Solving large linear systems

Parallel solvers based on runtime systems

Sommaire

01

Context

Inria Team HiePACS

High-End Parallel Algorithms for Challenging Numerical Simulations

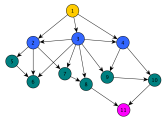
Linear algebra

$$AX = B$$

Sequential-Task-Flow

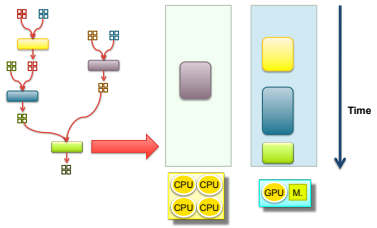
```
for (j = 0; j < N; j++)
Task (A[j]);
```

Direct Acyclic Graph



→

Runtime systems



→

Heterogeneous platforms

Main goal : performances, scaling

Chameleon: dense matrix solver

- BLAS: basic scalar, vector, matrix operations

$$\alpha \begin{pmatrix} \cdot \\ \cdot \end{pmatrix}, \quad \begin{pmatrix} \cdot \\ \cdot \end{pmatrix} + \begin{pmatrix} \cdot \\ \cdot \end{pmatrix}, \quad \begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot \\ \cdot \end{pmatrix}, \quad \begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \end{pmatrix}$$

- LAPACK: linear systems $\mathbf{AX} = \mathbf{B}$, square-roots, eigen val.

PaStiX: sparse matrix direct solver

- linear systems $\mathbf{AX} = \mathbf{B}$, factorization $\mathbf{LL^T}, \mathbf{LDL^T}, \mathbf{LU}$

MaPHyS: sparse matrix hybrid solver

- linear systems $\mathbf{AX} = \mathbf{B}$, **CG/GMRES + Preconditioner**
- Preconditioner = direct solver $\rightarrow$ MUMPS or PaStiX

Runtimes supported by Chameleon & PaStiX

StarPU

- Inria Storm Team
- **Dynamic Task Discovery**
- C/C++, Fortran
- MPI + Thread + Cuda
- Multiple scheduling strategies: Minimum Completion Time, Local Work Stealing, user defined...
- Computes cost models on the fly

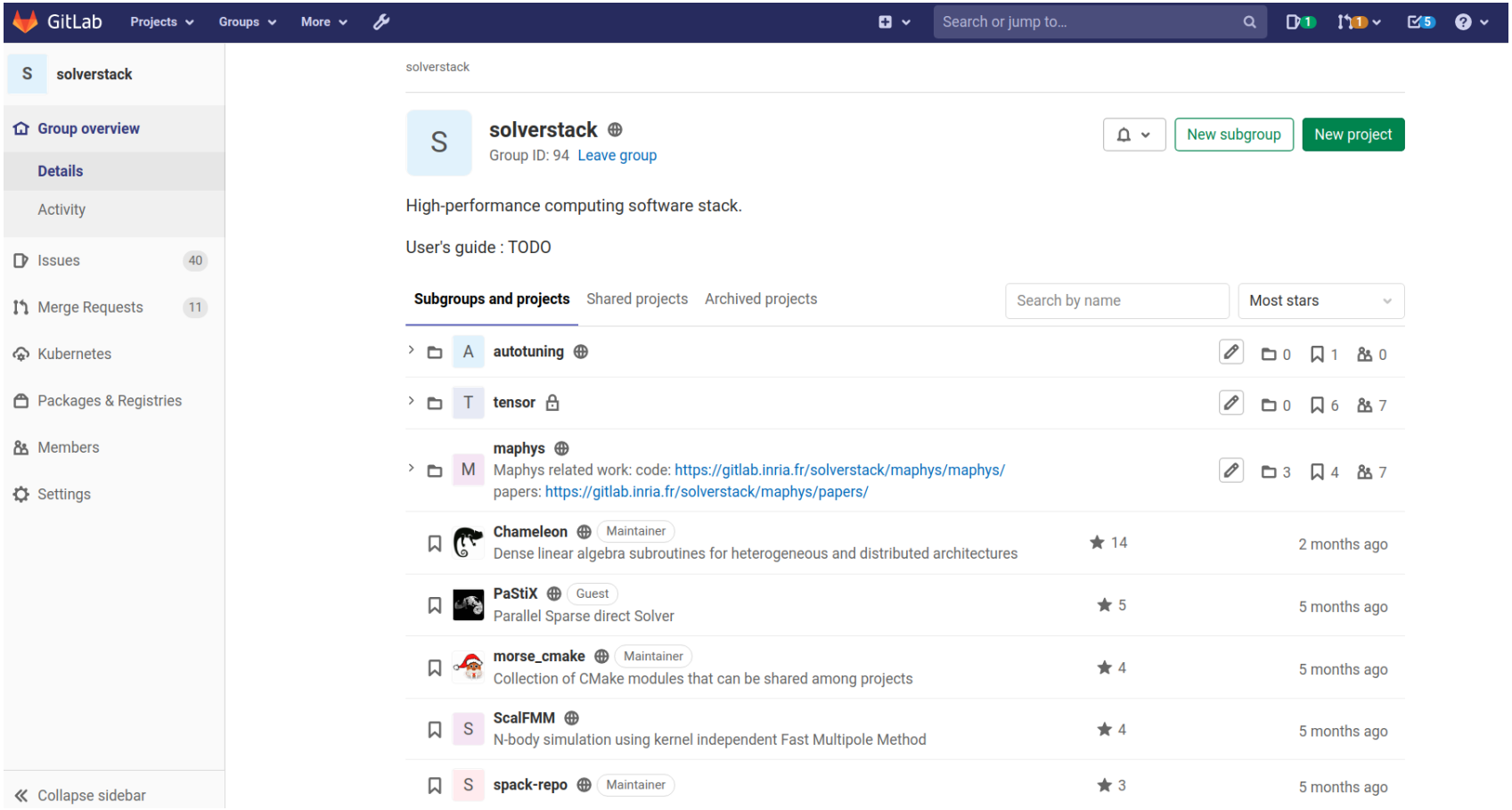
PaRSEC

- ICL – University of Tennessee, Knoxville
- **Parameterized Task Graph**
- C/C++, Fortran
- MPI + Thread + Cuda
- Scheduling strategy based on static performance model

Advantages of using a task-based runtime

- Several computing kernels can be associated with the task
  - C, OpenCL, NVIDIA CUDA
- Execute the task graph on the available resources
- Address the whole computing units and the whole potential parallelisms
- Insulate the algorithm from the architecture and data distribution
- Automatic handling of data transfers
- Finer parallelism handling

