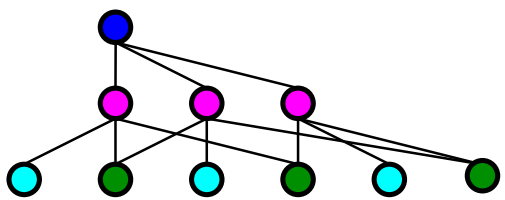
STF Cholesky Algorithm on homogeneous node

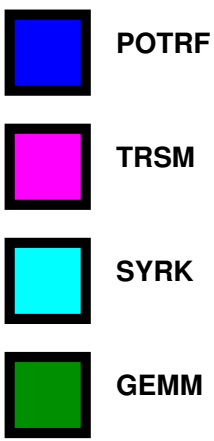
PhD Marc Sergent (Inria Storm / CEA)

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__();
```





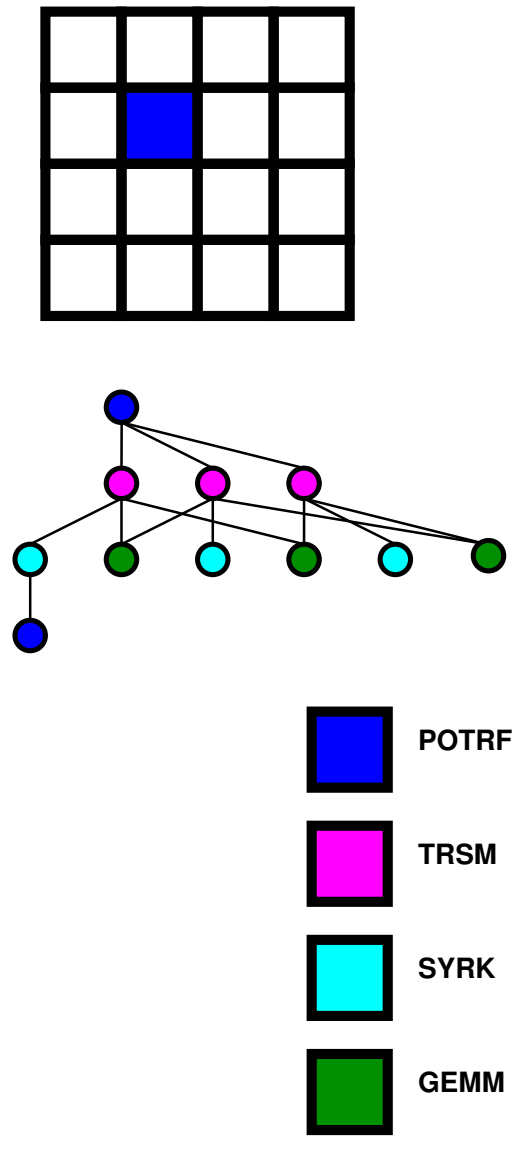


STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent (Inria Storm / CEA)

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__();
```

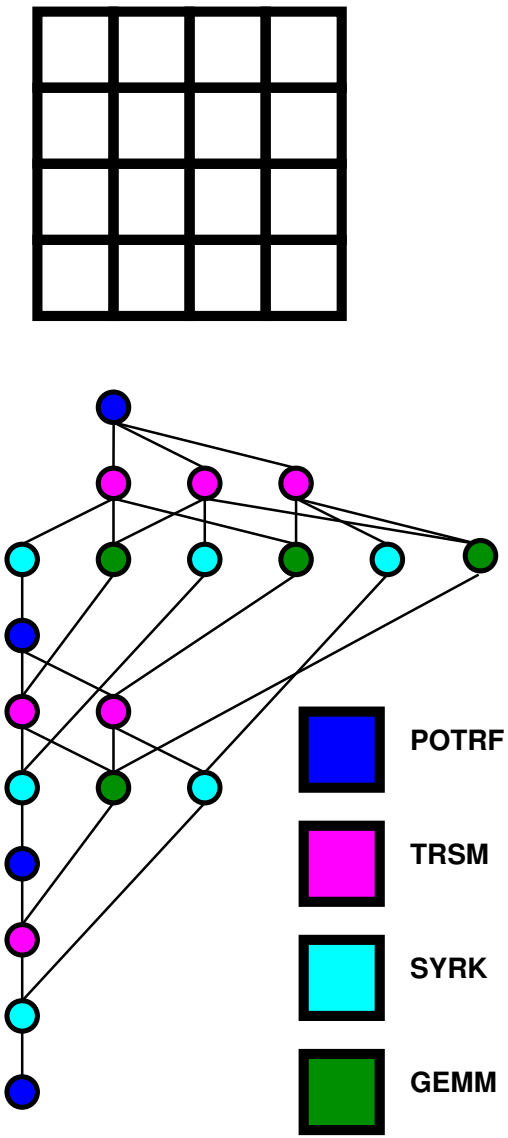


STF Cholesky Algorithm on homogeneous node

PhD Marc Sergent (Inria Storm / CEA)

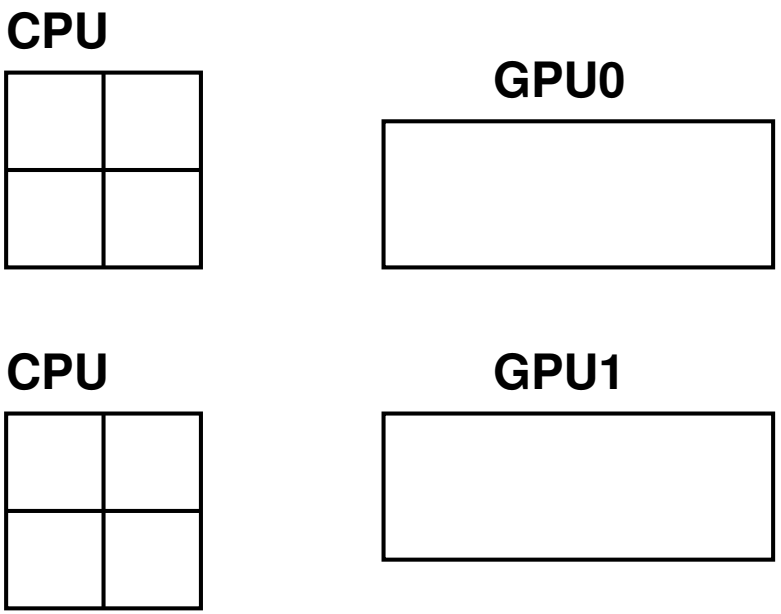
work on tiles $\rightarrow$ CPU kernels

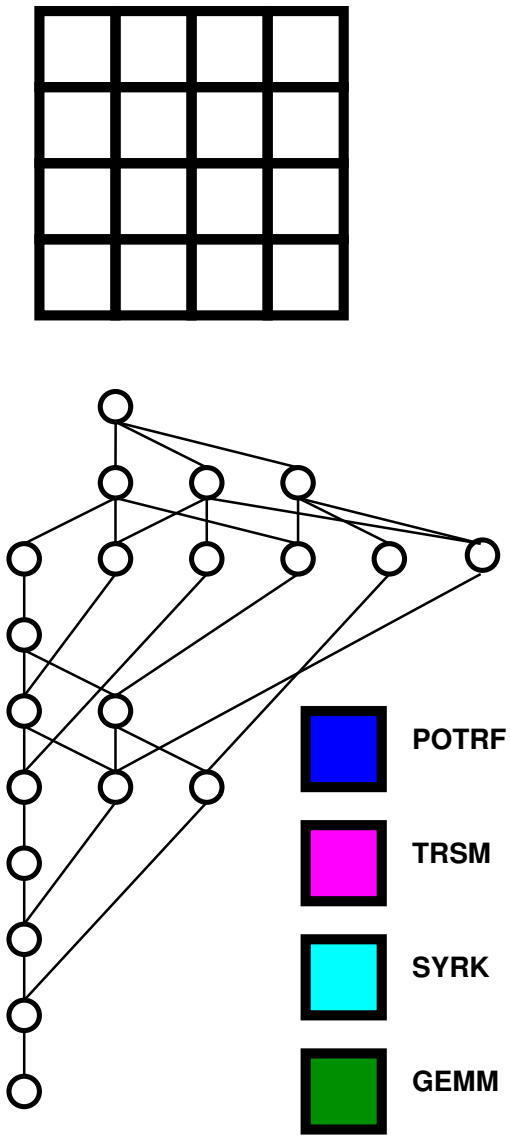
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```



On an heterogeneous node

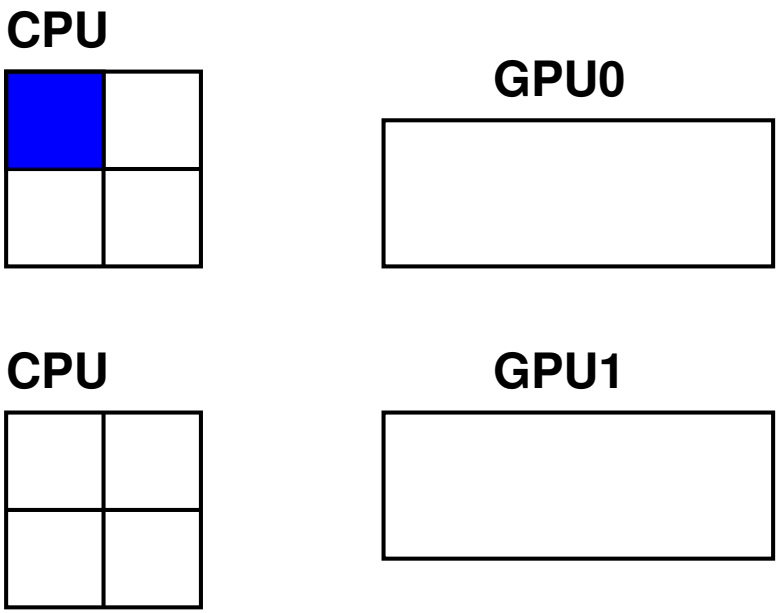
work on tiles $\rightarrow$ CPU + GPU kernels

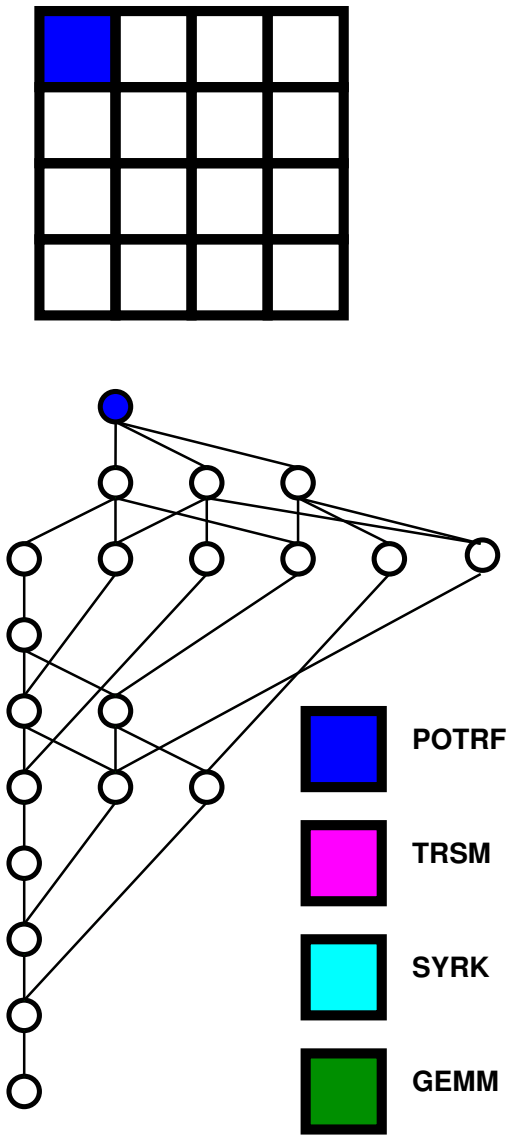




On an heterogeneous node

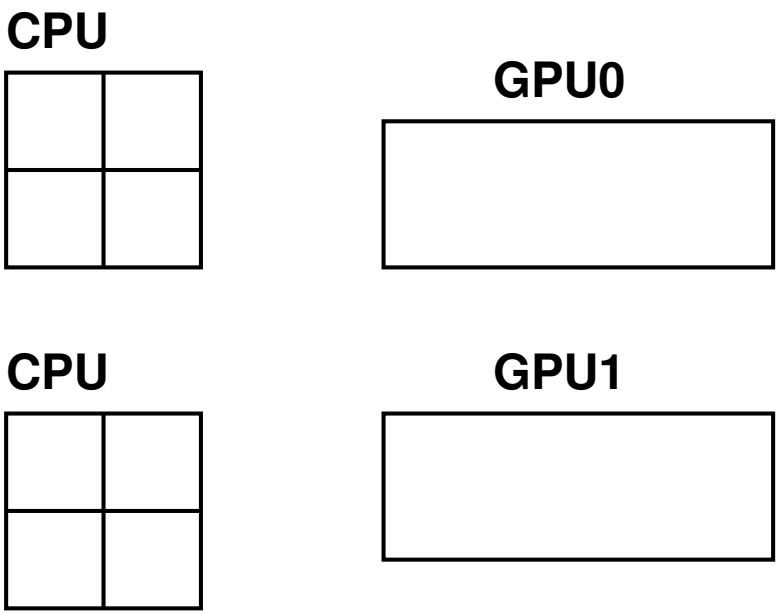
work on tiles $\rightarrow$ CPU + GPU kernels