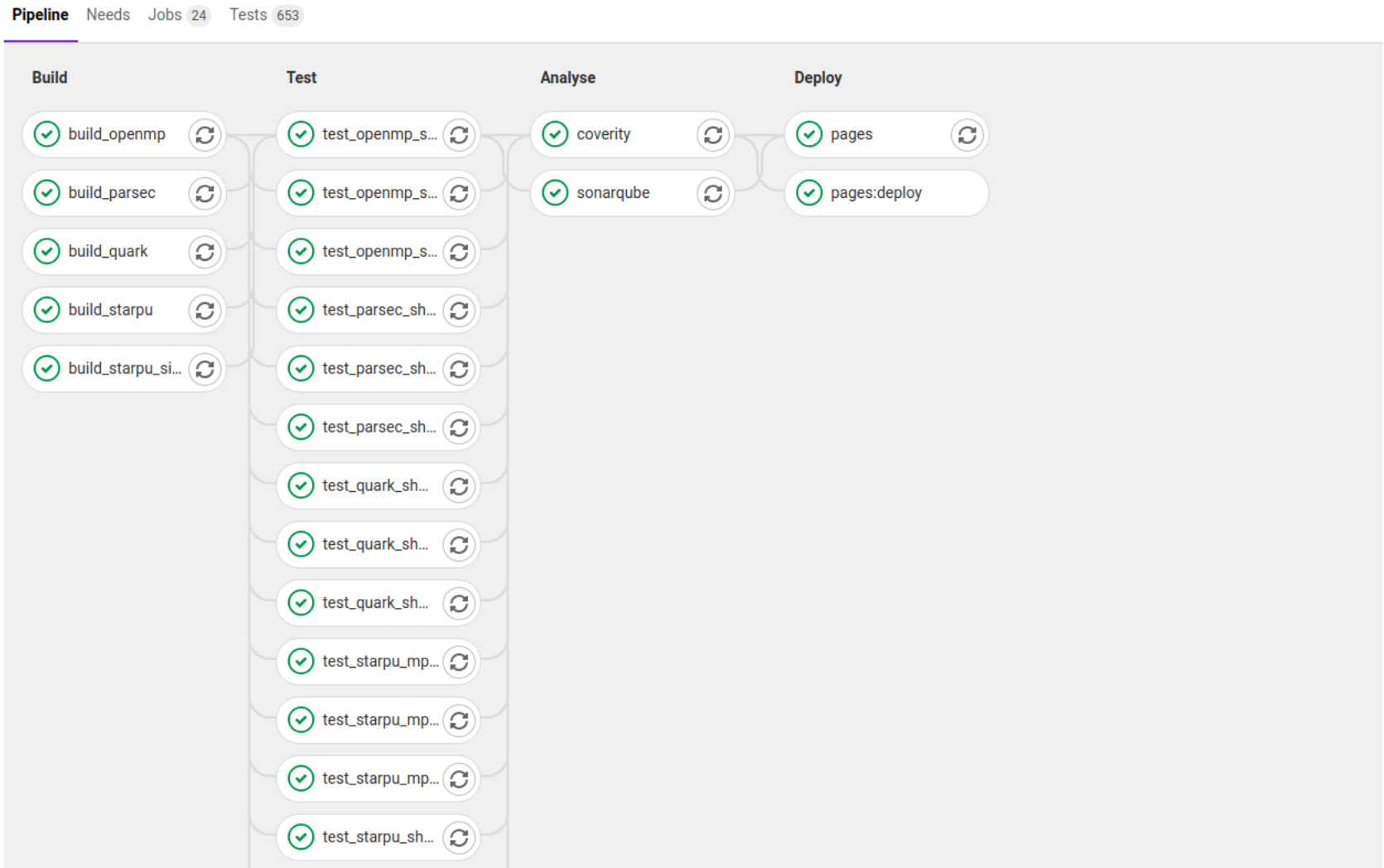
Software development quality



Opensource - Hosted at

https://gitlab.inria.fr/solverstack

Software development quality



Continuous integration: build, tests, coverage, etc

Software development quality

Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects and files... PF

hiepacs:pastix:gitlab:master master

January 25, 2021, 5:21 PM Version master

Overview Issues Measures Code Activity Administration More

Quality Gate Passed

Reliability Measures

New code: since previous version
started 2 years ago

0 A Bugs
started 2 years ago

0 A New Bugs

Security Measures

0 A Vulnerabilities

0 Security Hotspots

0 A New Vulnerabilities

0 New Security Hotspots

Maintainability Measures

28d A Debt

1.2k Code Smells

3d A New Debt

127 New Code Smells

Coverage Measures

62.4% Coverage

74.5% Coverage on 1.8k New Lines to Cover

About This Project
Parallel Sparse direct Solver

No tags

M 21k Lines of Code C (Comm... 21k

Project Activity

January 25, 2021
master

January 13, 2021
Project Analyzed

January 7, 2021
Quality Profile: **Changes in 'hiepacs:pastix' (C (Community))**

Show More

Quality Gate
(Default) Inria

Quality Profiles
(C (Community)) hiepacs:pastix

Source code checkers: SonarQube, clang-sa/tidy, cppcheck

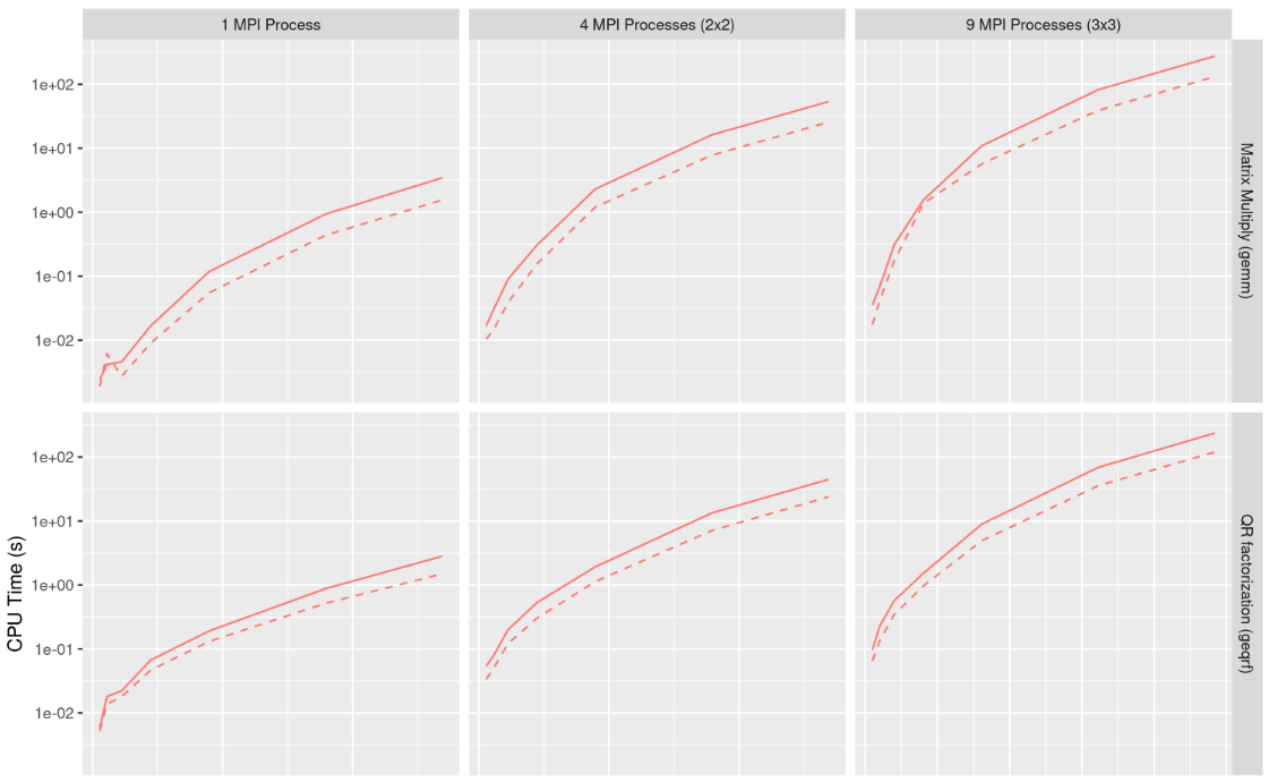
Software development quality

Created: 2021-01-17 Sun 11:07

7.1 bora (36 CPUs) nodes

- nmpi = **1**, **4**, **9**
- 2D block cyclic parameters : PxQ = 1x1, 2x2 and 3x3
- Number of threads (t) = **34**, one CPU being dedicated for the scheduler and one other for MPI communications
- Number of GPUs = **0**
- Tile Size (b) = **280**

7.1.1 CPU times



Performances tests once a week on www.plafrim.fr

CMake (ex: Ubuntu 20.04)

```
sudo apt install git python-numpy cmake build-essential gfortran \
                 pkg-config libmkl-dev libscotch-dev libstarpu-dev
git clone --recursive https://gitlab.inria.fr/solverstack/pastix.git
cd pastix && mkdir build && cd build
cmake .. -DPASTIX_WITH_STARPU=[ON|OFF] -DPASTIX_INT64=OFF && make
```

Spack (Linux, MacOS)

```
spack install chameleon pastix maphys
```

https://gitlab.inria.fr/solverstack/spack-repo

GNU Guix (Linux)

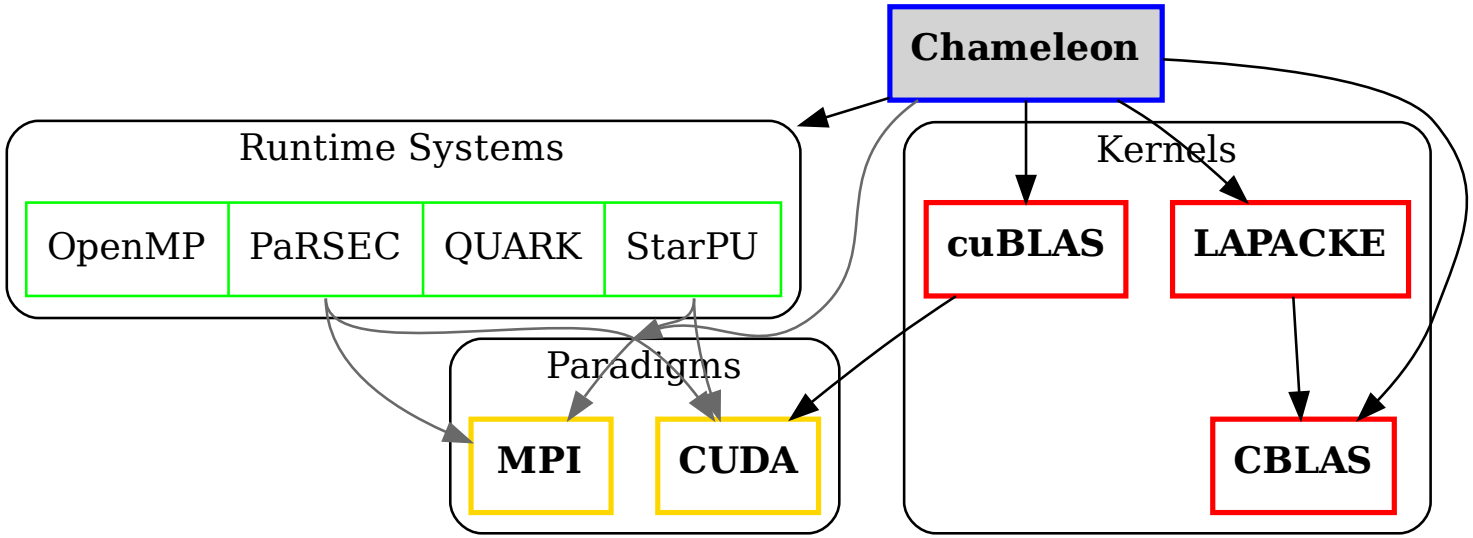
```
guix install chameleon pastix maphys
```

https://gitlab.inria.fr/guix-hpc/guix-hpc-non-free

02

Chameleon

https://gitlab.inria.fr/solverstack/chameleon

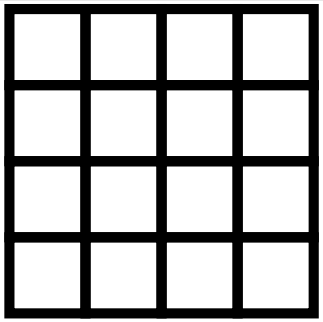


- choose between 4 runtime systems
- MPI and CUDA/cuBLAS optionals

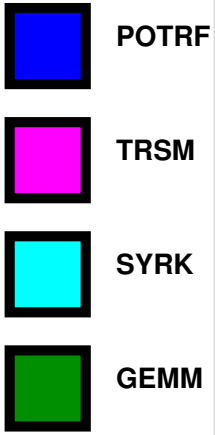
- **Language**: C/CMake, Fortran interface
- **Algorithms**: GEMM, POTRF, GETRF, GEQRF, GESVD, ...
- **Matrices forms**: general, symmetric, triangular
- **Precisions**: simple, double, complex, double complex

See list of available routines: https://solverstack.gitlabpages.inria.fr/chameleon/users_guide.html

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

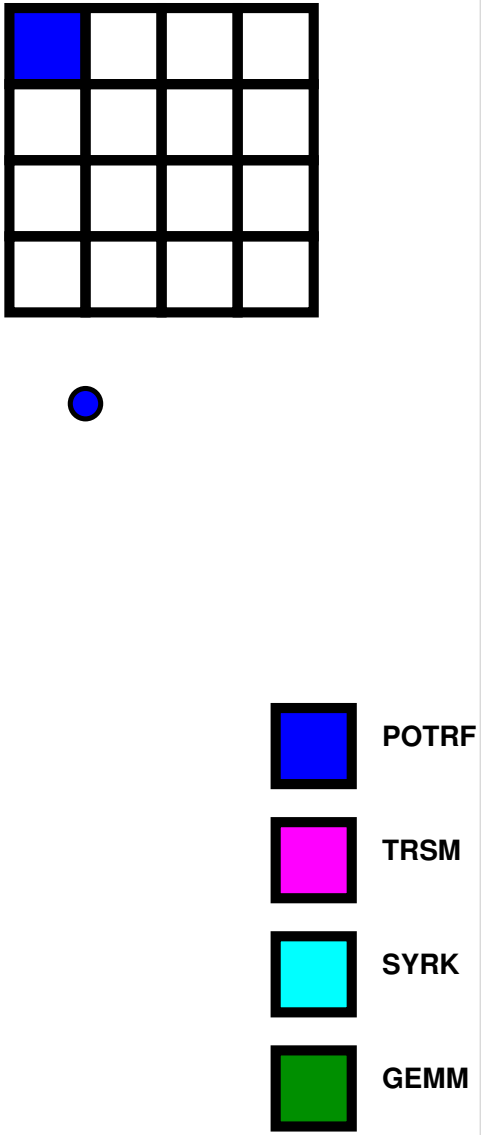


```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
  R,A[i][j], R,A[k][j]);
}
}
__wait__();
```



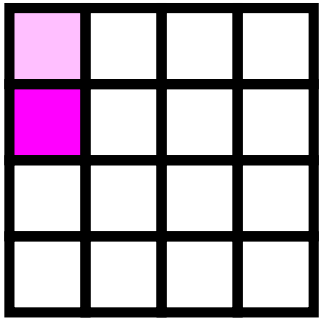
- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

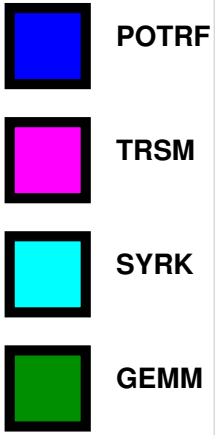
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
  R,A[i][j], R,A[k][j]);
}
}
__wait__();
```



- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

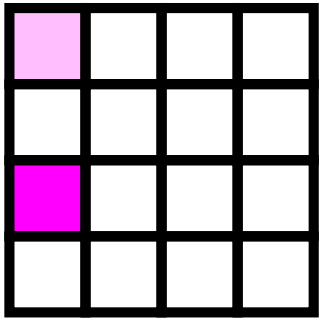
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
  R,A[i][j], R,A[k][j]);
}
}
__wait__();
```

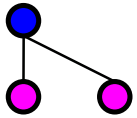


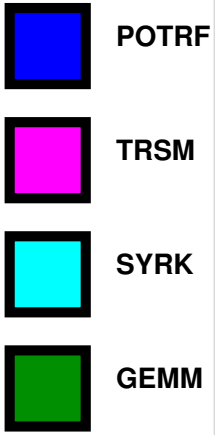


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

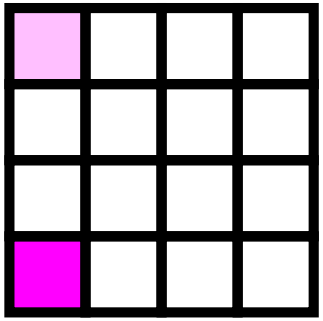


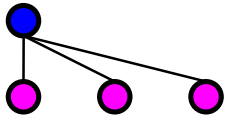


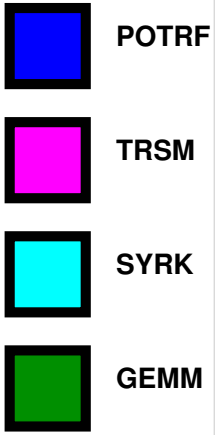


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

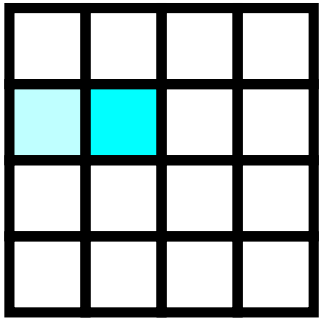


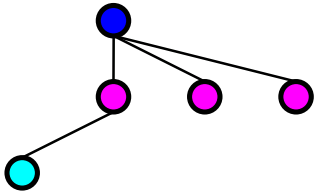


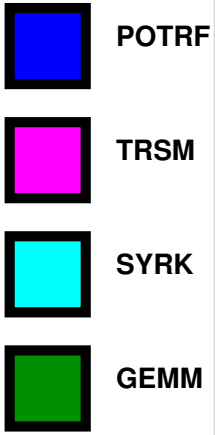


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

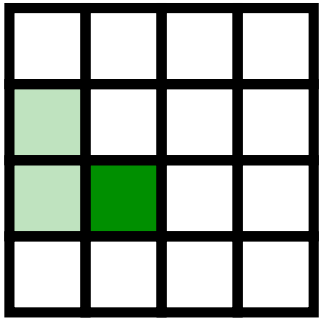


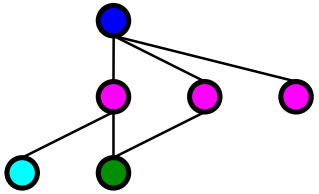


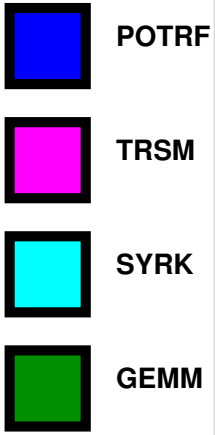


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

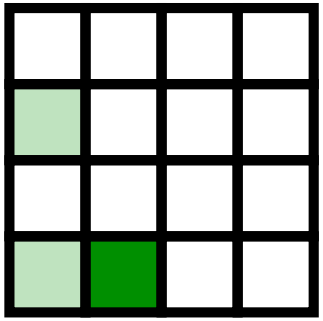


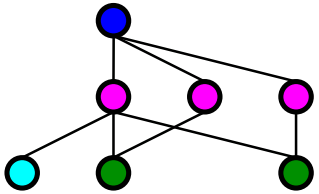


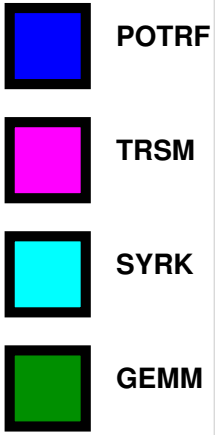


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

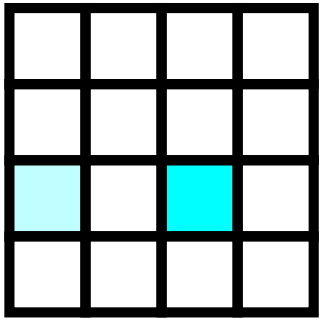


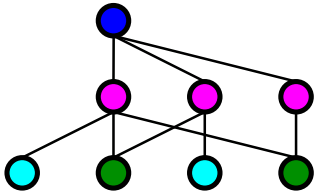