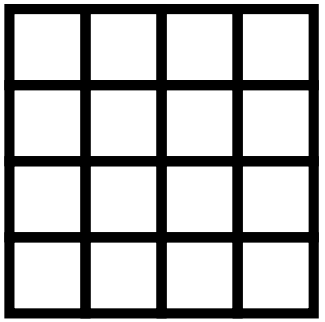


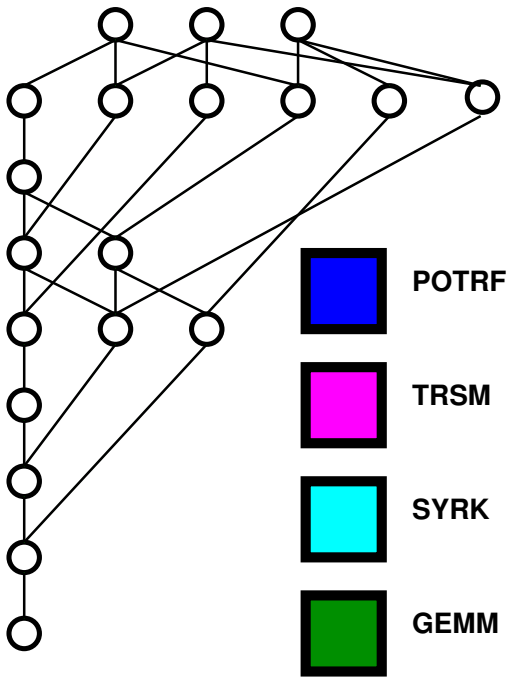


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

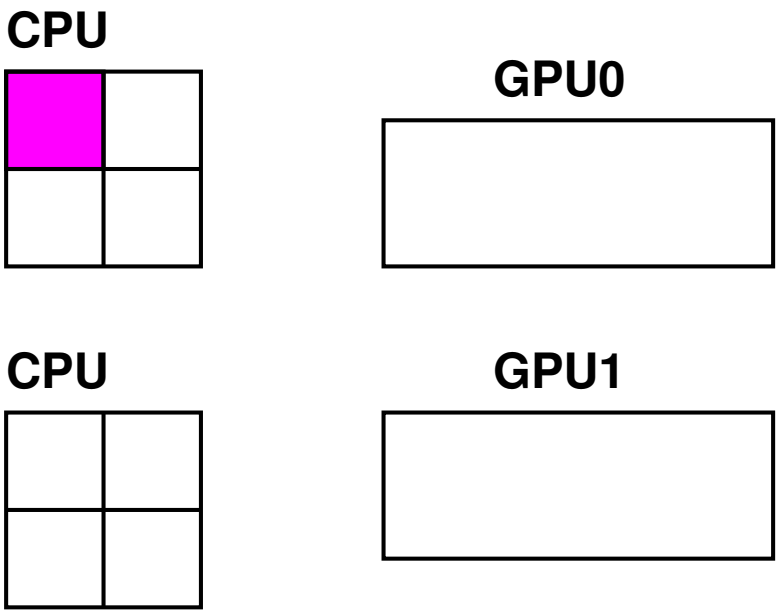


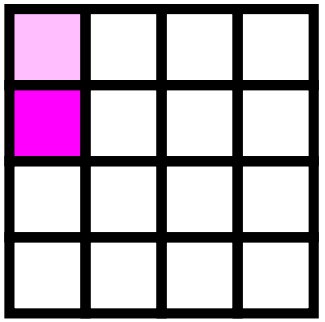


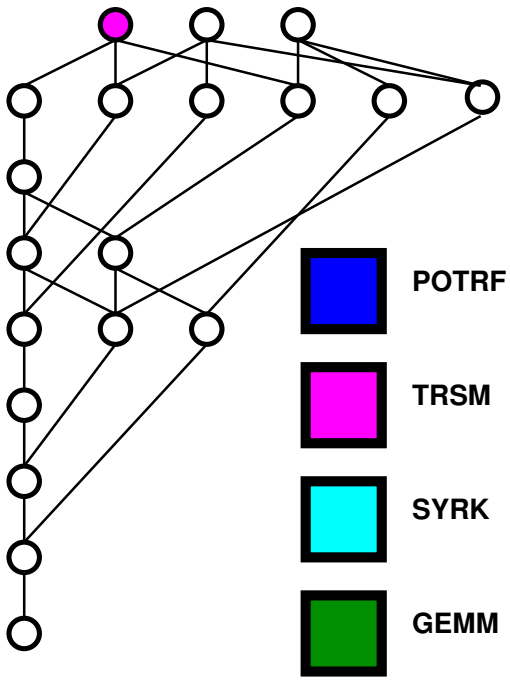


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

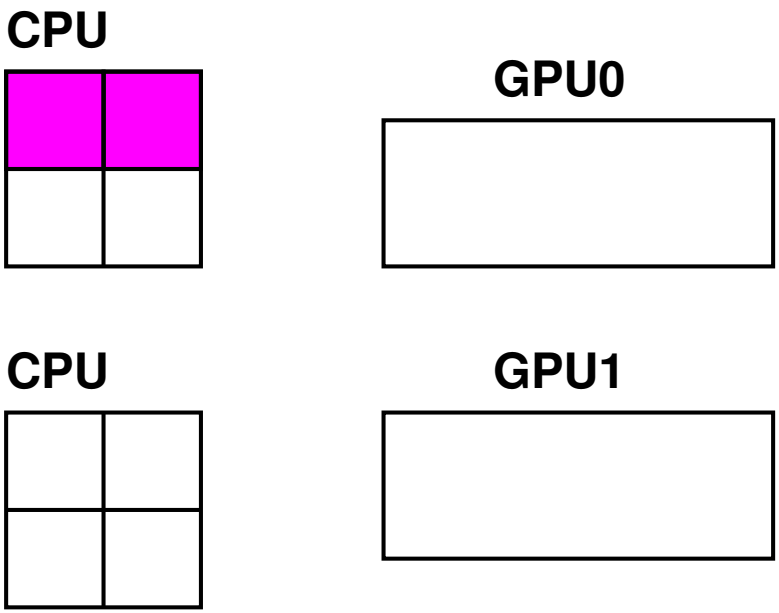


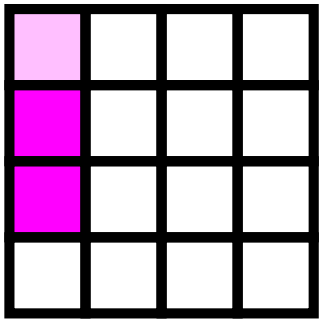


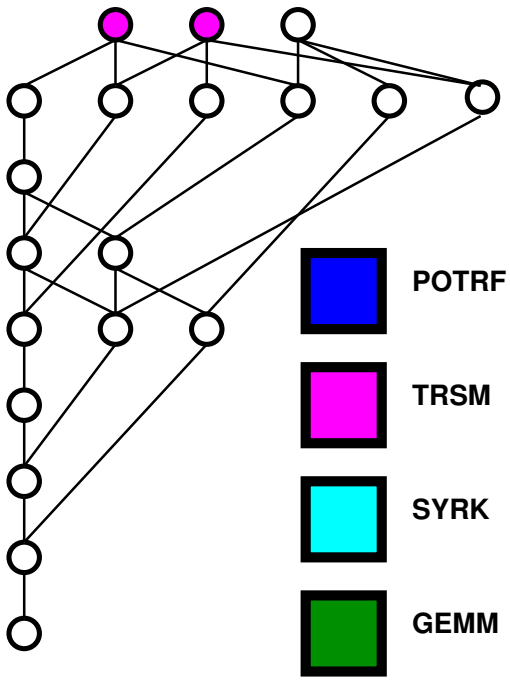


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

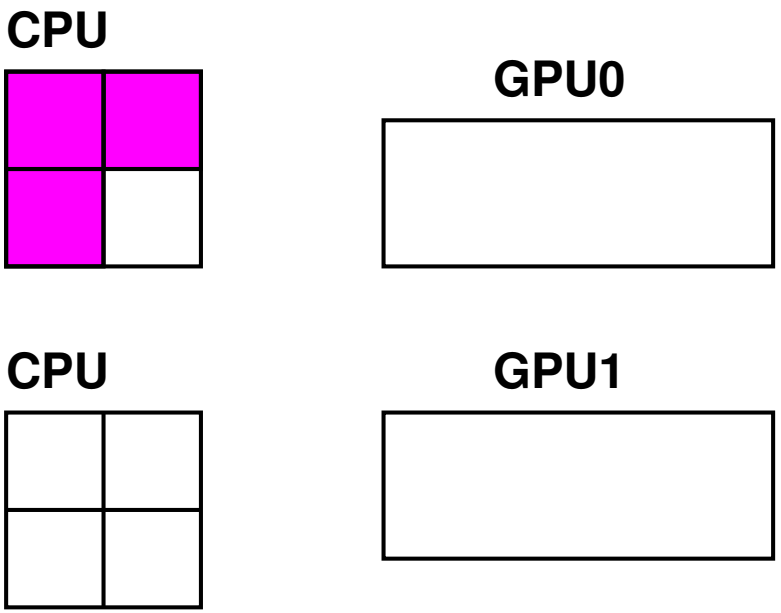


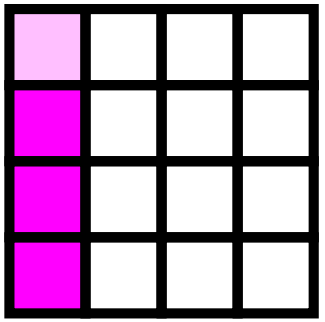


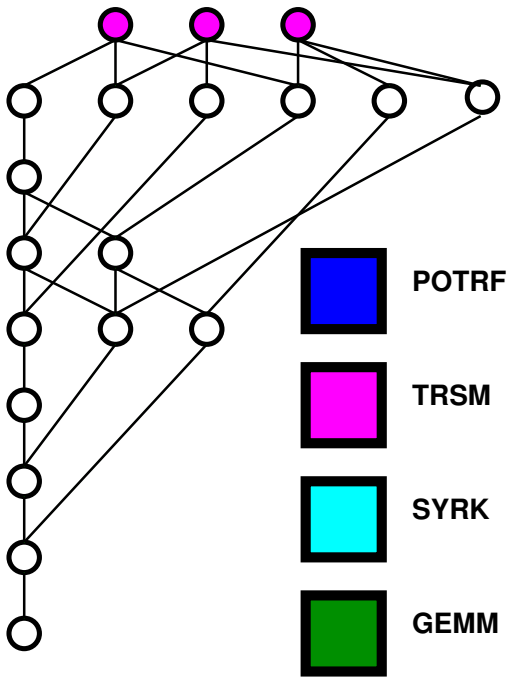


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

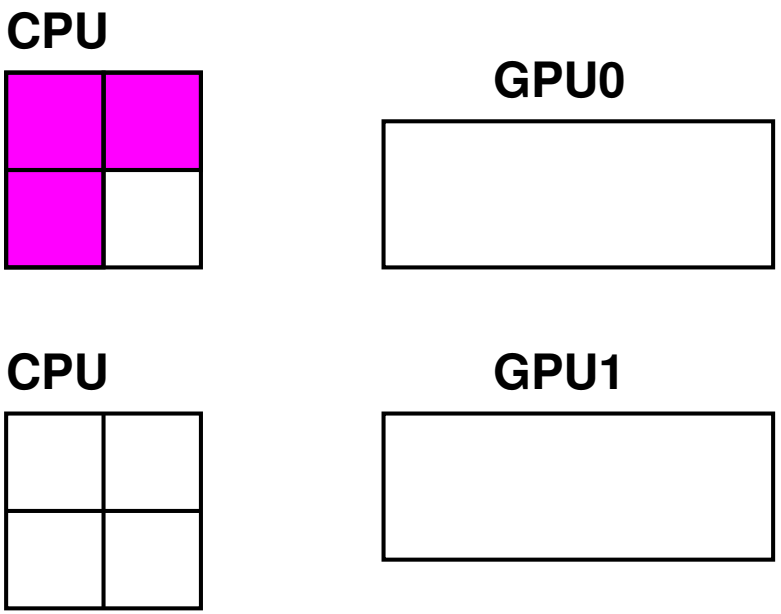


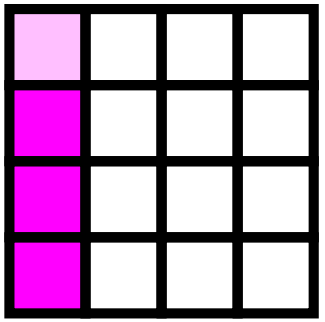


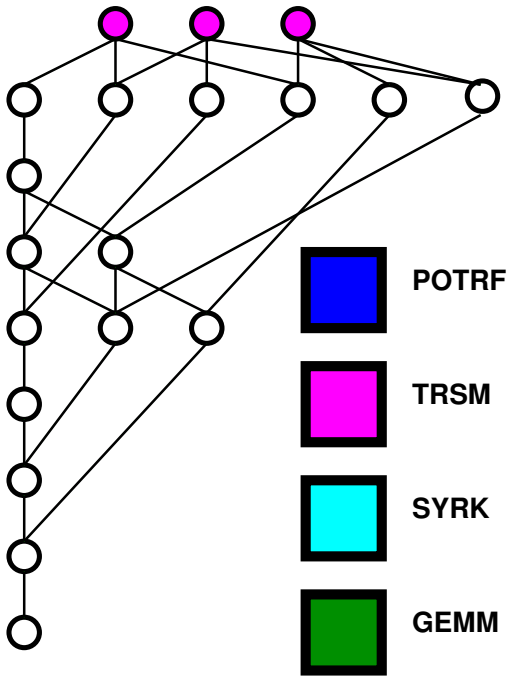


On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels



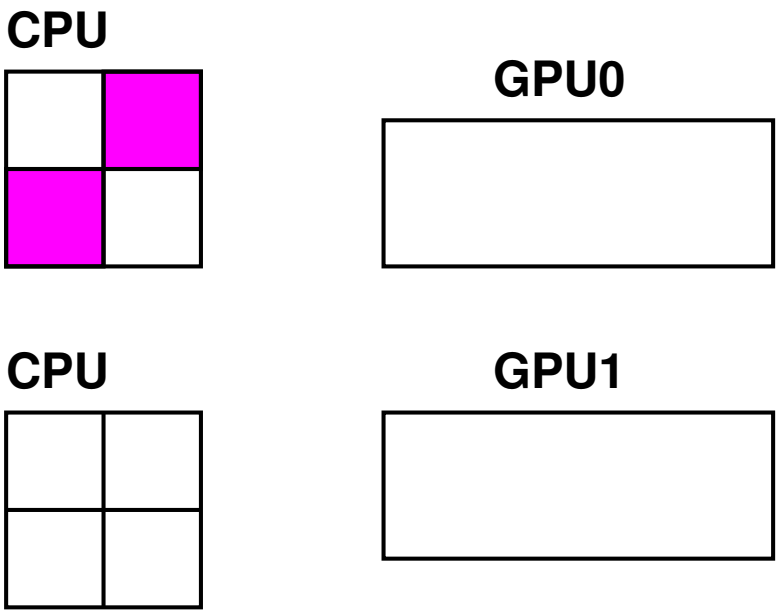


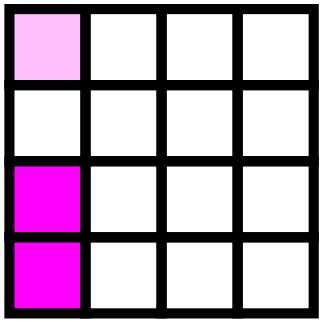


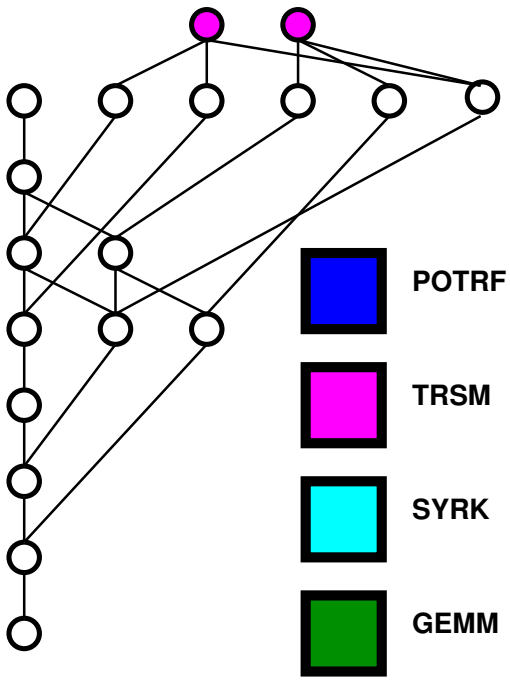
- Handles dependencies

On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels



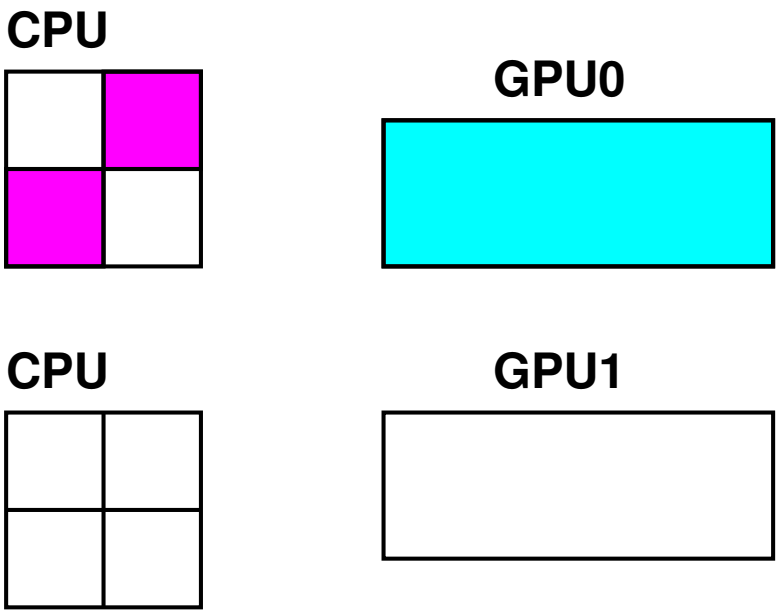


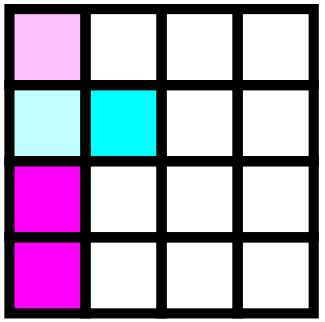


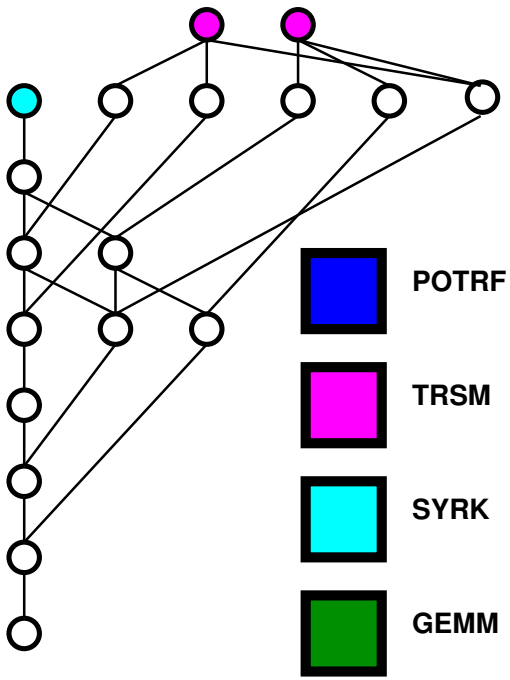
- Handles dependencies

On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels



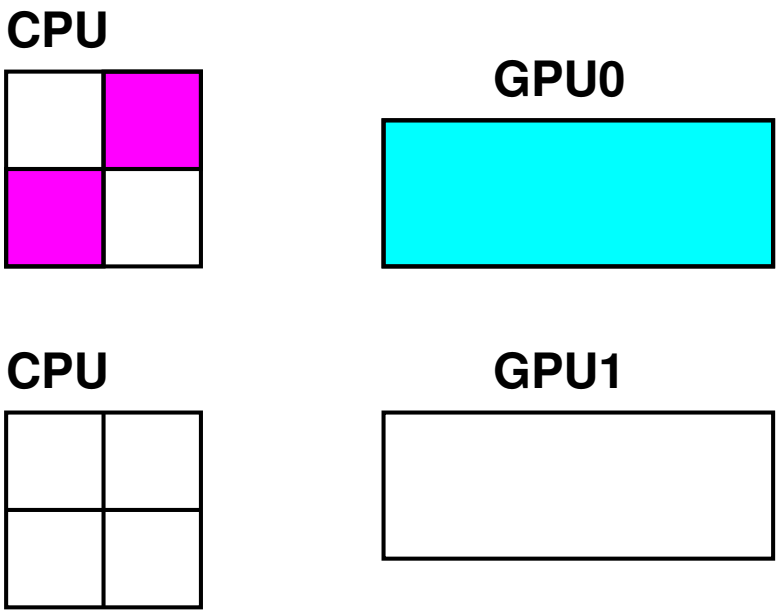


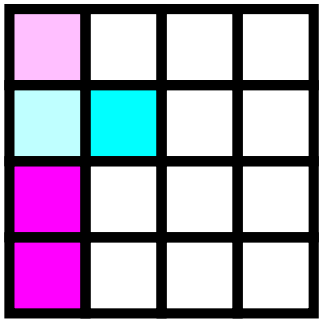


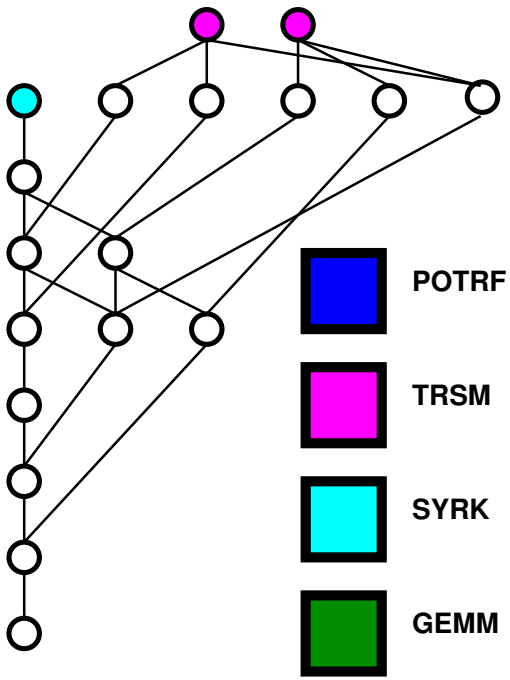
- Handles dependencies

On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels



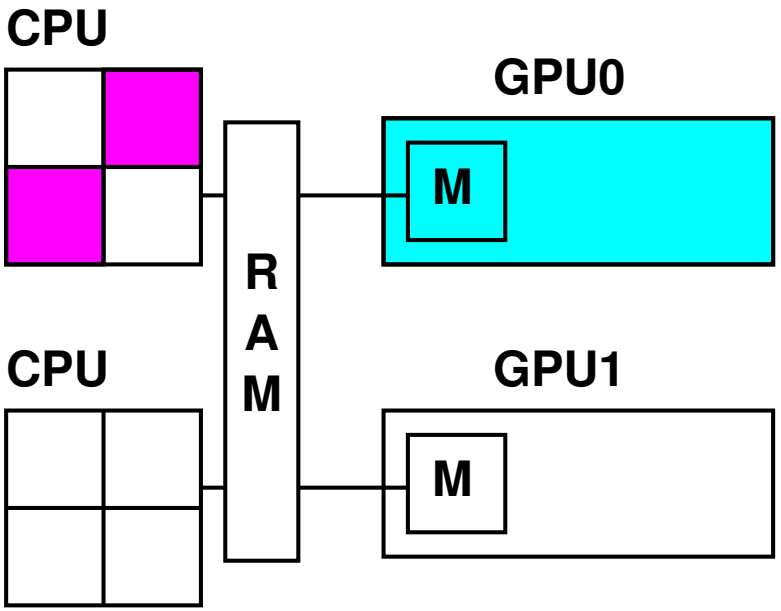


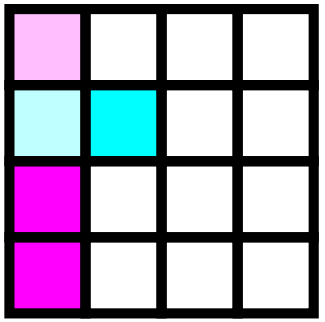


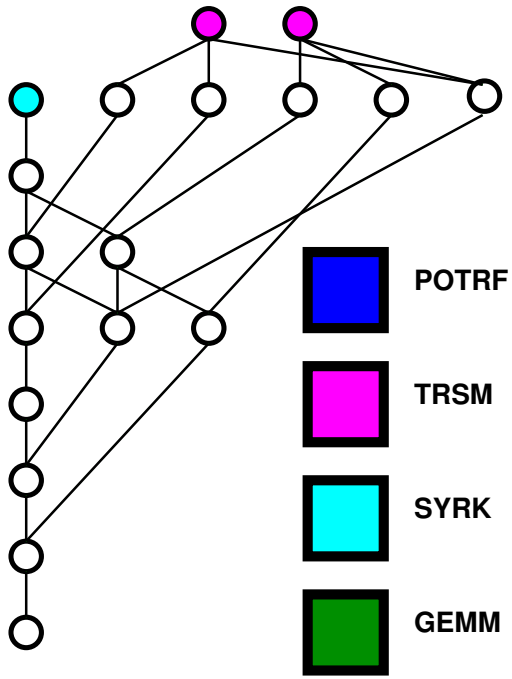
- Handles dependencies
- Handles scheduling

On an heterogeneous node

work on tiles → CPU + GPU kernels



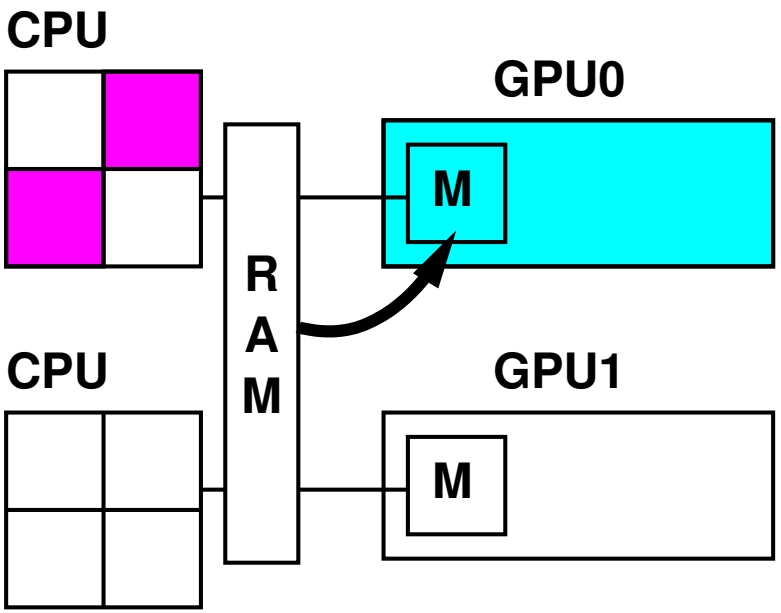


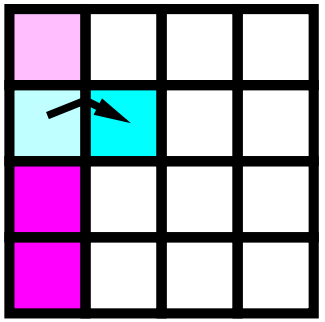


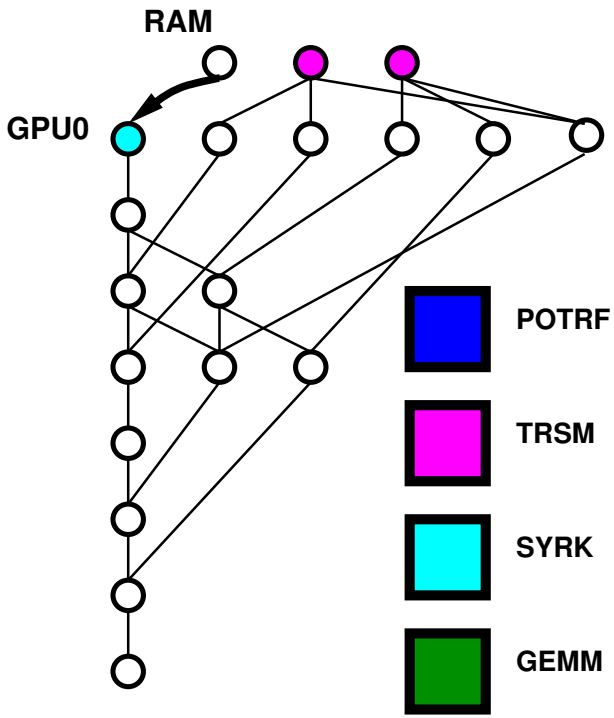
- Handles dependencies
- Handles scheduling

On an heterogeneous node

work on tiles → CPU + GPU kernels







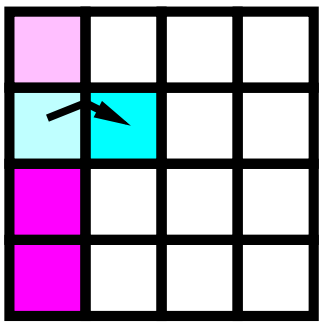
- Handles dependencies
- Handles scheduling
- Handles data consistency

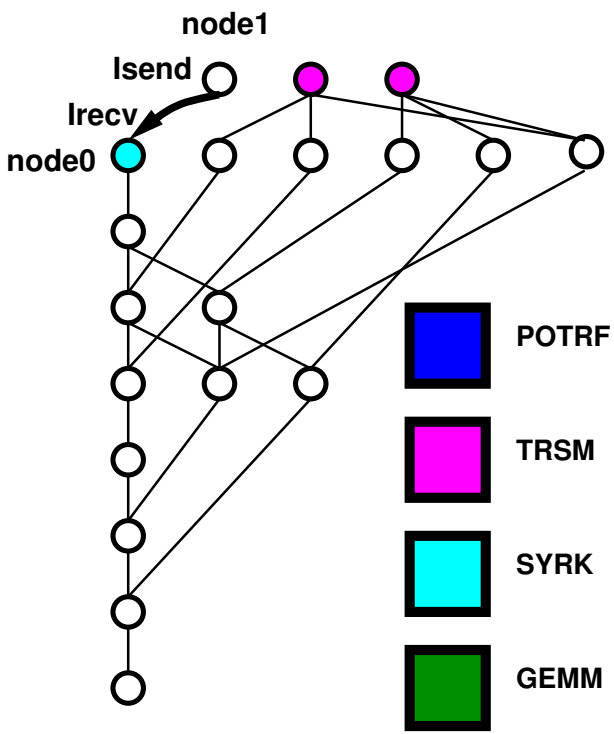
Address scalability

A new programming paradigm for clusters?