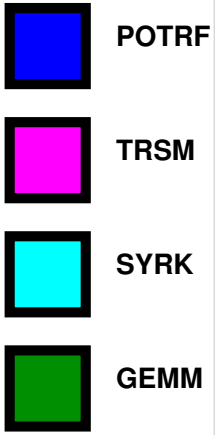


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

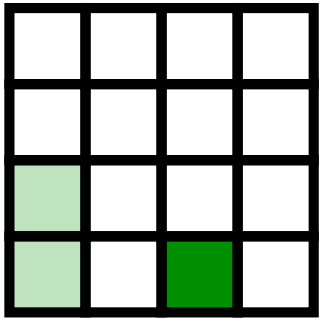


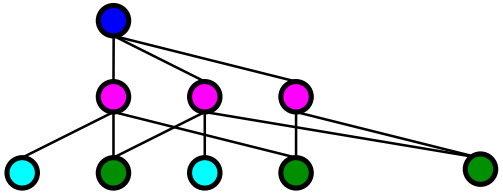


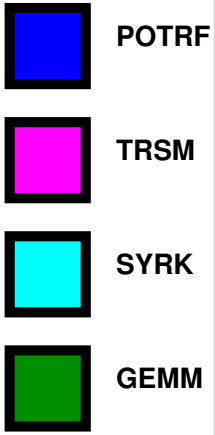


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

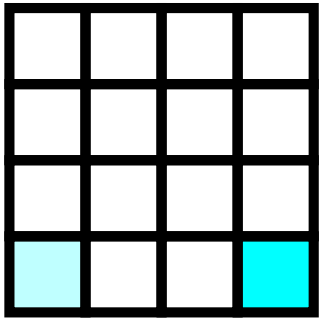


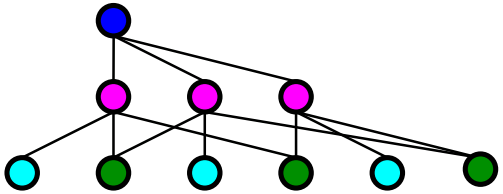


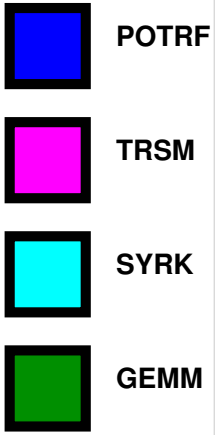


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

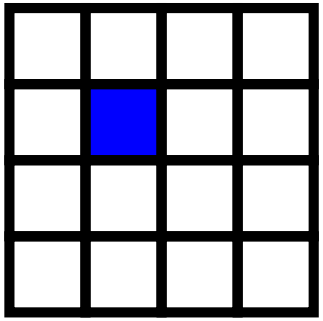


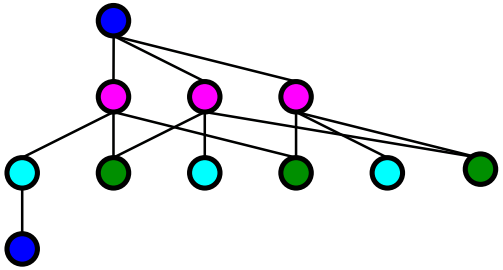


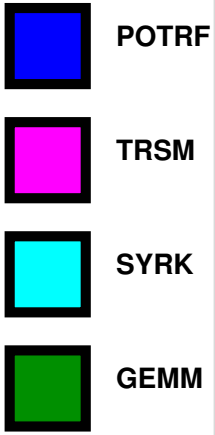


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
```

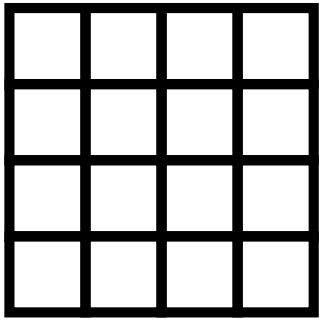


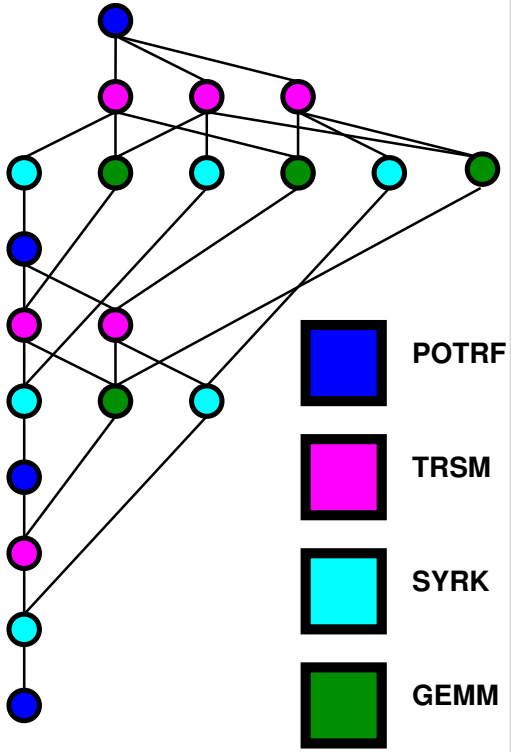


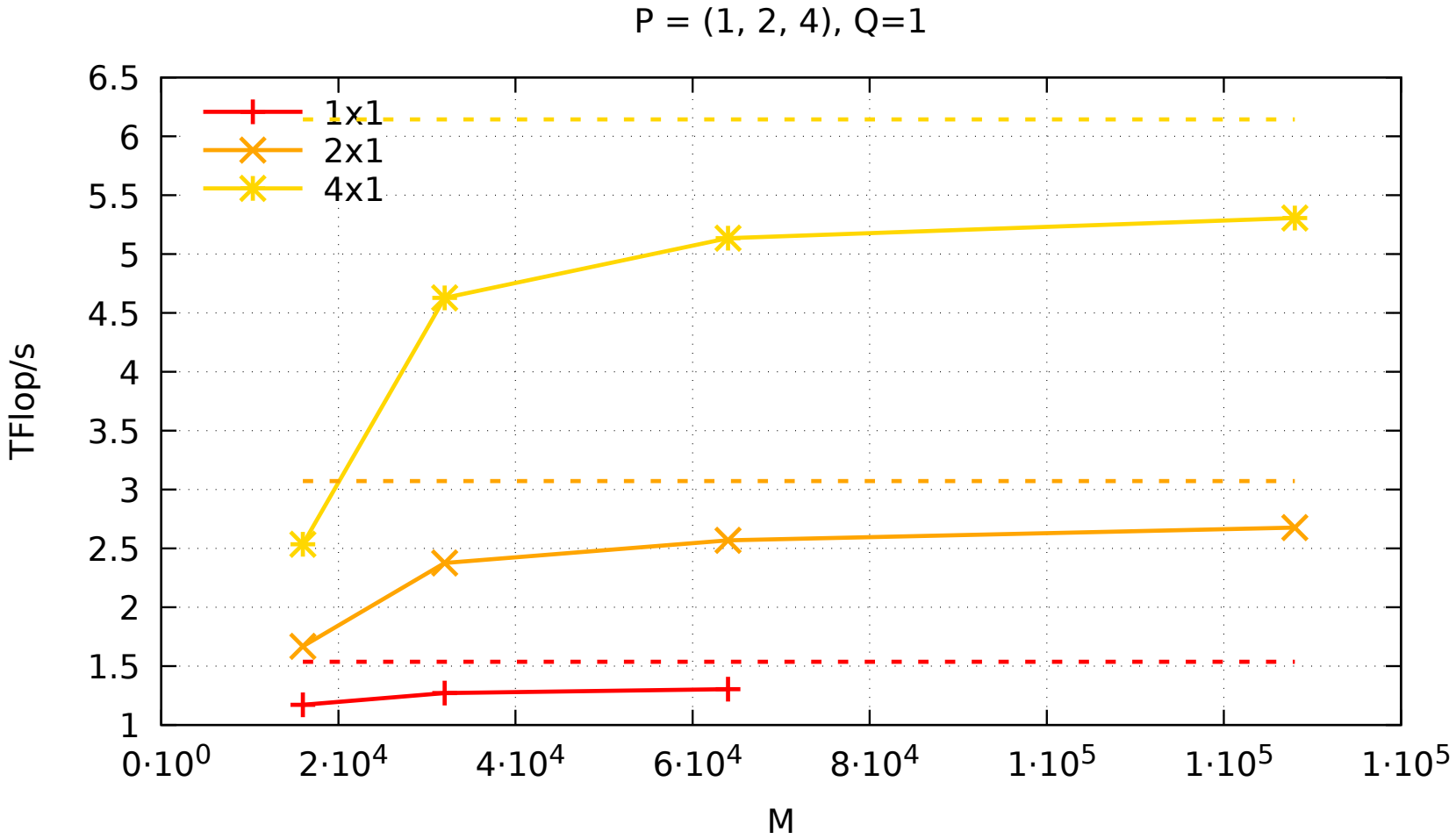


- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

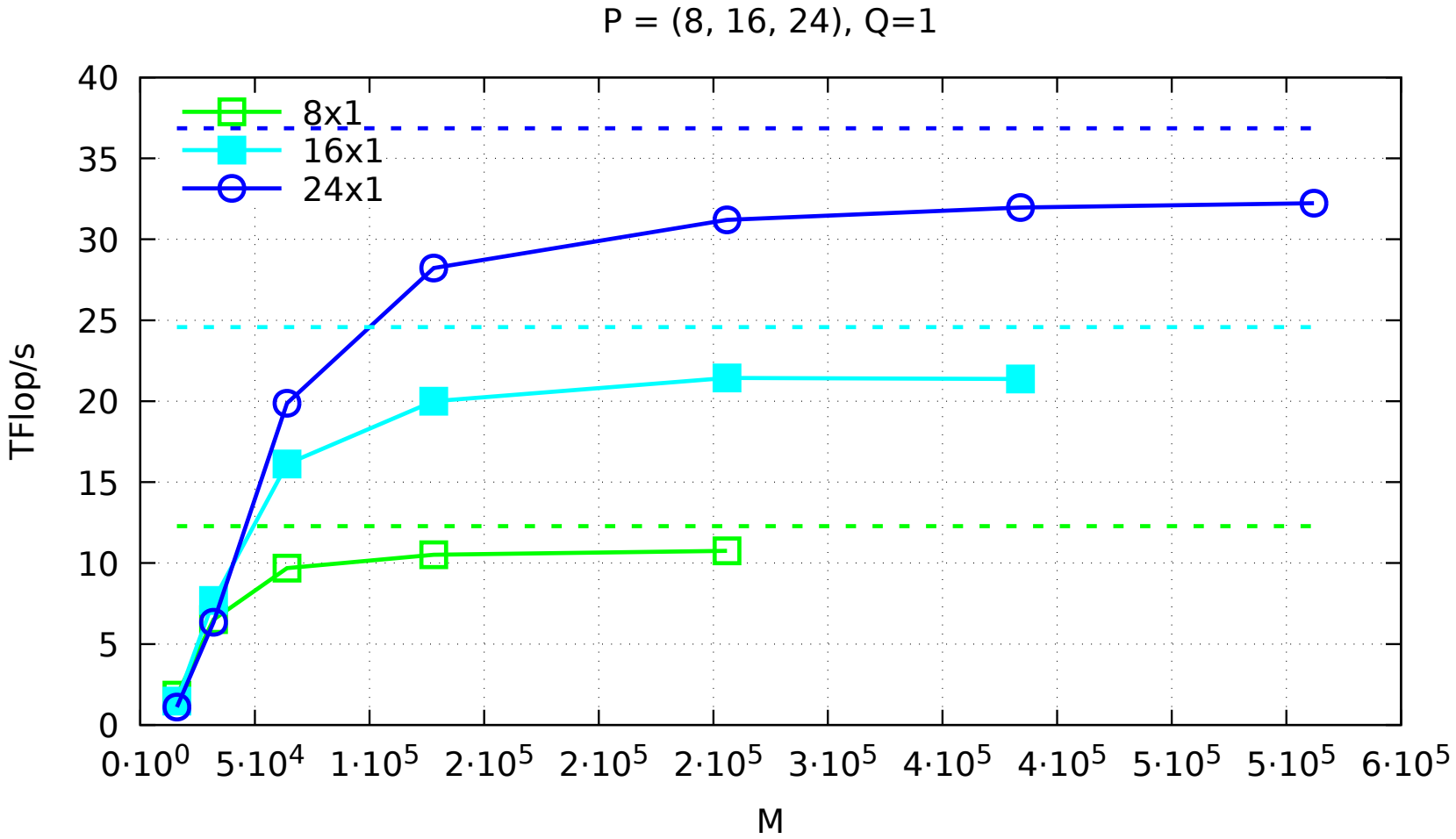
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
  R,A[i][j], R,A[k][j]);
}
}
__wait__();
```