Questions

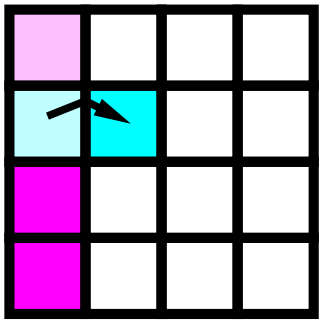
Existing methods





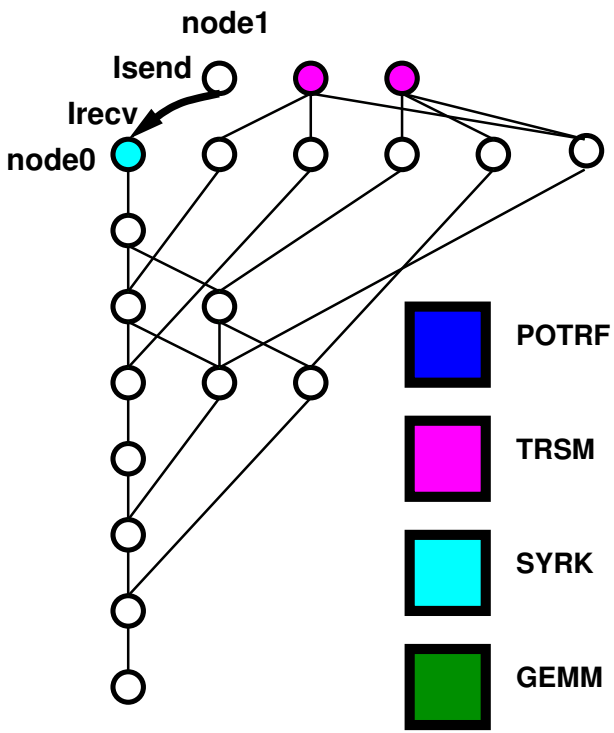
Address scalability

A new programming paradigm for clusters?



Questions

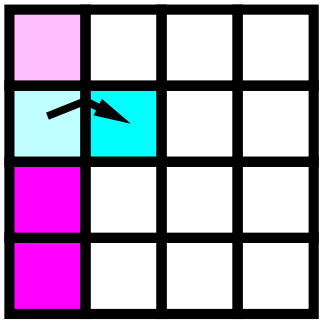
- How to establish the mapping?



Existing methods

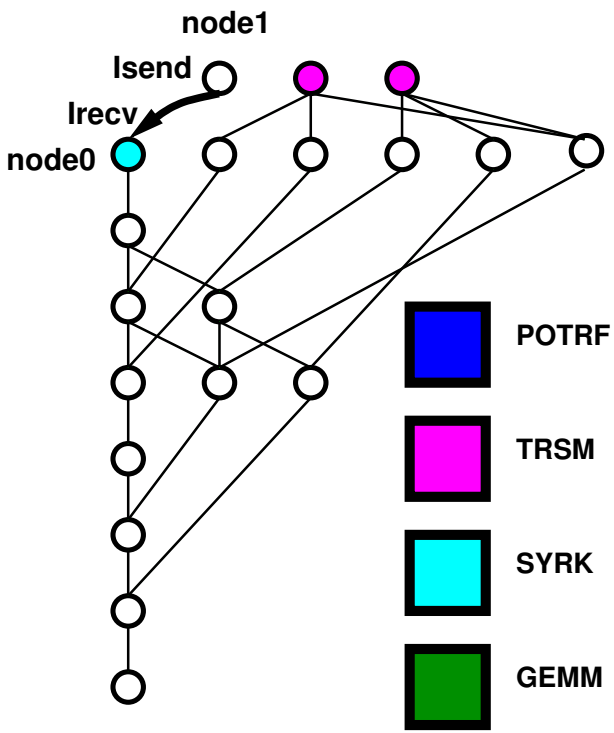
Address scalability

A new programming paradigm for clusters?



Questions

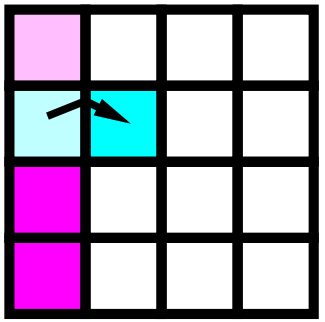
- How to establish the mapping?
- How to manage communications?



Existing methods

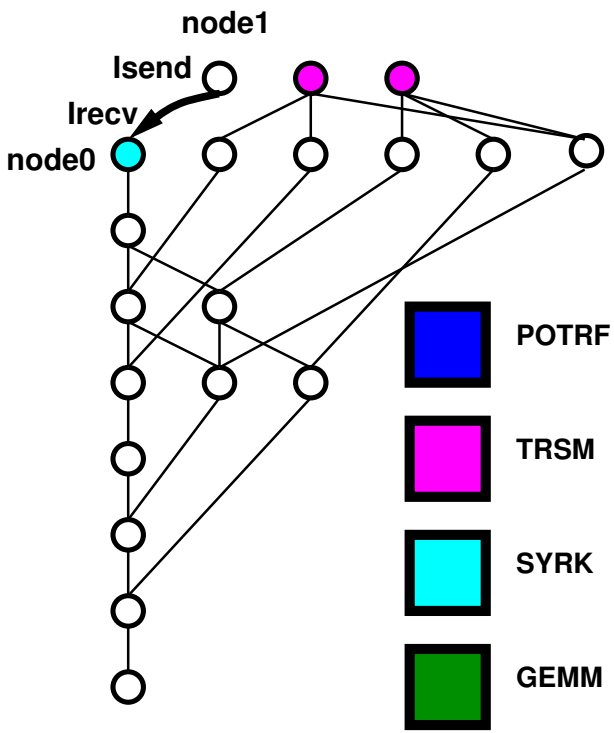
Address scalability

A new programming paradigm for clusters?



Questions

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

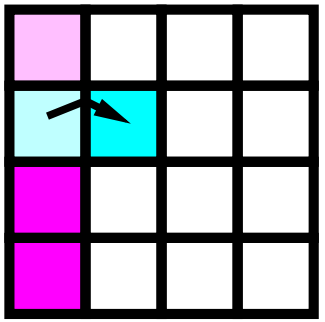


Existing methods

- Explicit MPI communications tasks

Address scalability

A new programming paradigm for clusters?

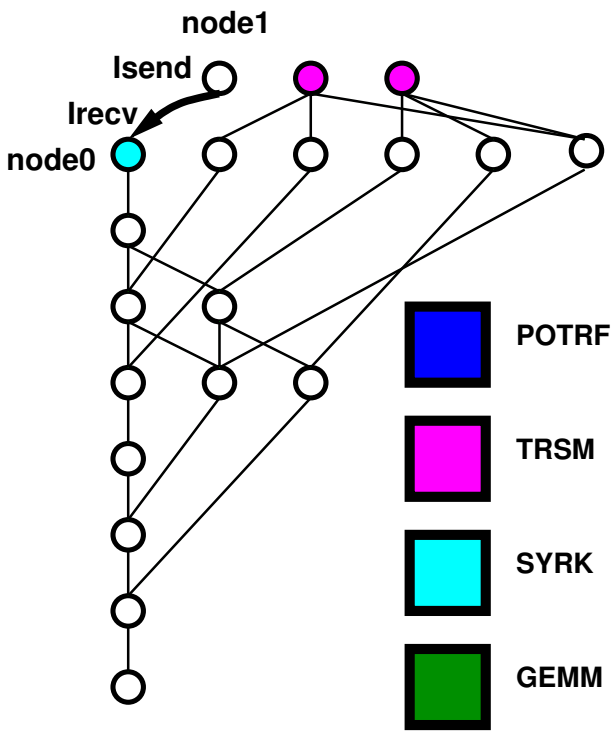


Questions

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

Existing methods

- Explicit MPI communications tasks
- PTG model (PaRSEC)



Address scalability

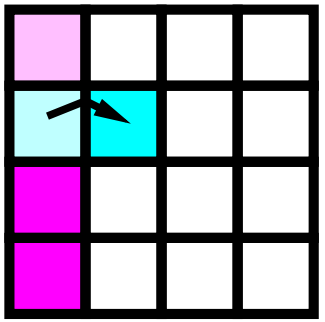
A new programming paradigm for clusters?

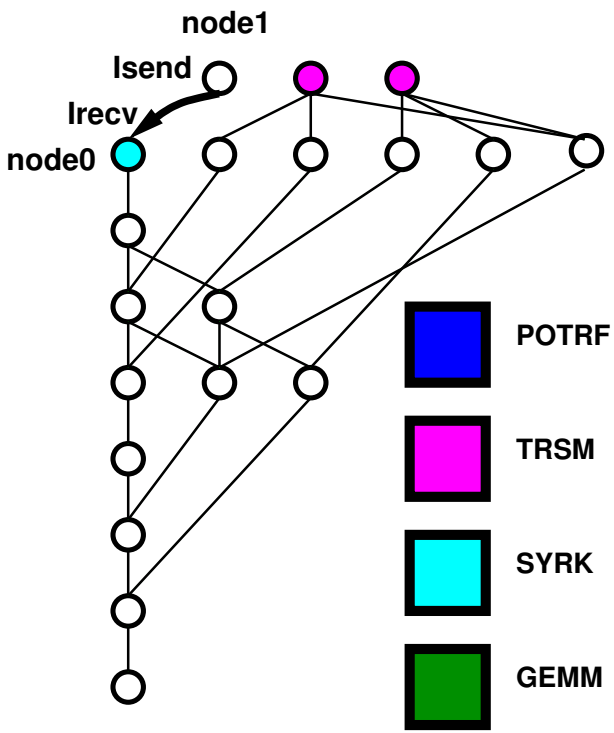
Questions

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

Existing methods

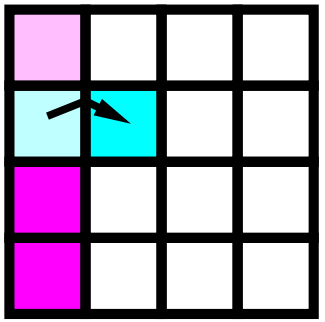
- Explicit MPI communications tasks
- PTG model (PaRSEC)
- STF model - Master/Slave (clusterSS)





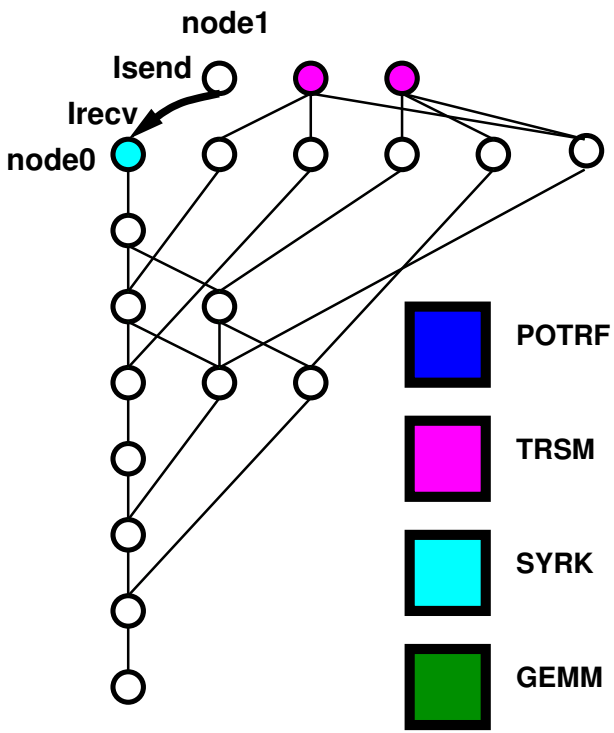
Address scalability

A new programming paradigm for clusters?



Questions

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

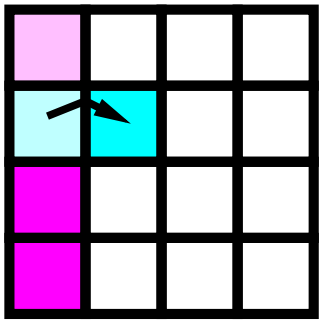


Existing methods

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

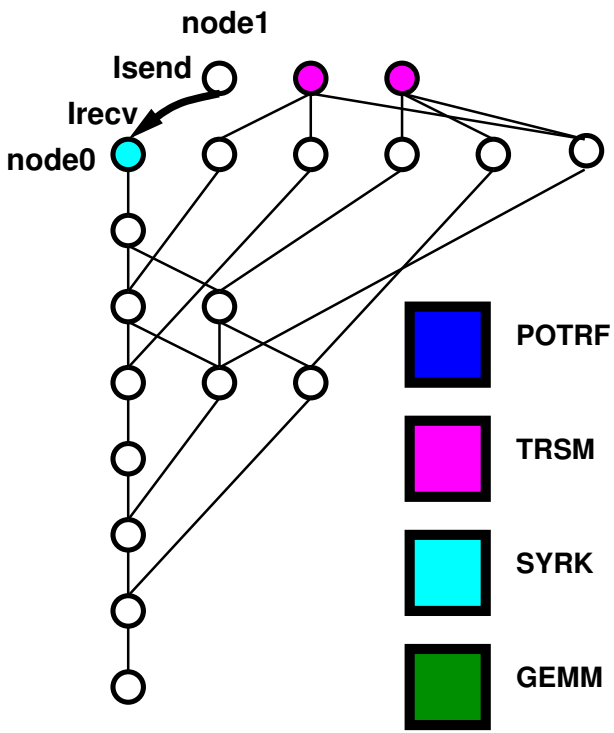
Address scalability

A new programming paradigm for clusters?