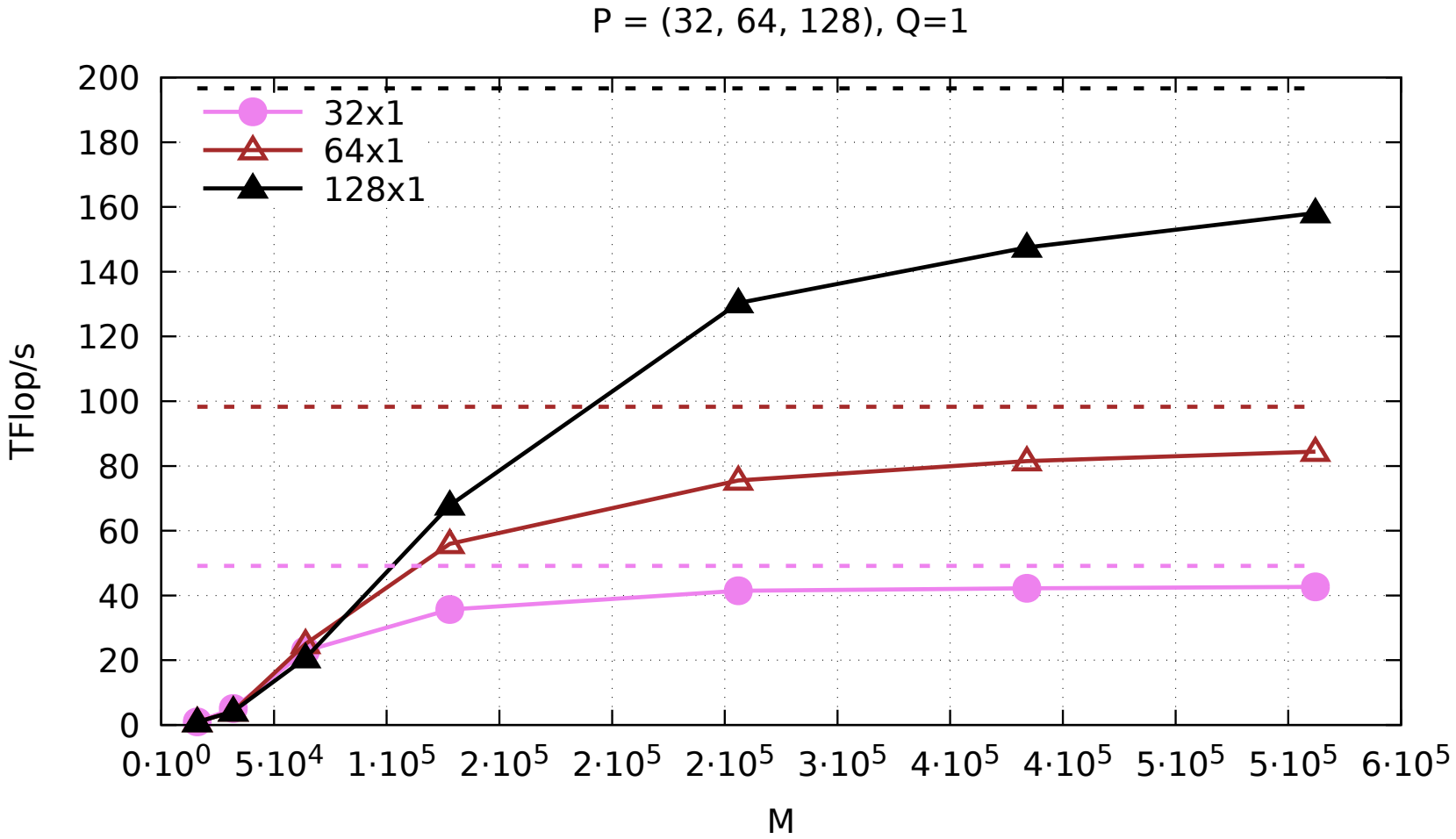




SGEMM 1-4 Occigen nodes

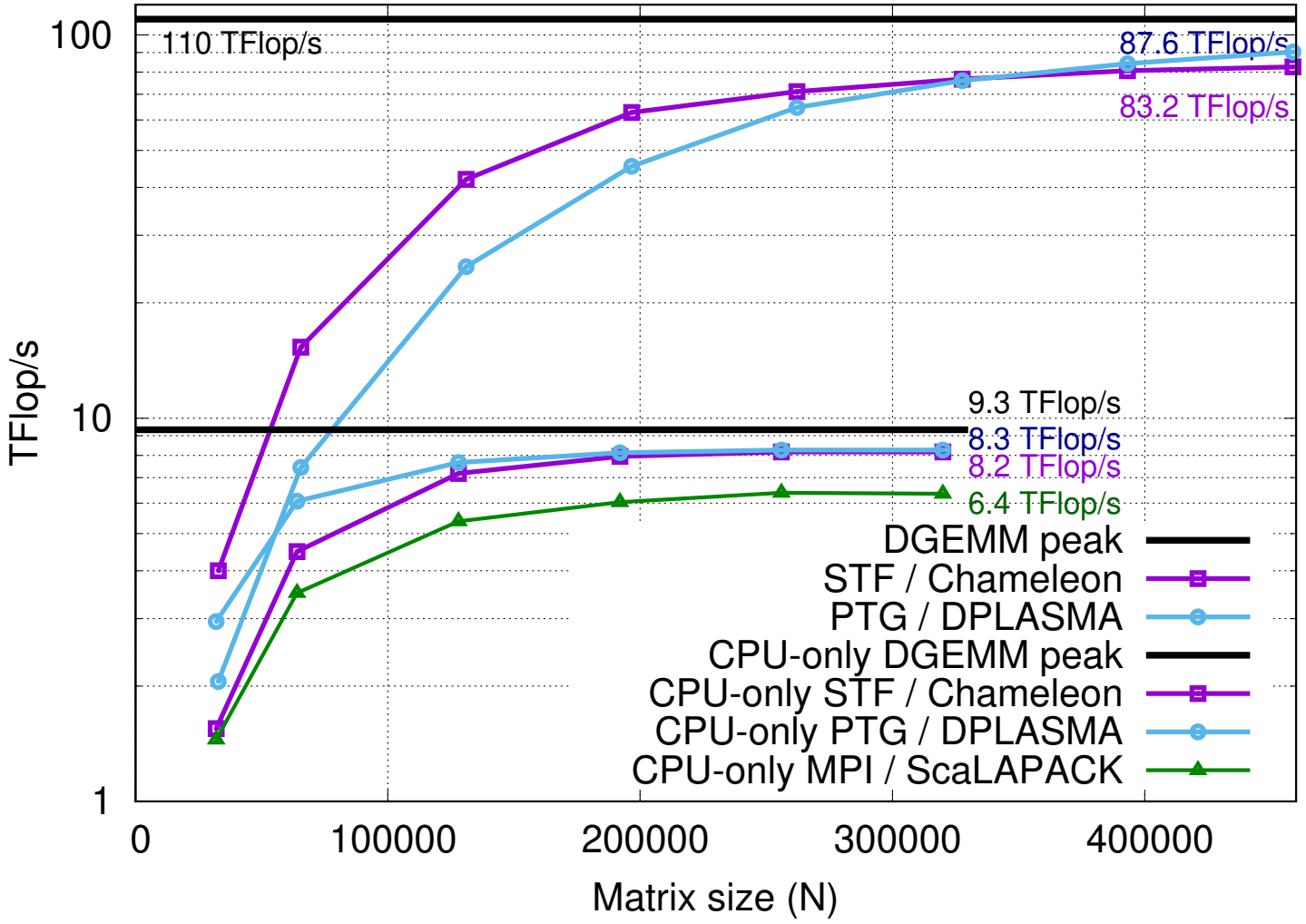


SGEMM 8-24 Occigen nodes



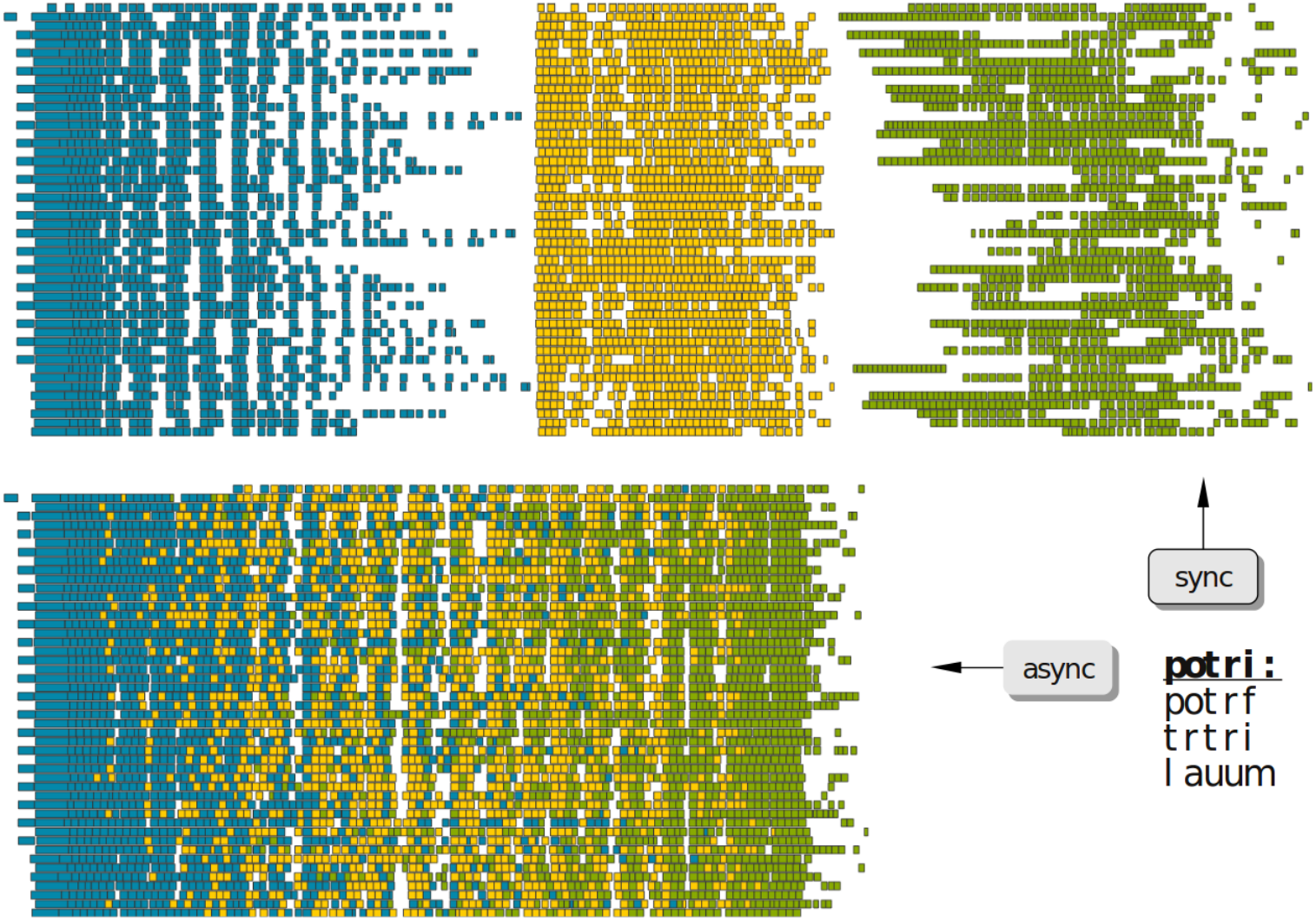
SGEMM 32-128 Occigen nodes

Software performances examples



DPOTRF 144 TERA-100 nodes with 2 Tesla M2090 GPUs

Software performances examples

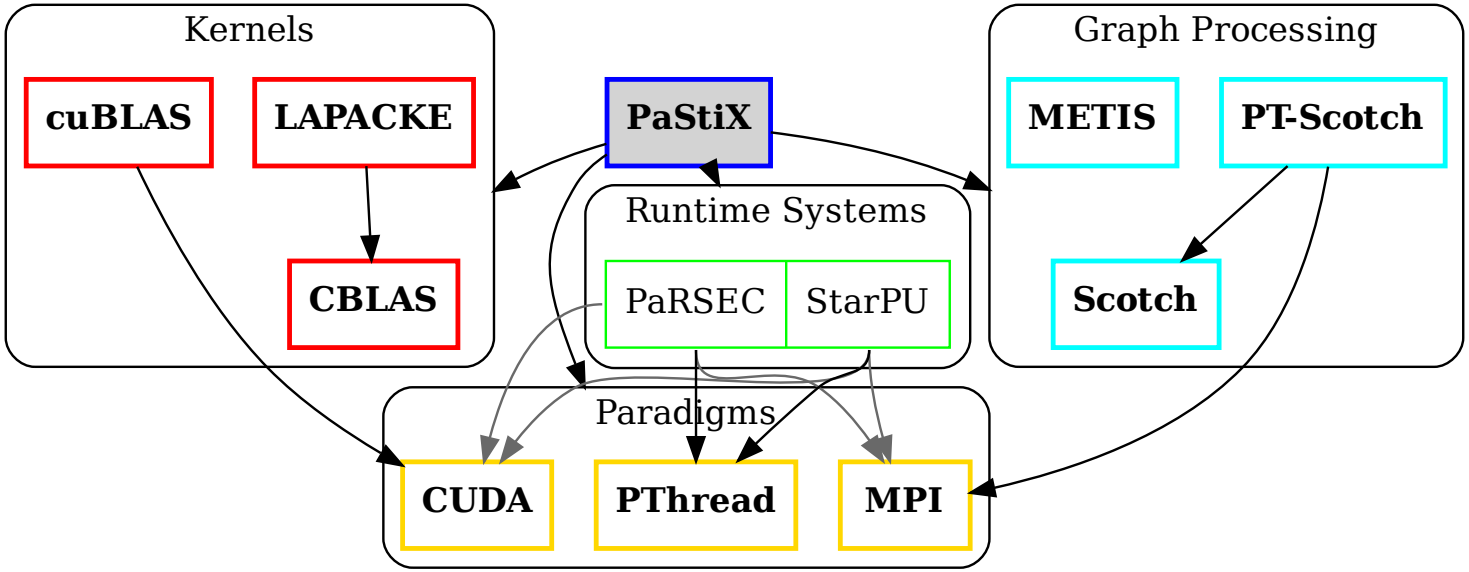


Algorithms can be pipelined

03

PaStiX

https://gitlab.inria.fr/solverstack/pastix



- choose between 2 runtime systems
- choose between 2 graph partitioners
- MPI and CUDA/cuBLAS optionals

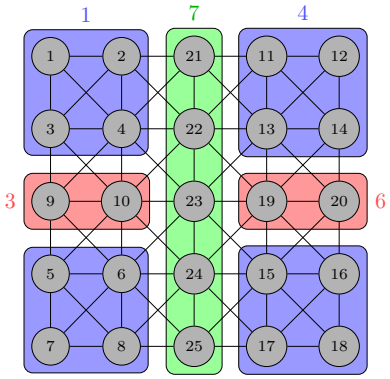
Software features

- **Language**: C/CMake, Fortran/Python/Julia interfaces
- **Algorithms**: POTRF, GETRF, TRSM, ...
- **Matrices forms**: general, symmetric, triangular
- **Storage formats**: CSC, CSR, and IJV
- **Precisions**: simple, double, complex, double complex
- **Low-Rank compression**

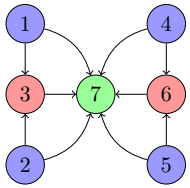
See list of available routines:
https://solverstack.gitlabpages.inria.fr/pastix/

General approach

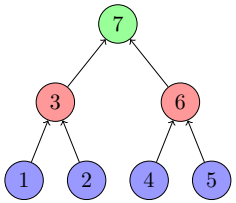
1. Use the nested dissection process to partition a sparse matrix
2. Use the minimum degree solution when leaves are small enough
3. Generate a symbolic block structure of L



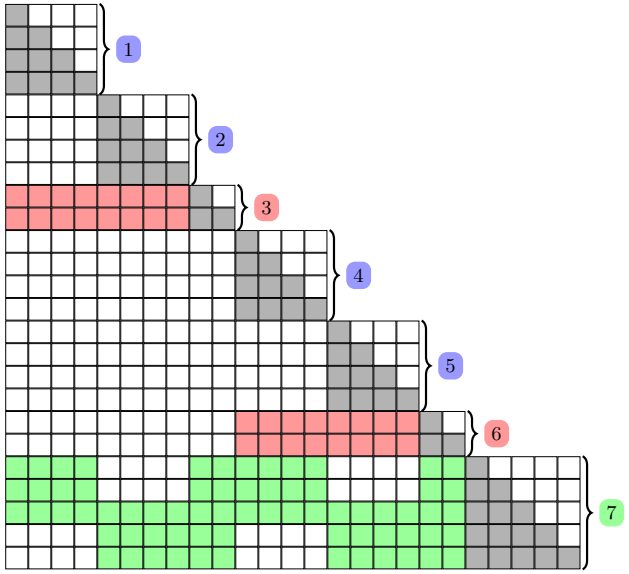
Adjacency graph (G).