Questions

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



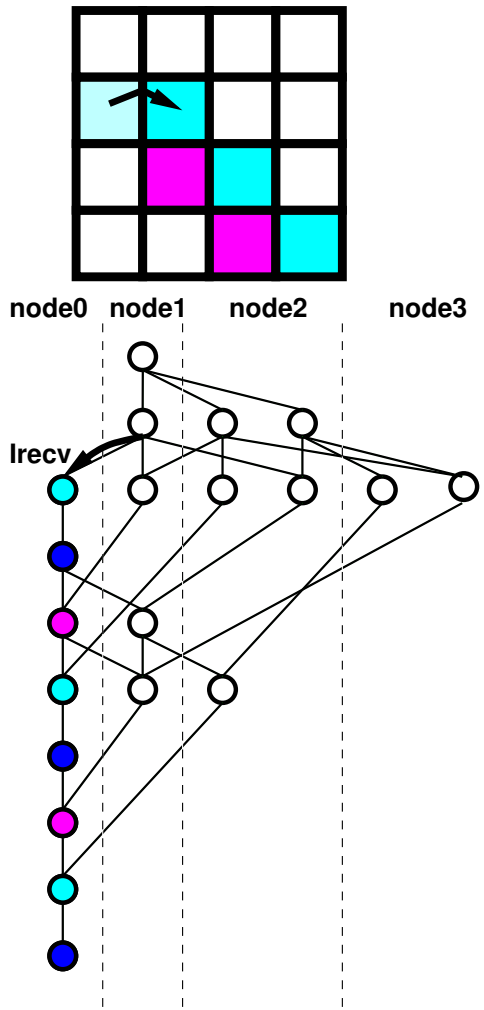
Existing methods

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

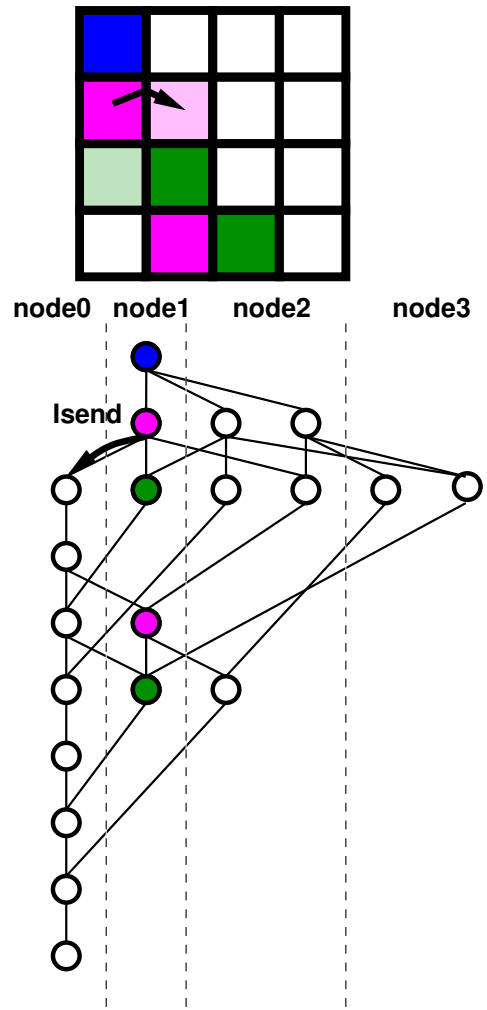
Data transfers between nodes

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



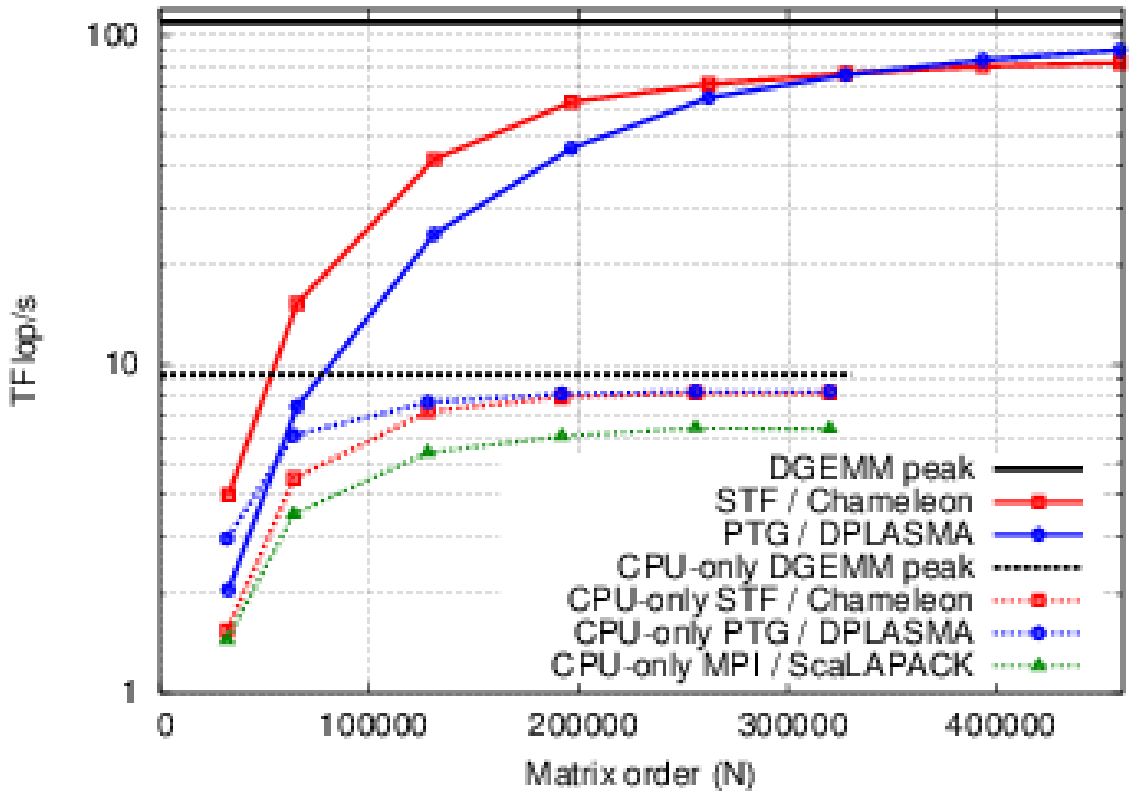
Node 0 execution



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)



Get Chameleon

Get Chameleon at

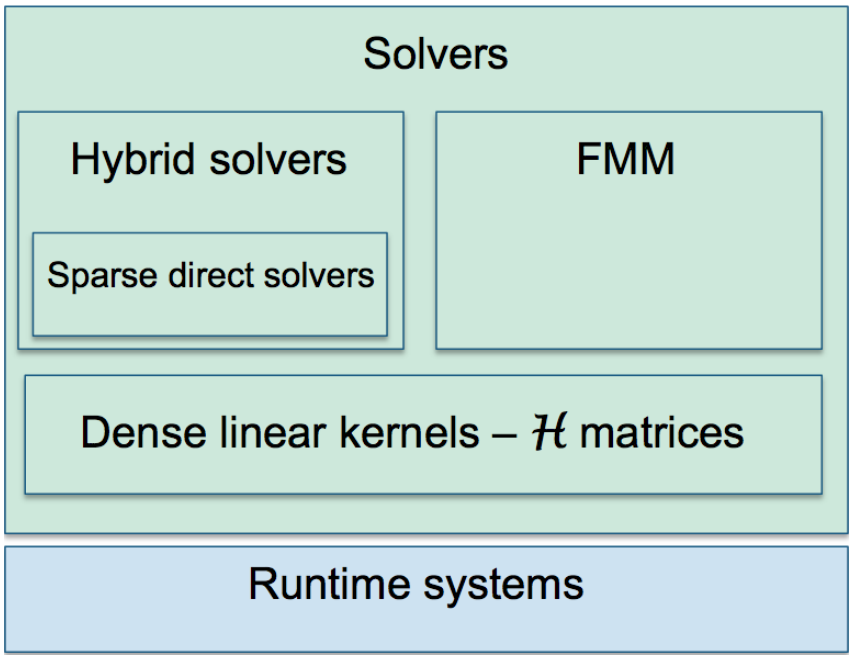
`https://project.inria.fr/chameleon/`

or install it using Spack
`http://morse.gfogre.inria.fr/spack`


```
git clone https://github.com/fpruvost/spack.git
cd spack
git checkout morse
spack install chameleon
```

Outline

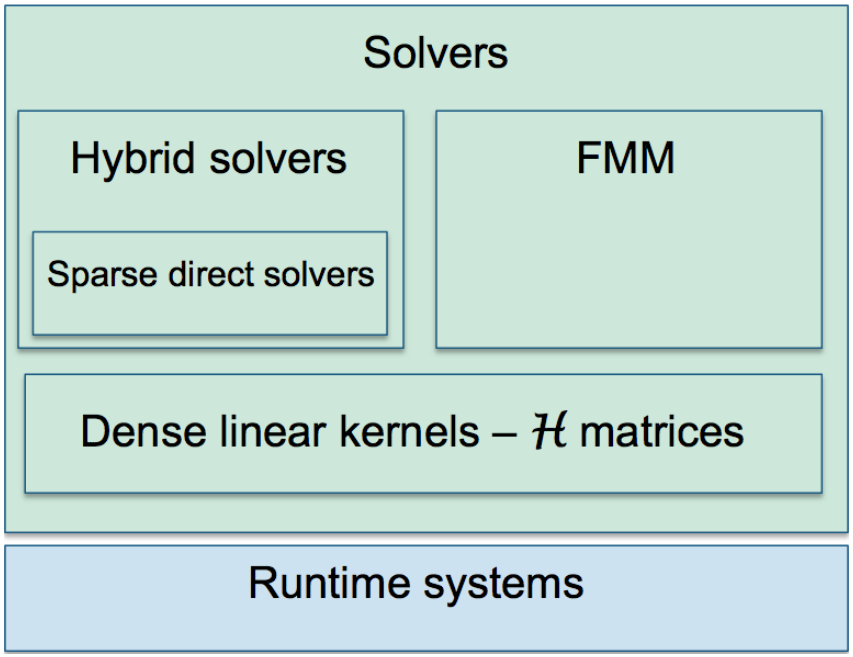
Solver stack



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)

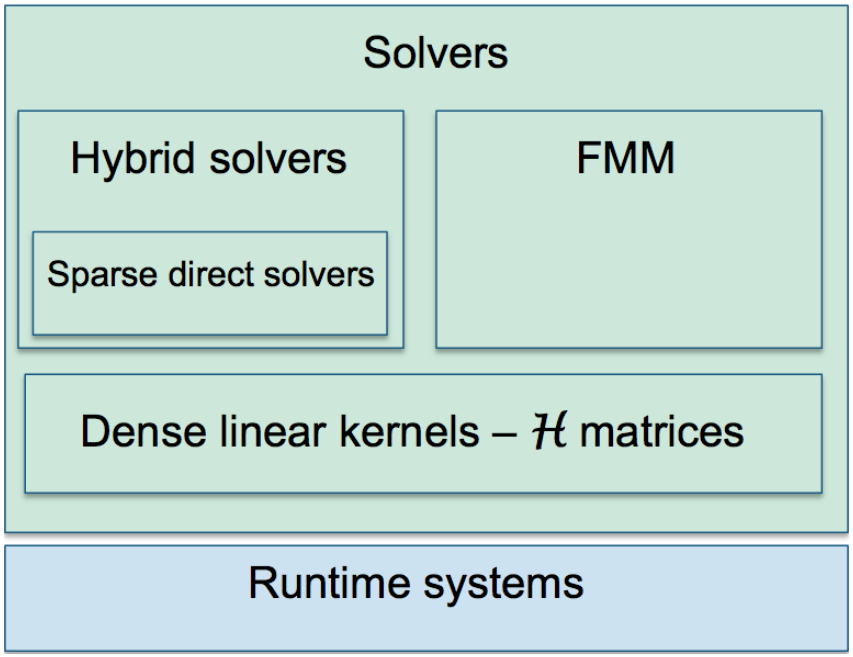
Solver stack



PaStiX: sparse direct 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

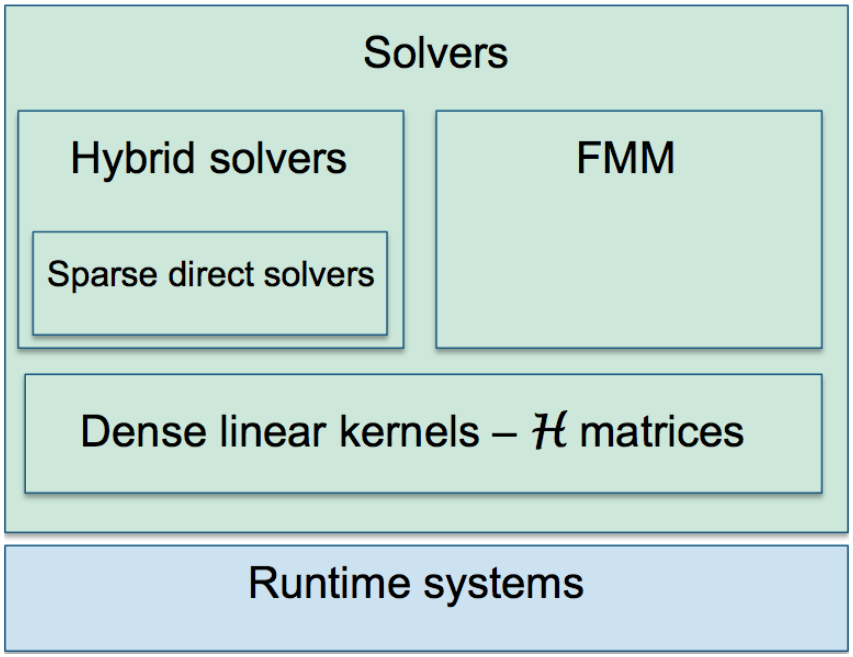
Solver stack



qr_mumps: sparse direct solver

- multifrontal **QR**, communication-avoiding kernel, memory-aware
- Original version: OpenMP
- Versions with runtimes: on top of StarPU (+ PaRSEC prototype)