Quotient graph (G^*/P) $= (G/P)^*$



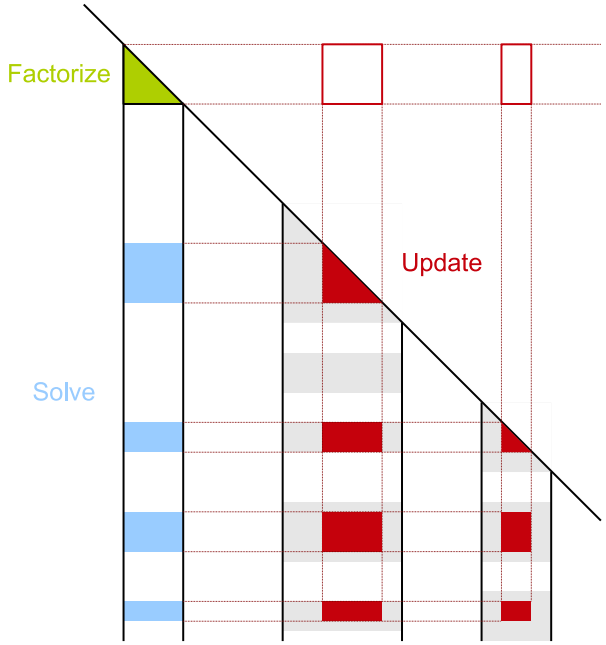
Elimination tree (T).



Factorized matrix (L).

Algorithm to eliminate the block column k

1. Factorize the diagonal block (POTRF/GETRF)
2. Solve off-diagonal blocks in the current column (TRSM)
3. Update the trailing matrix with the column's contribution (GEMM)

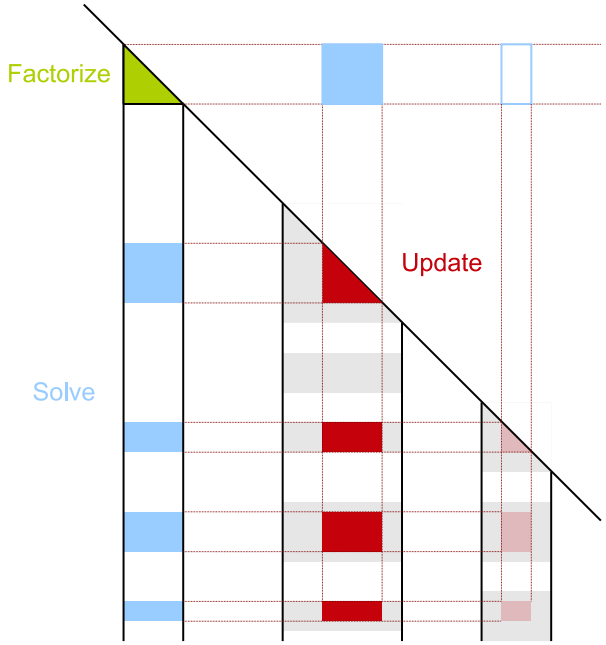


Many possibilities to do it

- One single update ≈ multi-frontal

Algorithm to eliminate the block column k

1. Factorize the diagonal block (POTRF/GETRF)
2. Solve off-diagonal blocks in the current column (TRSM)
3. Update the trailing matrix with the column's contribution (GEMM)

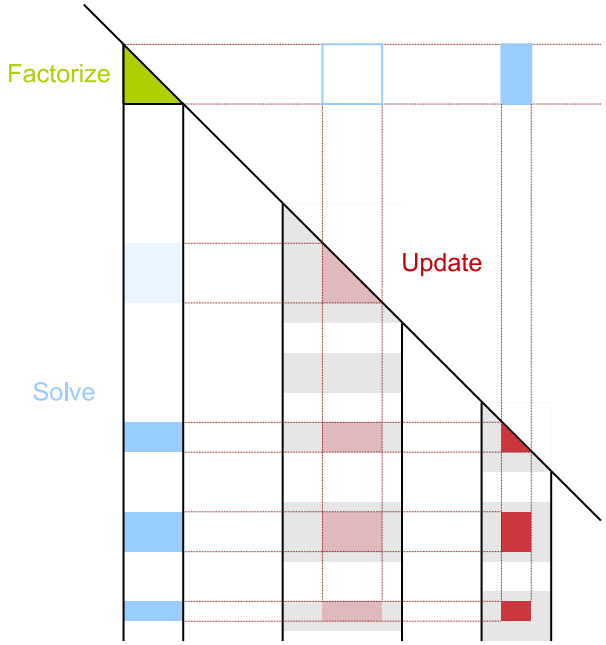


Many possibilities to do it

- One single update ≈ multi-frontal
- 1D updates per block of columns

Algorithm to eliminate the block column k

1. Factorize the diagonal block (POTRF/GETRF)
2. Solve off-diagonal blocks in the current column (TRSM)
3. Update the trailing matrix with the column's contribution (GEMM)

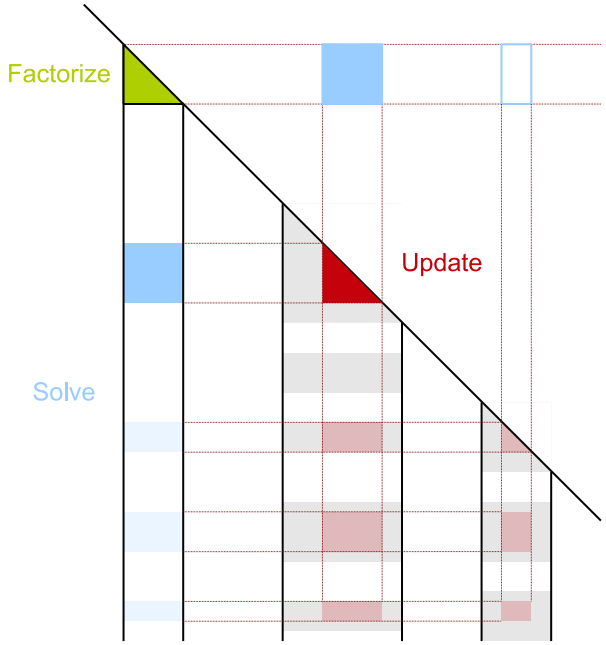


Many possibilities to do it

- One single update ≈ multi-frontal
- 1D updates per block of columns

Algorithm to eliminate the block column k

1. Factorize the diagonal block (POTRF/GETRF)
2. Solve off-diagonal blocks in the current column (TRSM)
3. Update the trailing matrix with the column's contribution (GEMM)

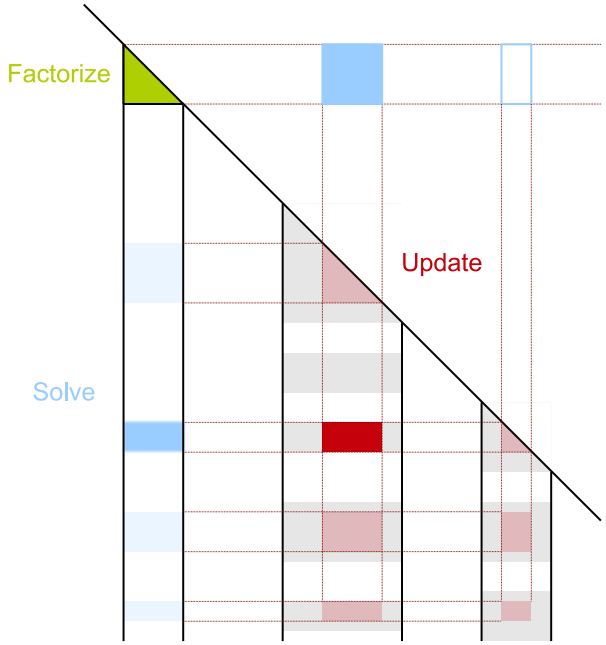


Many possibilities to do it

- One single update ≈ multi-frontal
- 1D updates per block of columns
- 2D updates ≈ Dense factorization

Algorithm to eliminate the block column k

1. Factorize the diagonal block (POTRF/GETRF)
2. Solve off-diagonal blocks in the current column (TRSM)
3. Update the trailing matrix with the column's contribution (GEMM)

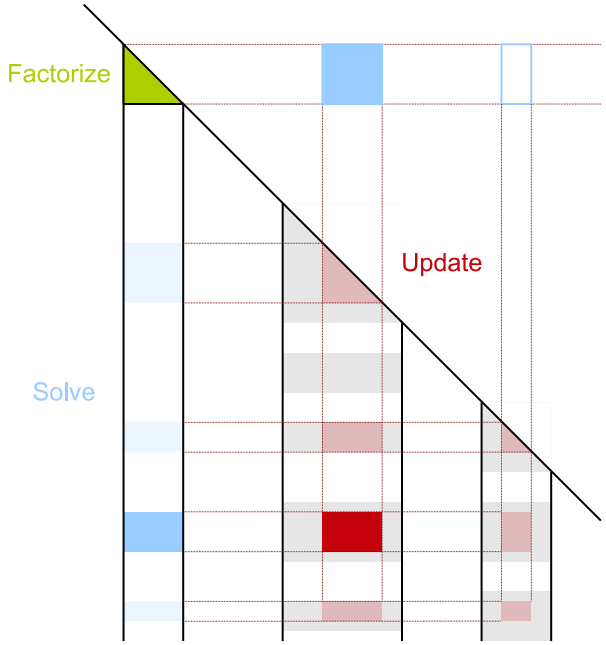


Many possibilities to do it

- One single update ≈ multi-frontal
- 1D updates per block of columns
- 2D updates ≈ Dense factorization

Algorithm to eliminate the block column k

1. Factorize the diagonal block (POTRF/GETRF)
2. Solve off-diagonal blocks in the current column (TRSM)
3. Update the trailing matrix with the column's contribution (GEMM)