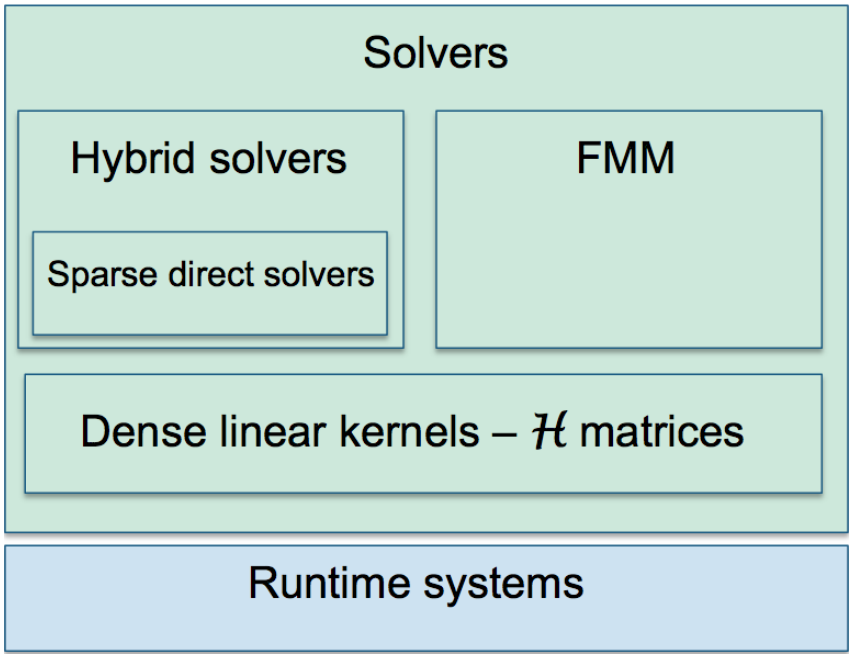
Solver stack



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
- Prototype task-based version
- Coarse grid correction (L. Poirel talk)

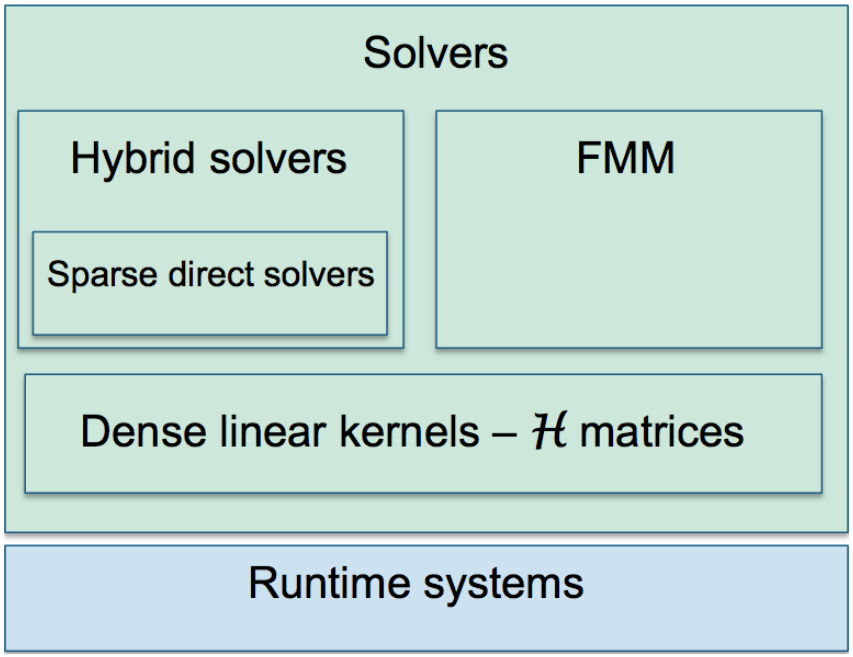
Solver stack



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)

Solver stack



hmat: hierarchical matrix (Airbus Group Innovation)

- H-matrices can be viewed as an algebraic version of FMM
- Runtimes version: StarPU, embedded $\rightarrow$ distributed, out-of-core

Outline

The SOLHAR project

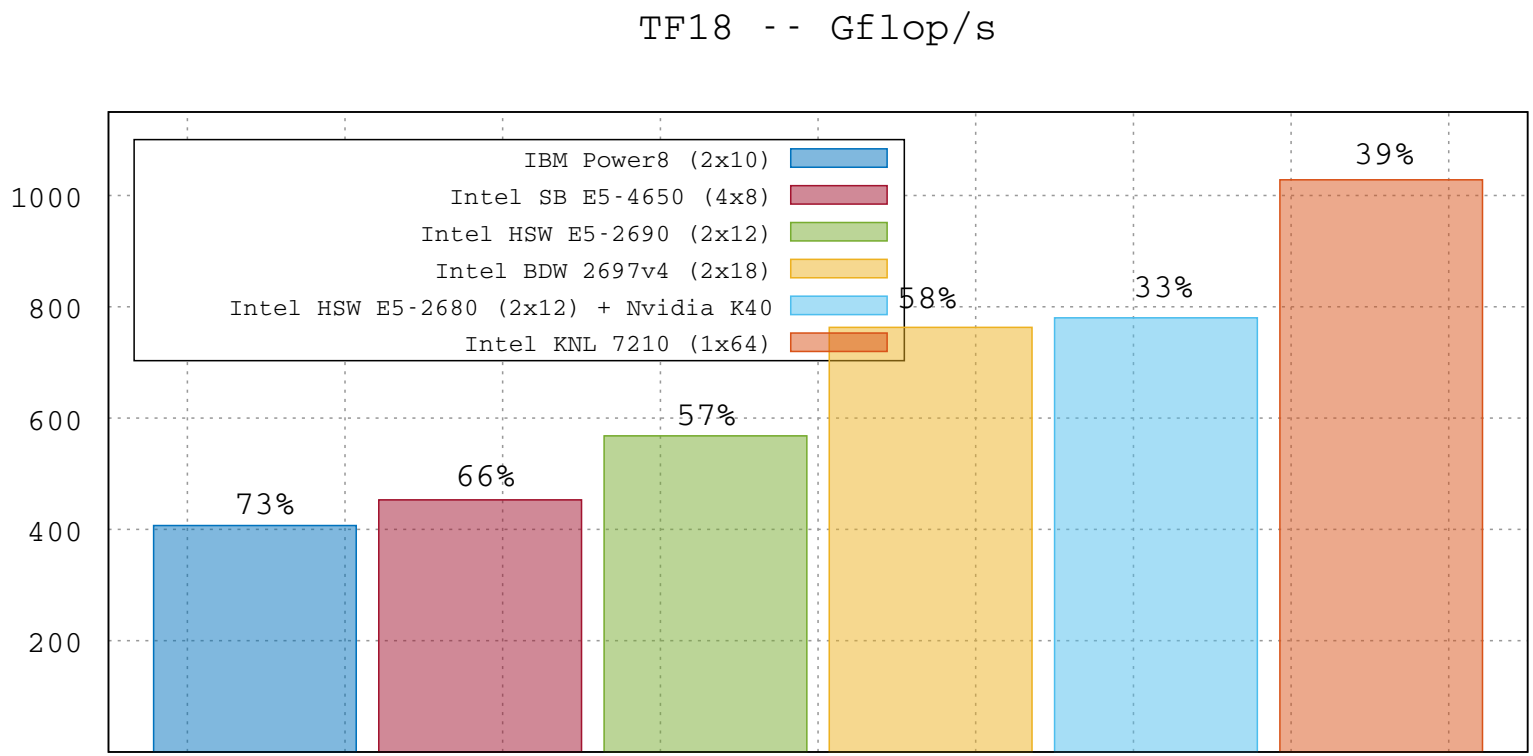
SOLvers for Heterogeneous Architectures using Runtimes (ANR-13-MONU0007)

- Sparse direct (`qr_mumps`, PaStiX) and dense (Chameleon) solvers
- Runtime system (StarPU)
- Scheduling
- Performance analysis

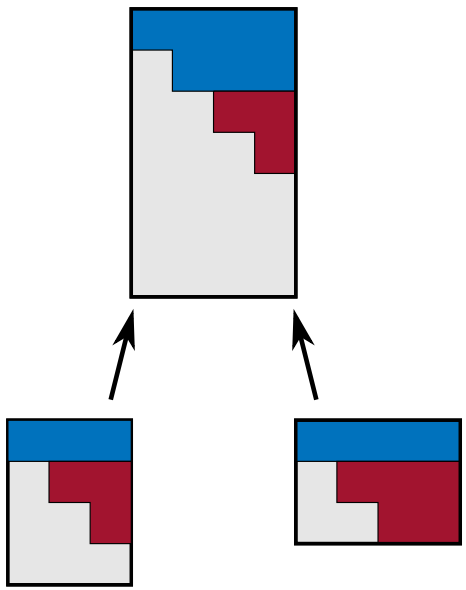
More at `http://solhar.gforge.inria.fr`

Performance highlight 1: qr_mumps

PhD. F. Lopez (N7-IRIT) - led by A. Buttari (CNRS/IRIT)



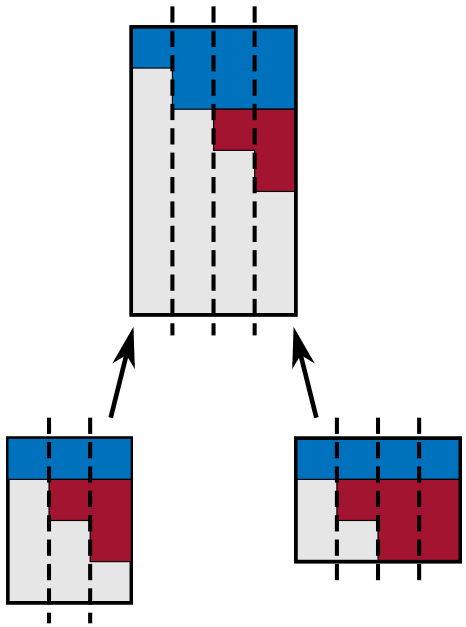
The task-based multifrontal QR factorization



```
forall fronts f in topological order

   ! compute front structure
   call activate(f)
   ! allocate and initialize front
   call init(f)

   ! front assembly
   forall children c of f
      call assemble(c, f)
      ! Deactivate child
      call deactivate(c)
   end do

   ! front factorization
   call factorize(f)
end do
```

Sequential multifrontal *QR* code

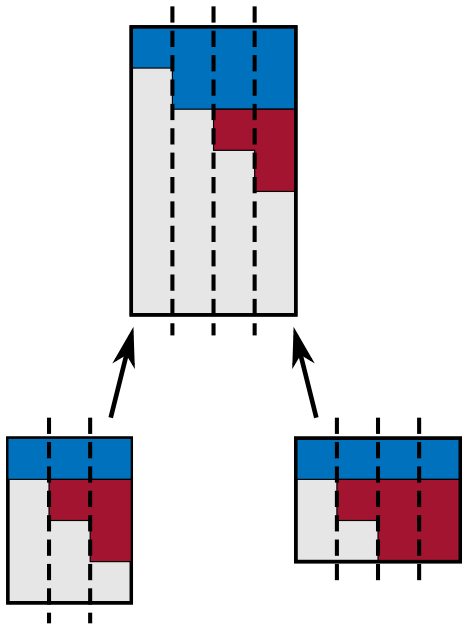
The task-based multifrontal QR factorization



```
forall fronts f in topological order
   ! allocate and initialize front
   call activate(f)

   ! front assembly
   forall children c of f
      call assemble(c, f)
      ! Deactivate child
      call deactivate(c)
   end do

   ! front factorization
   call factorize(f)
end do
```

Sequential multifrontal *QR* code with 1D block partitioning

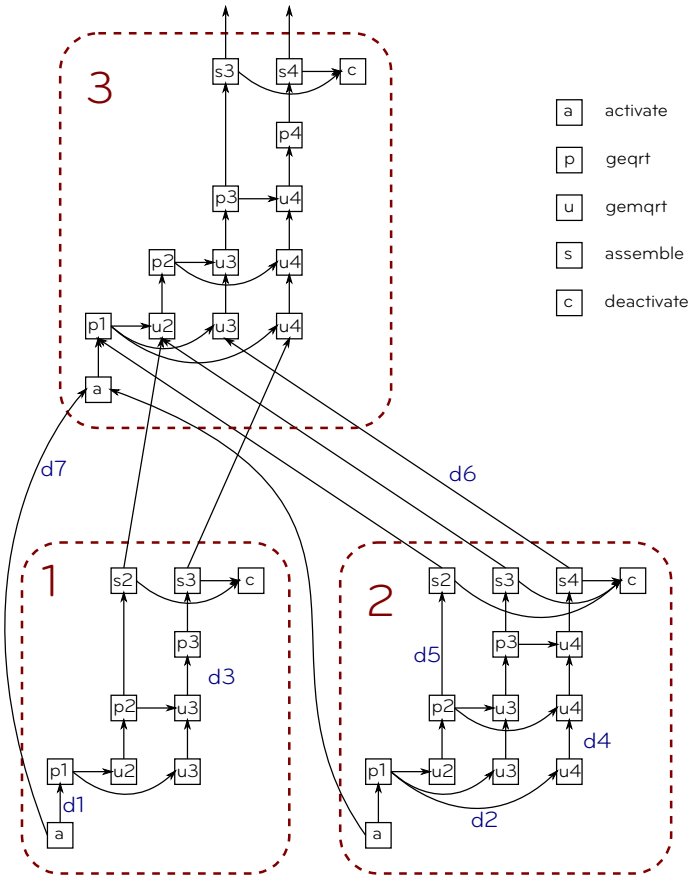
The task-based multifrontal QR factorization



```
forall fronts f in topological order
   ! allocate and initialize front
   call submit(activate, f:RW, children(f):R
     )

   ! front assembly
   forall children c of f
      call submit(assemble, c:R, f:RW|C) >\
     label{code:mf-stfcode1}>

      call submit(deactivate, c:RW)
   end do

   ! front factorization
   call submit(factorize, f:RW)  >\label{
     code:mf-stfcode2}>
end do

call wait_tasks_completion()
```

- STF multifrontal *QR* code with 1D block partitioning
- Elimination tree is transformed into a DAG