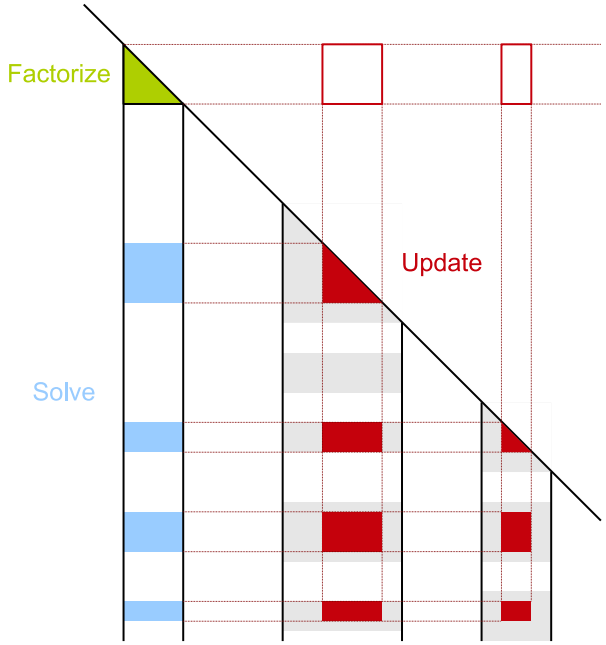


Many possibilities to do it

- One single update ≈ multi-frontal
- 1D updates per block of columns
- 2D updates ≈ Dense factorization

Algorithm to eliminate the block column k

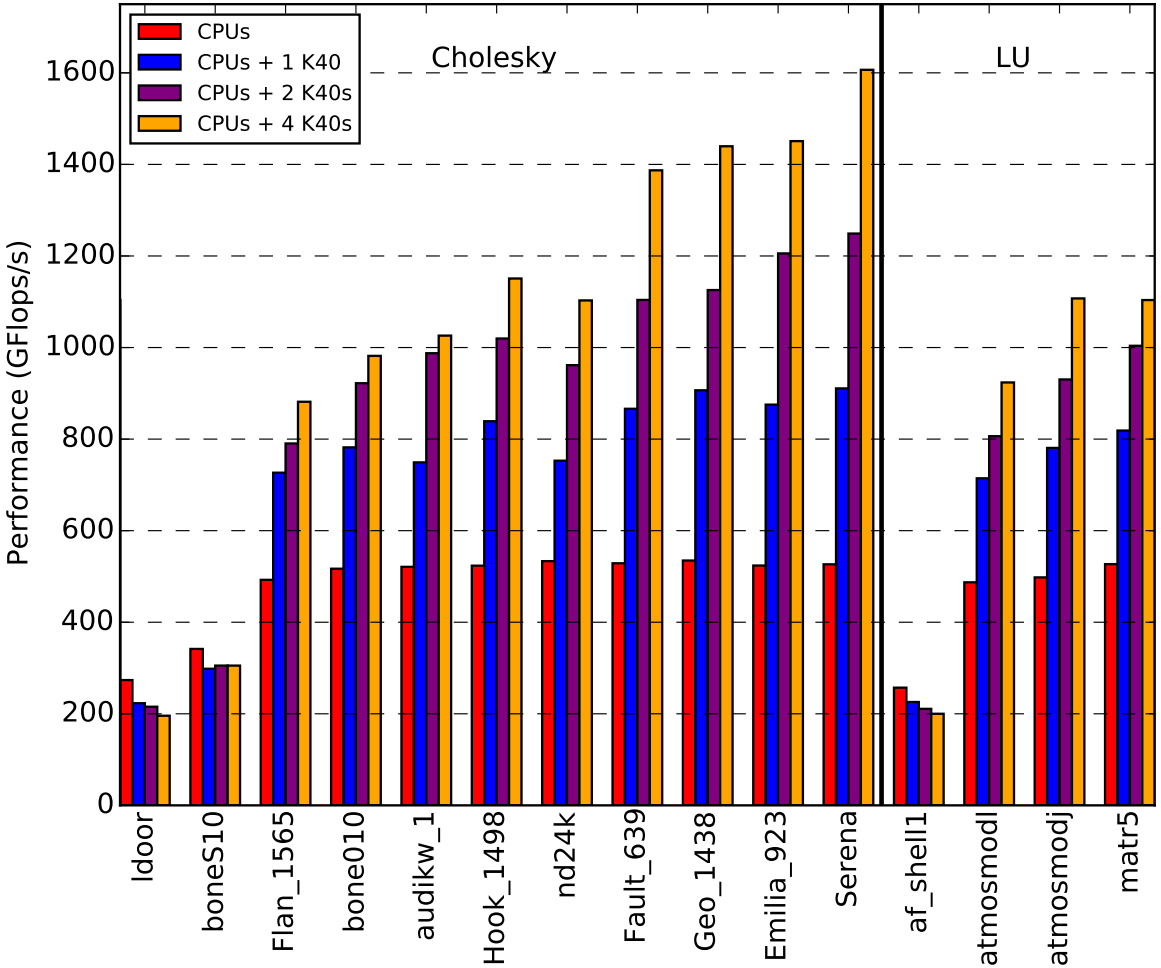
1. Factorize the diagonal block (POTRF/GETRF)
2. Solve off-diagonal blocks in the current column (TRSM)
3. Update the trailing matrix with the column's contribution (GEMM)



Many possibilities to do it

- One single update ≈ multi-frontal
- 1D updates per block of columns
- 2D updates ≈ Dense factorization

Software performances examples

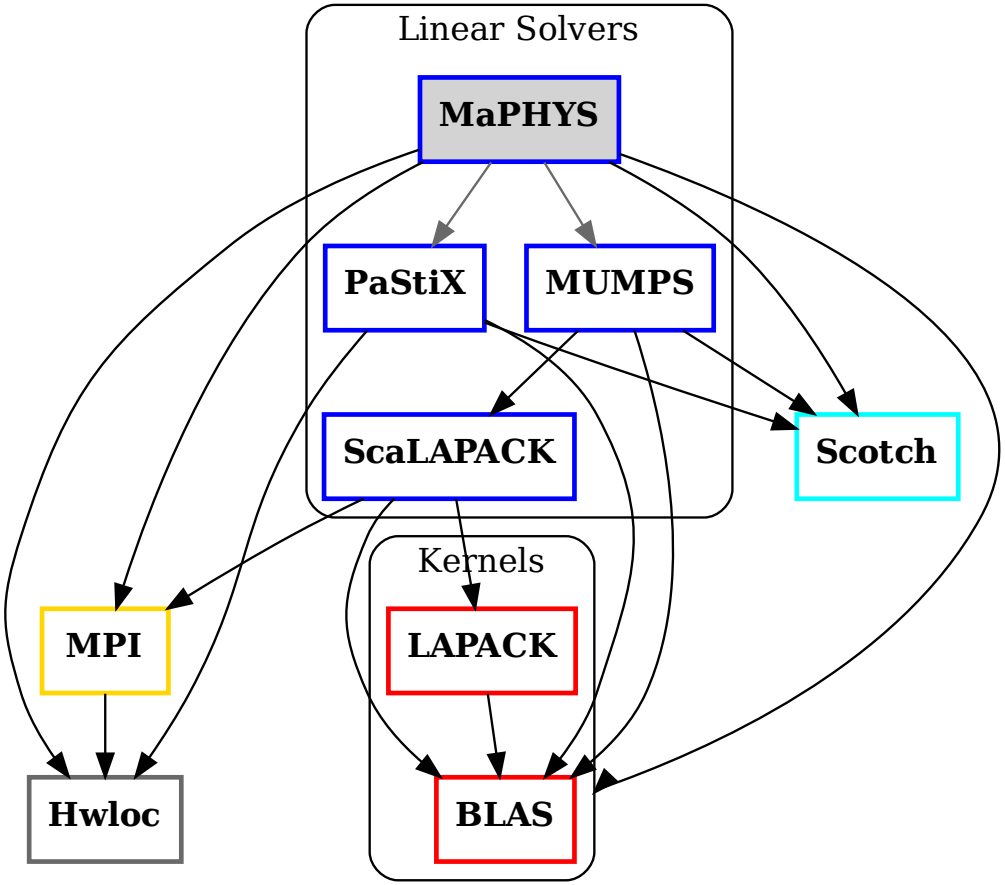


PaStiX with K40 GPUs

04

MaPHyS

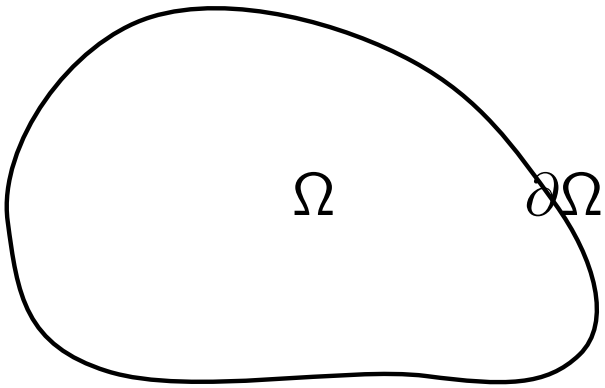
https://gitlab.inria.fr/solverstack/maphys



- choose between 2 direct solvers
- MPI + Threads

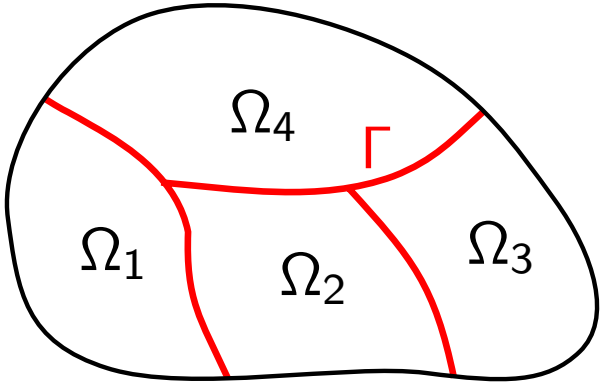
Software features

- **Language**: Fortran 90, modern C++ (WIP), CMake
- **Precisions**: simple, double, complex, double complex
- **Algorithms**: CG/GMRES, Algebraic Additive Schwarz, ...
- **Matrices forms**: general, symmetric, SPD
- **Storage formats**: IJV