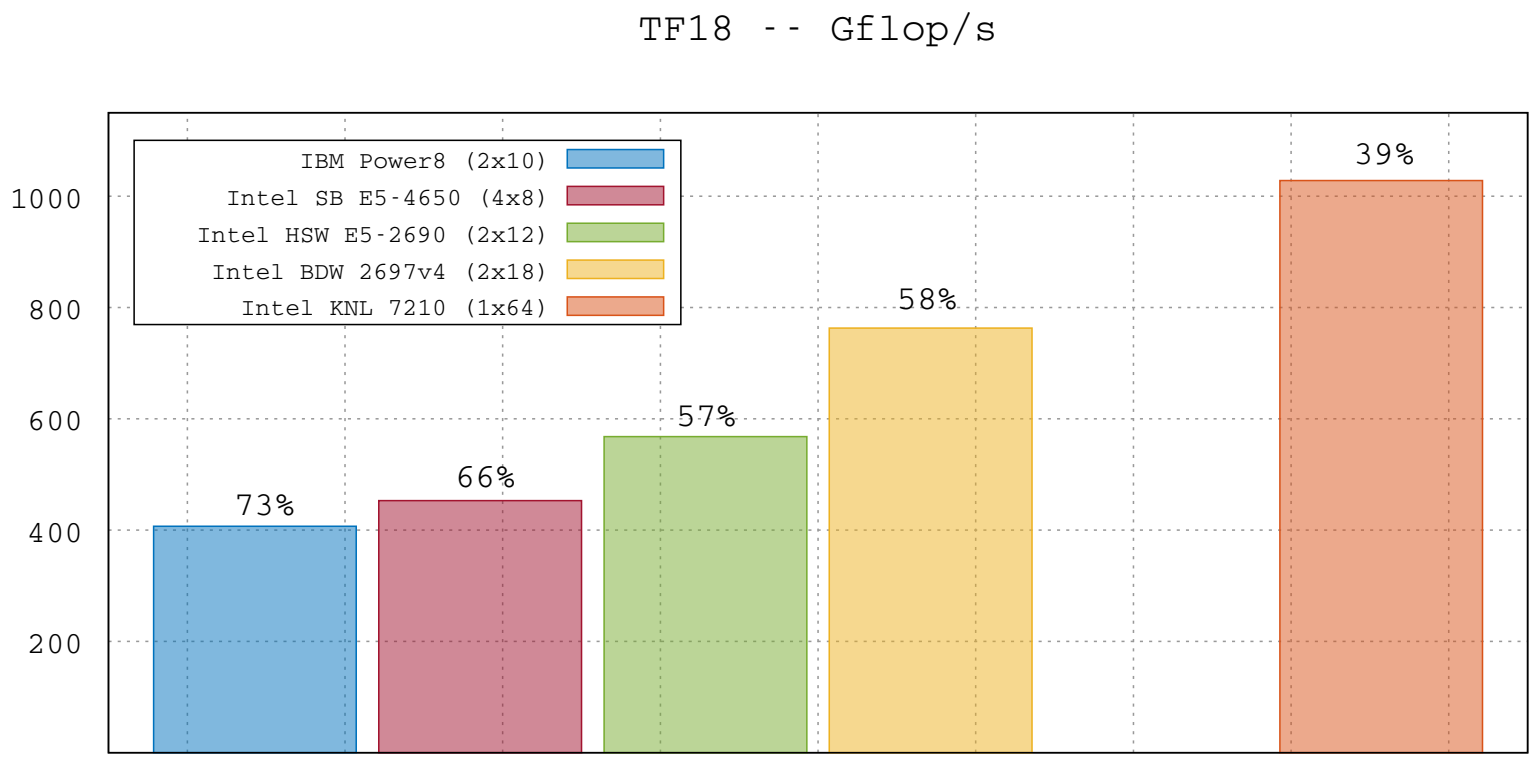
The task-based multifrontal QR factorization




```
forall fronts f in topological order
   ! allocate and initialize front
   call submit(activate, f:RW, children(f):R
     )

   ! front assembly
   forall children c of f
      call submit(assemble, c:R, f:RW|C) >\
     label{code:mf-stfcode1}>

      call submit(deactivate, c:RW)
   end do

   ! front factorization
   call submit(factorize, f:RW)  >\label{
     code:mf-stfcode2}>
end do

call wait_tasks_completion()
```

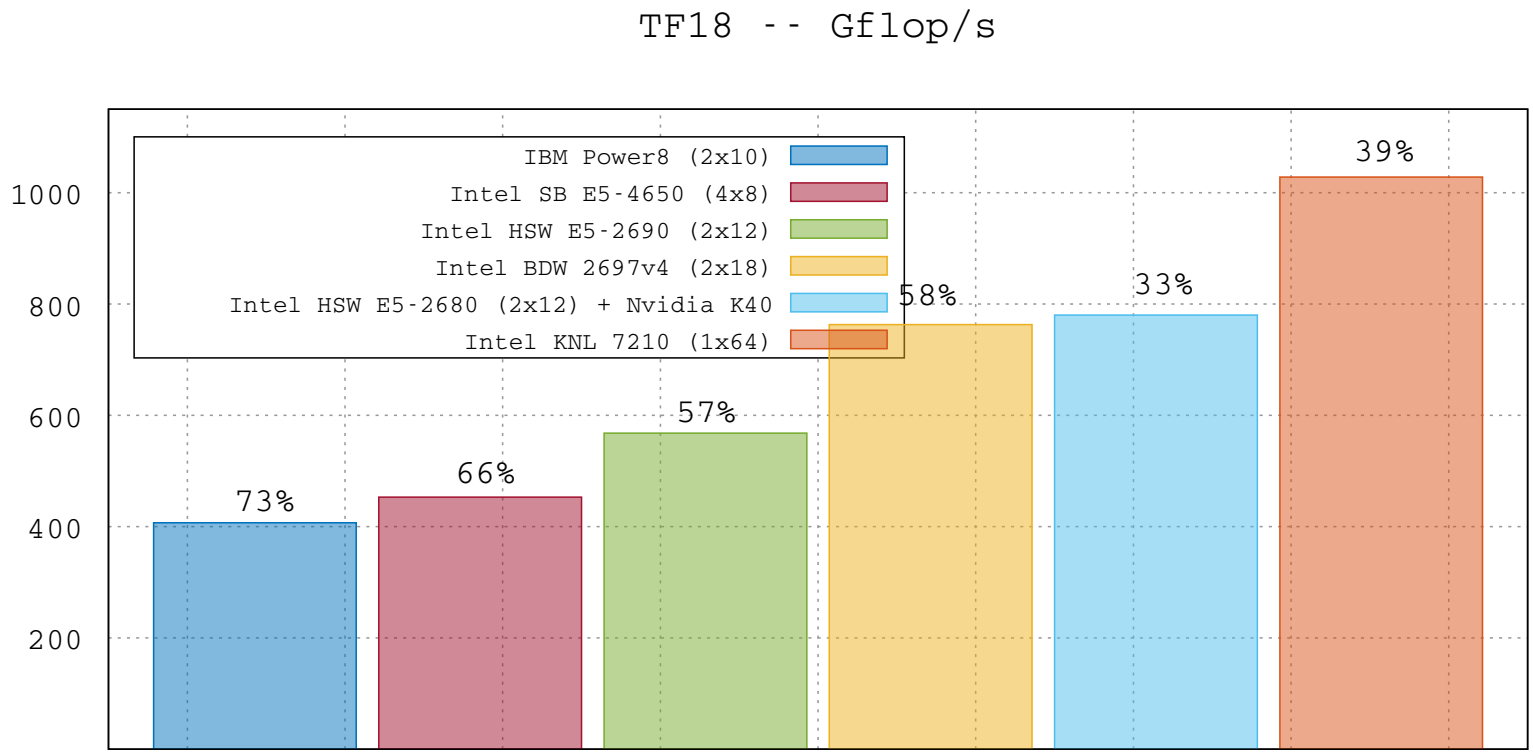
- Seamless exploitation of tree and node parallelism.
- Inter-level concurrency (father-child pipelining).

Performance



Credits: IBM, Intel, GENCI, CINES, IDRIS

Performance



Credits: IBM, Intel, GENCI, CINES, IDRIS

Get qr_mumps

Get qr_mumps at

http://buttari.perso.enseeiht.fr/qr_mumps

or install it using Spack
http://morse.gfogre.inria.fr/spack


```
% git clone https://github.com/fpruvost/spack.git
% cd spack
% git checkout morse
spack install qr_mumps
```

Get qr_mumps

Get qr_mumps at

http://buttari.perso.enseeiht.fr/qr_mumps

or install it using Spack
http://morse.gfogre.inria.fr/spack

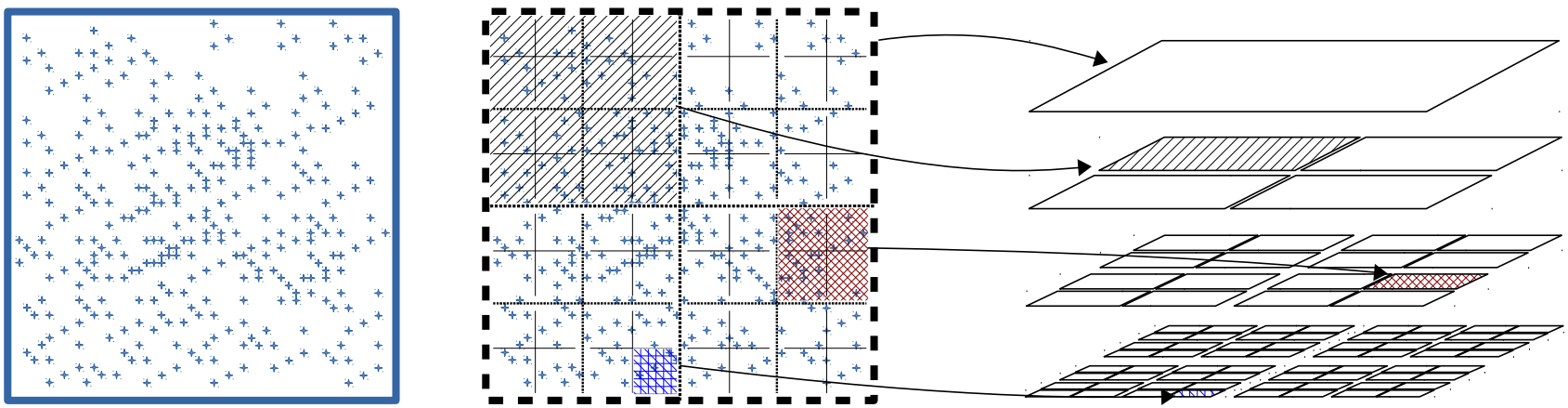

```
% git clone https://github.com/fpruvost/spack.git
% cd spack
% git checkout morse
spack install qr_mumps
```

See Alfredo Buttari's talk @ PMAA'16
(MS - Task-based scientific libraries on top of runtime systems)

Outline

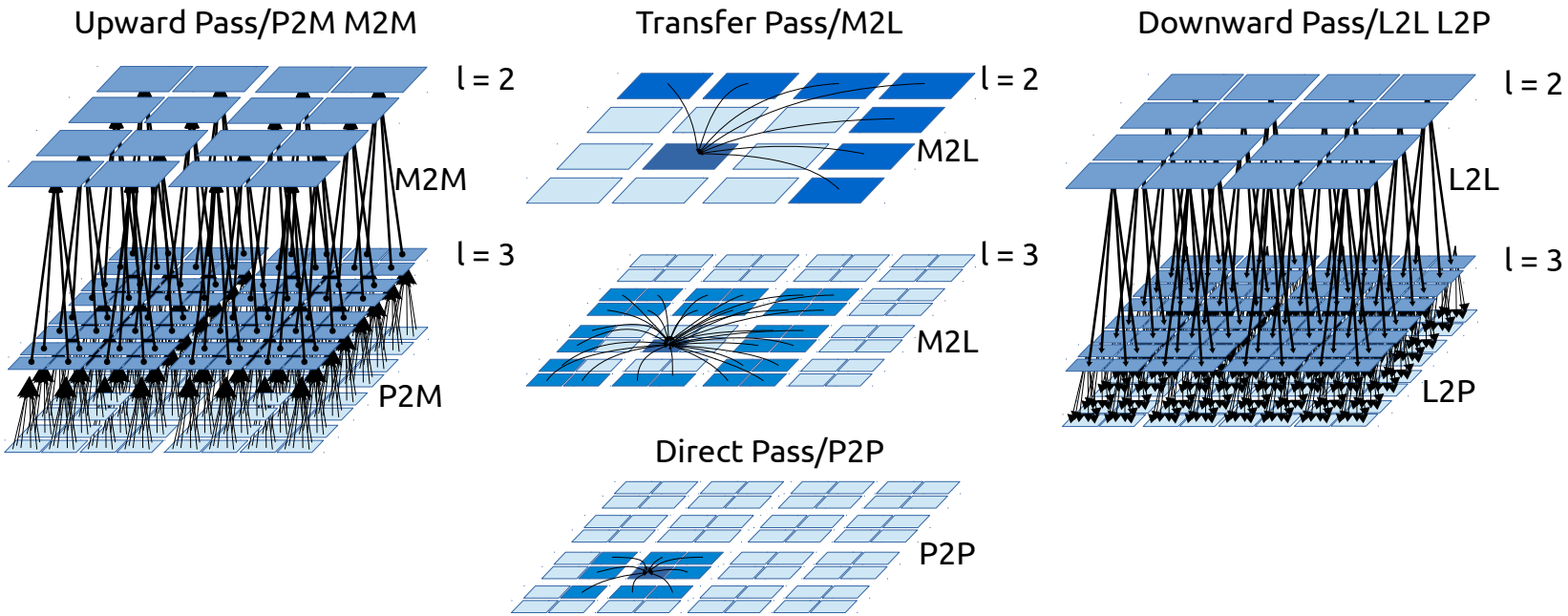
Performance highlight 2: ScalFMM

PhD. B. Bramas (Inria HiePACS / Airbus) - in collaboration with Storm



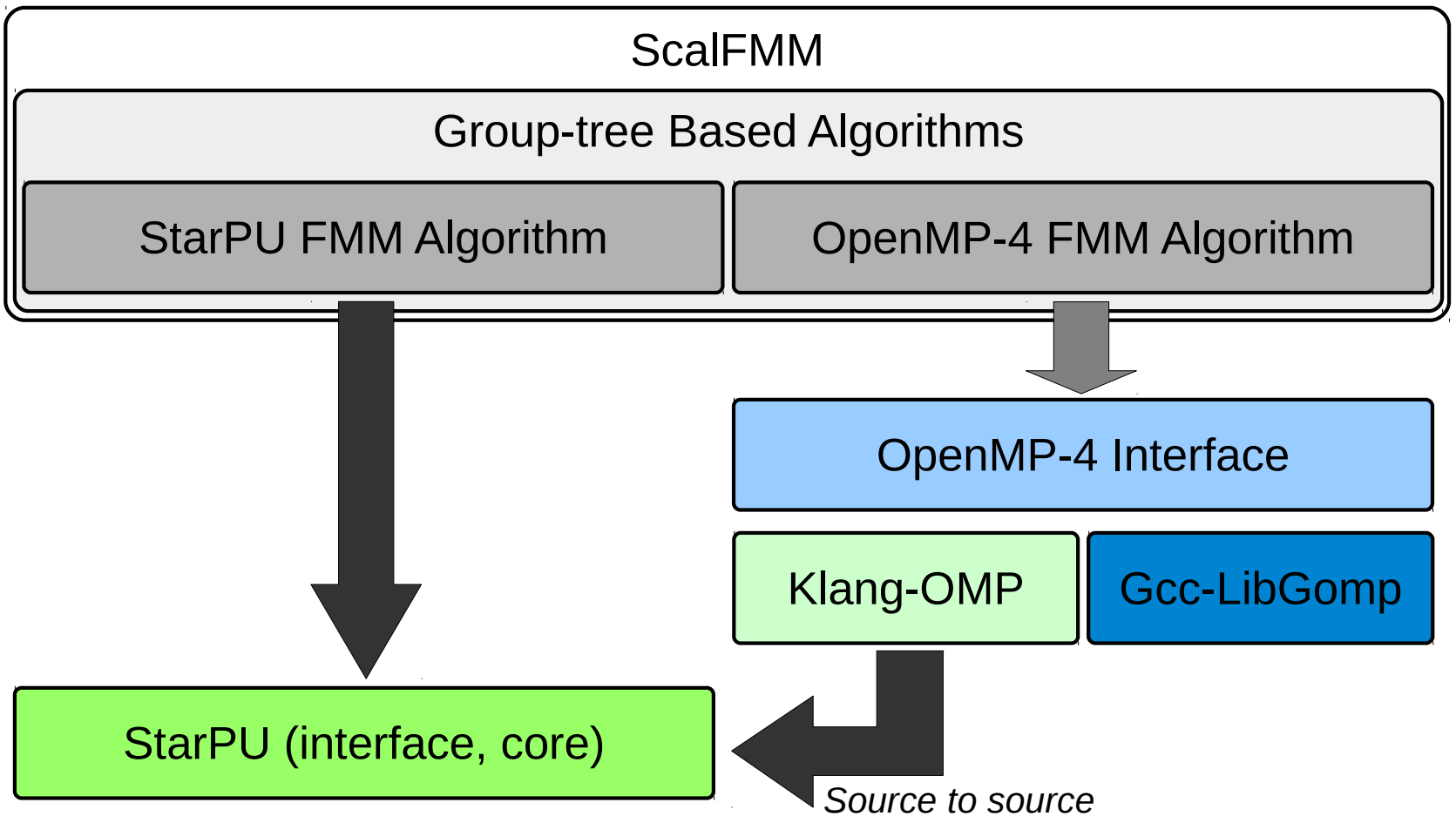
Performance highlight 2: ScalFMM

PhD. B. Bramas (Inria HiePACS / Airbus) - in collaboration with Storm



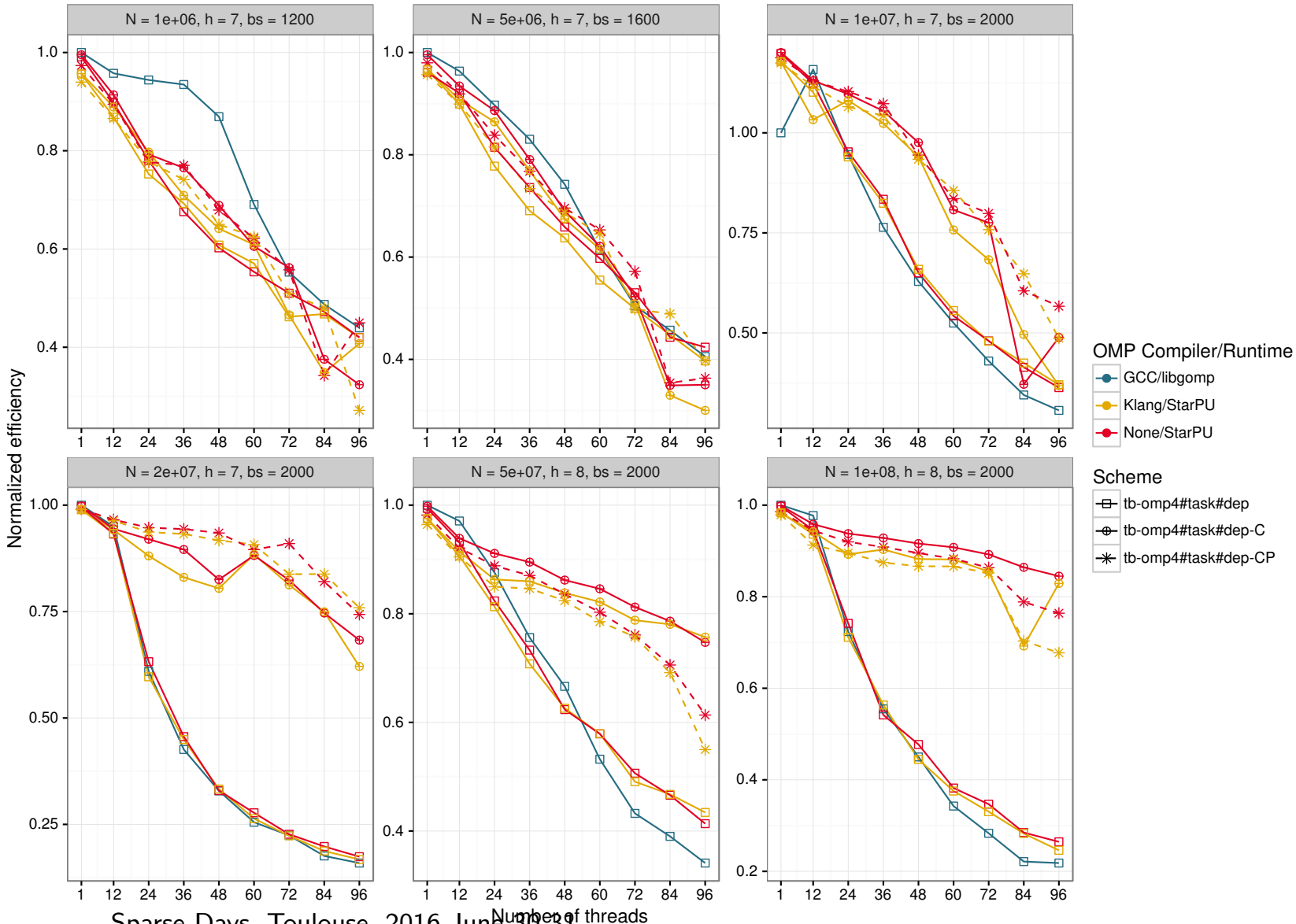
Performance highlight 2: ScalFMM

PhD. B. Bramas (Inria HiePACS / Airbus) - in collaboration with Storm



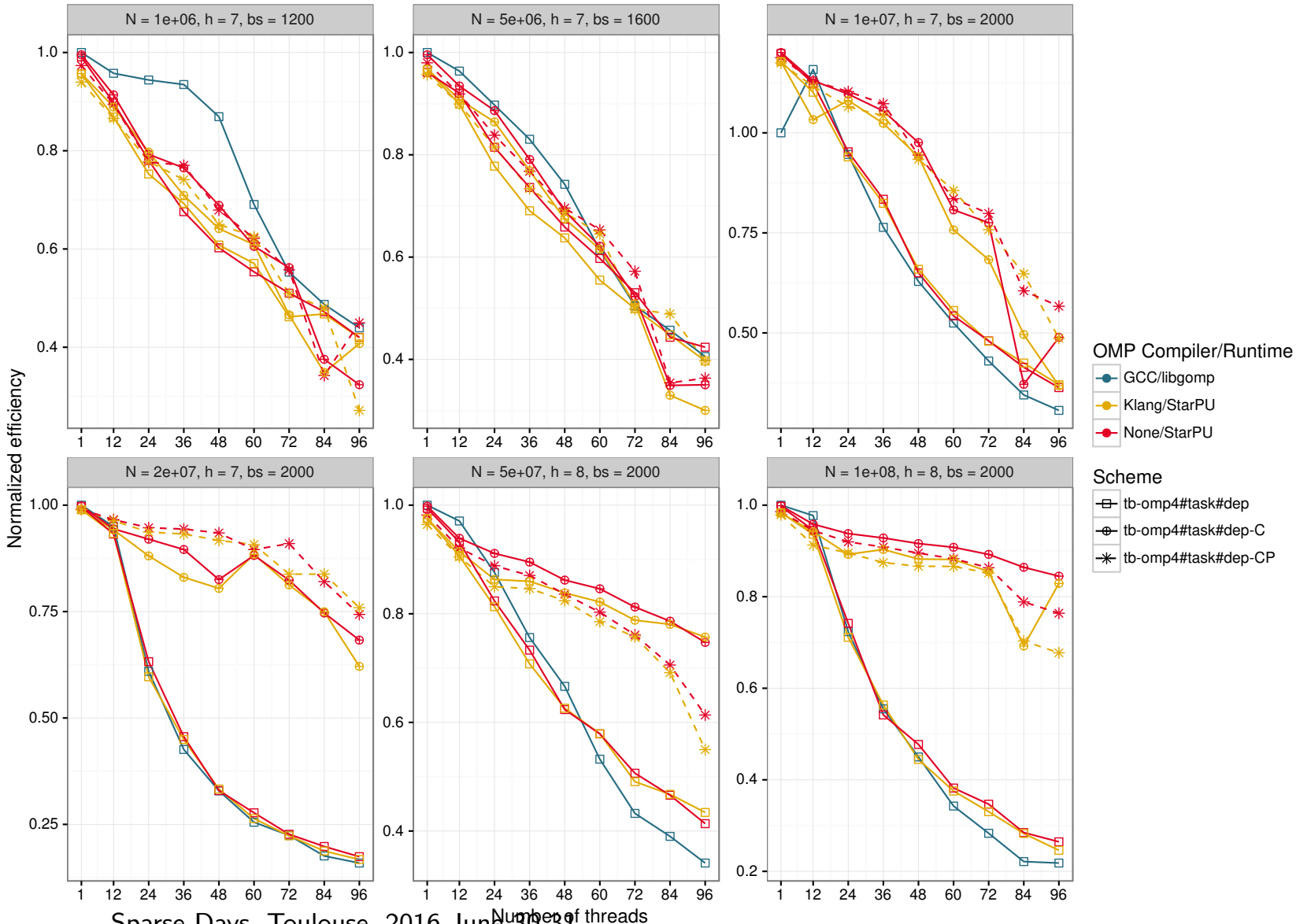
Performance highlight 2: ScalFMM

PhD. B. Bramas (Inria HiePACS / Airbus) - in collaboration with Storm



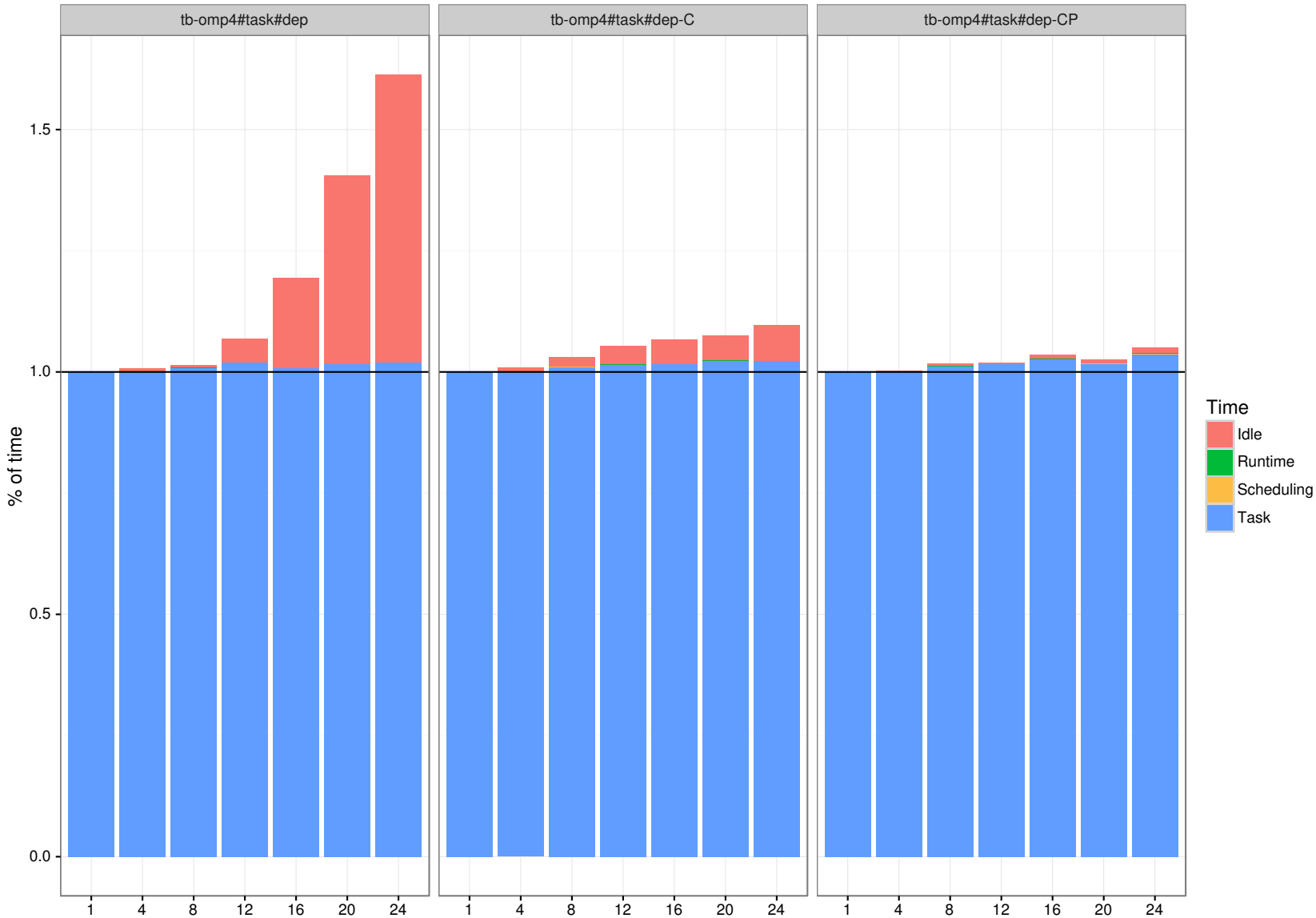
Performance highlight 2: ScalFMM

PhD. B. Bramas (Inria HiePACS / Airbus) - in collaboration with Storm



Performance highlight 2: ScalFMM

PhD. B. Bramas (Inria HiePACS / Airbus) - in collaboration with Storm



Get ScalFMM

Get ScalFMM at

`http://scalfmm-public.gforge.inria.fr`

or install it using Spack
`http://morse.gfogre.inria.fr/spack`


```
% git clone https://github.com/fpruvost/spack.git
% cd spack
% git checkout morse
spack install scalfmm
```

Outline

Conclusion and perspectives

Beyond the solver stack:

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

On-going work

- Numerical algorithms
- Runtime systems
- Automatic parallelization
- Scheduling
- Communication layers
- Simulation, verification and reproducibility
- Resilience
- Autotuning