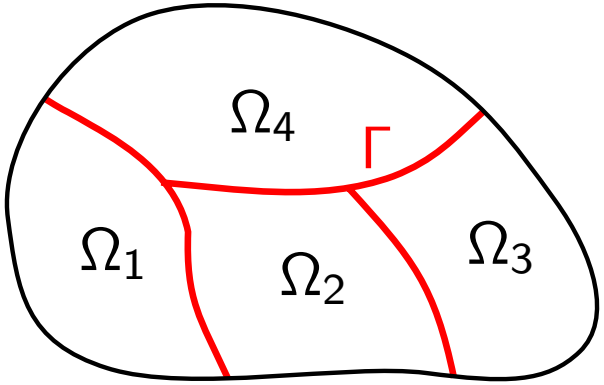
1. PDE defined on the global domain Ω

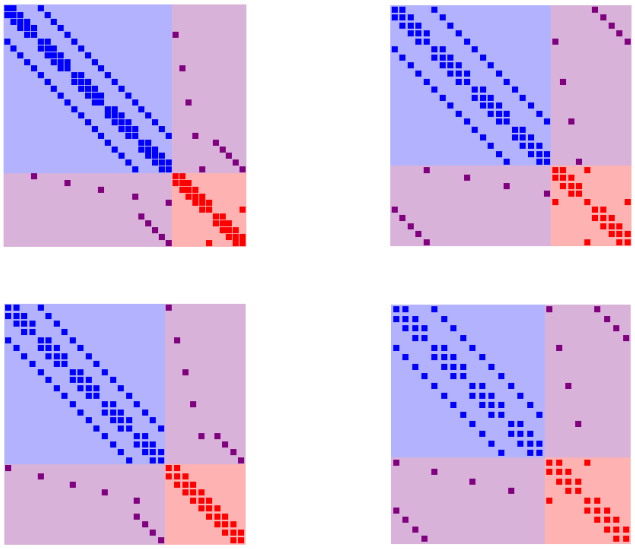
Under the hood



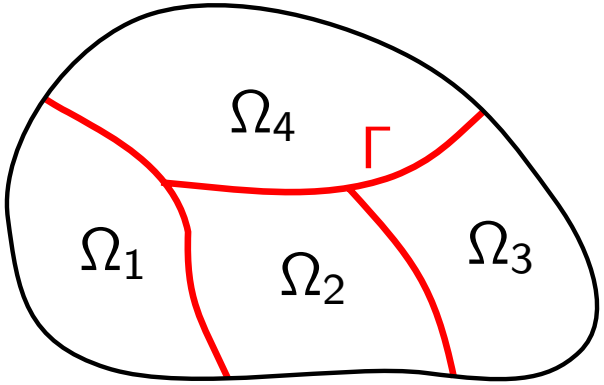
1. PDE defined on the global domain Ω
2. Partition into subdomains Ω_i

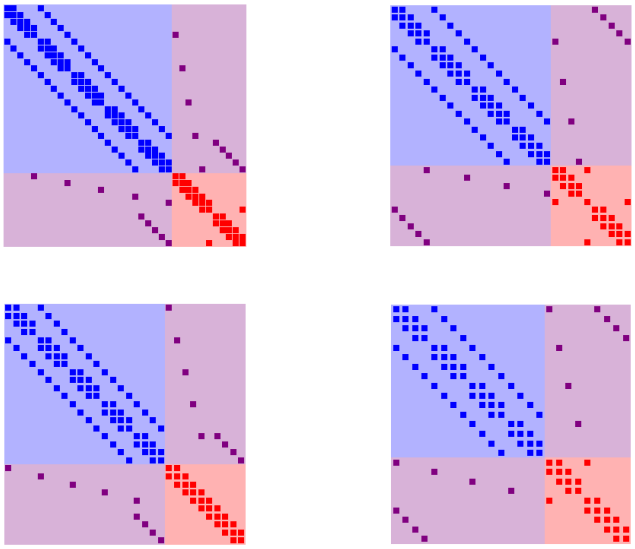
Under the hood





1. PDE defined on the global domain Ω
2. Partition into subdomains Ω_i
3. Discretization at the subdomain level $\Omega_i \rightarrow \mathcal{A}_i$
   - Neumann boundary condition on Γ





1. PDE defined on the global domain Ω
2. Partition into subdomains Ω_i
3. Discretization at the subdomain level $\Omega_i \rightarrow \mathcal{A}_i$
   - Neumann boundary condition on Γ
4. **DDM at the algebraic level**

Domain decomposition

Partition unknowns: interior and interface unknowns Computed by SCOTCH or provided by the user

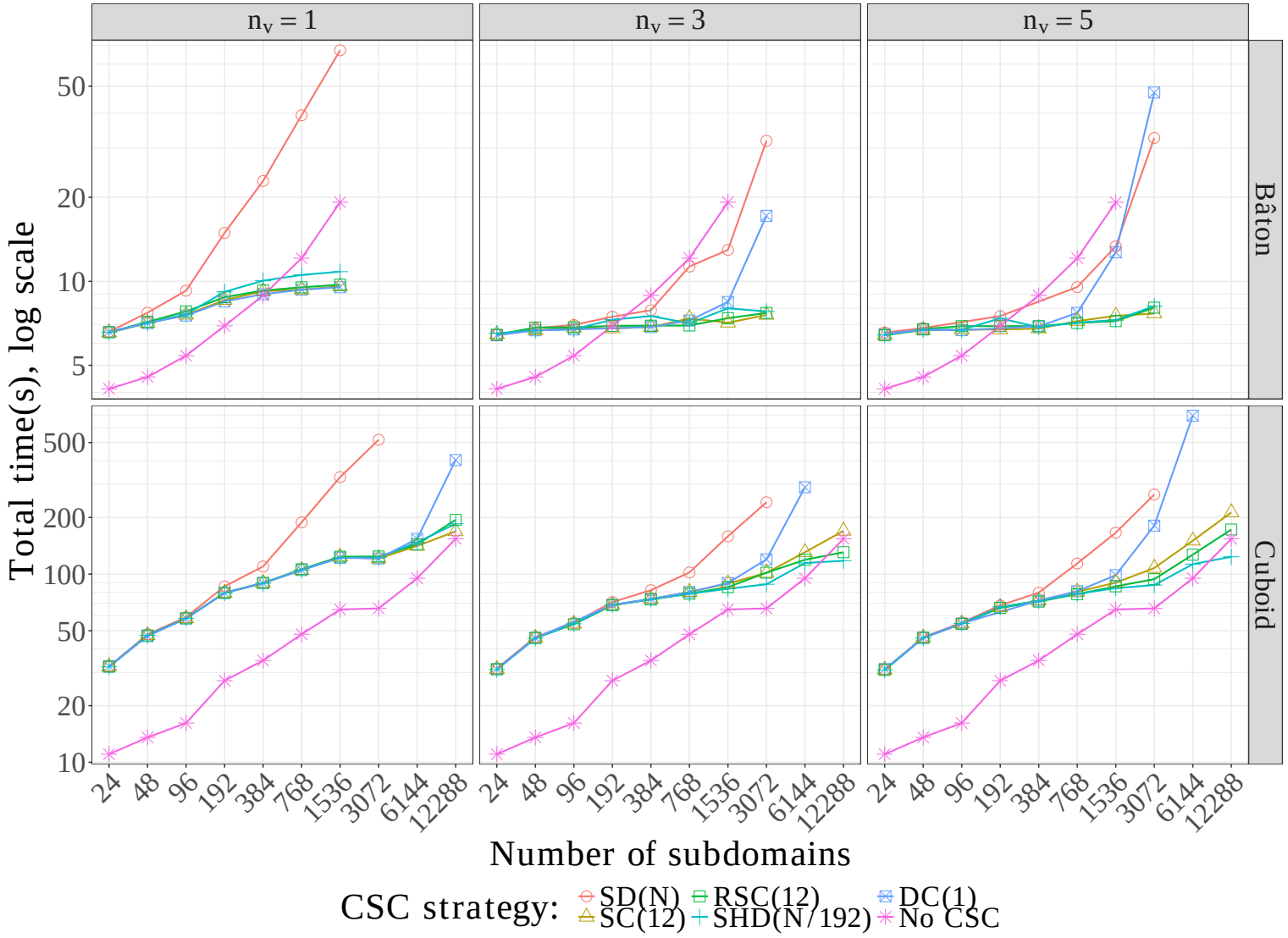
Direct sparse factorization

Direct methods to factorize interior unknowns and compute Schur complement on interface unknowns (using PaStiX or MUMPS)

Iterative solve

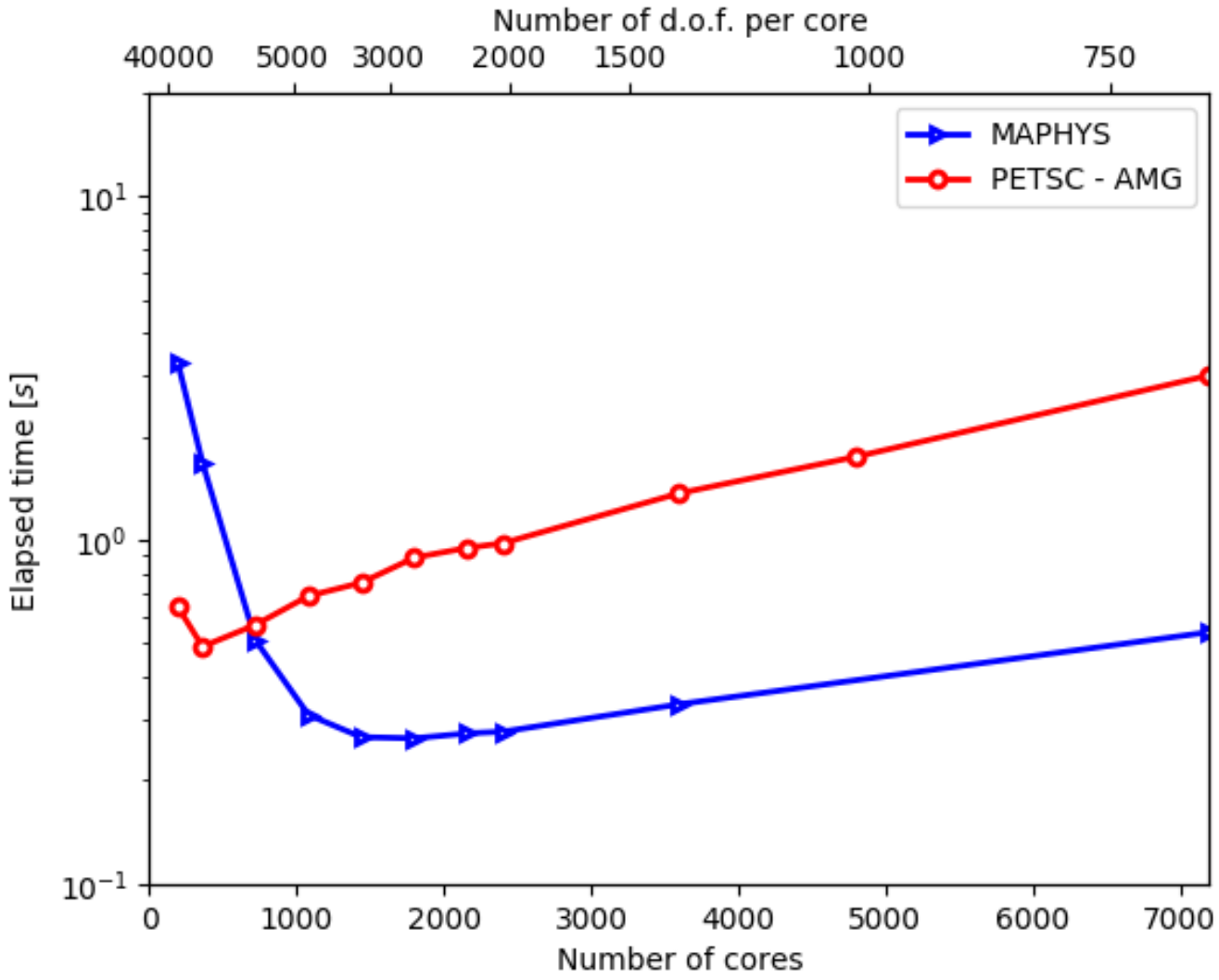
- Computation of an Additive Schwarz preconditioner on the Schur complement, optionally with a Coarse Space Correction
- Iterative solve on the Schur complement (CG, GMRES algorithms, with BLAS/LAPACK)

Software performances examples



Heterogeneous diffusion

Software performances examples



Plasma propulsion (4.5 Mdof)