PaStiX and MaPHYS overview

INRIA, Bordeaux

X. Lacoste, S. Nakov

X. Lacoste
S. Nakov
LaBRI – Inria Bordeaux Sud-Ouest

July 10, 2014

1
PaStiX

PaStiX

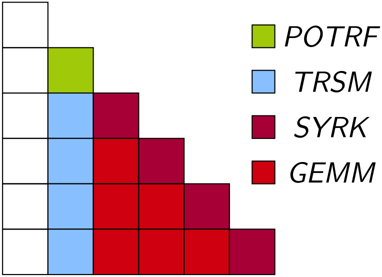
About PaStiX

- Direct sparse linear solver;
- Hybrid MPI and P-Thread implementation.

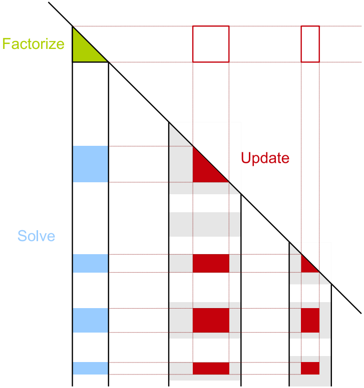
Current works

- Use GPUs to accelerate factorization;
  - dedicated sparse GEMM kernel;
  - implementation on top of generic runtimes (StarPU and PaRSEC).
- Optimized schur complement computation;
- Finite element assembly API;
- H-Matrix.

Tasks structure

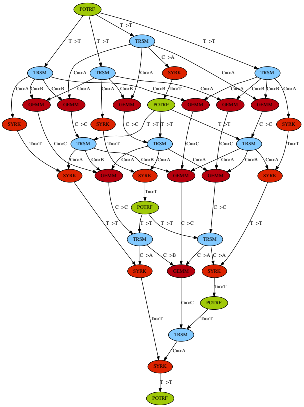


(a) Dense tile task decomposition

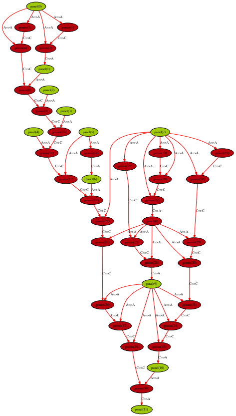


(b) Decomposition of the task applied while processing one panel

DAG representation



(c) Dense DAG



(d) Sparse DAG representation of a sparse LDL^T factorization

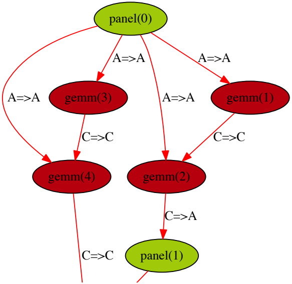
Supernodal sequential algorithm

```
forall the Supernode S_1 do
    panel (S_1);
    /* update of the panel                                        */
    forall the extra diagonal block B_i of S_1 do
        S_2 ← supernode_in_front_of (B_i);
        gemm (S_1,S_2);
        /* sparse GEMM B_{k,k≥i} × B_i^T substracted from
            S_2                                                   */
    end
end
```

StarPU Tasks submission

```
forall the Supernode S_1 do
    submit_panel (S_1);
    /* update of the panel                                  */
    forall the extra diagonal block B_i of S_1 do
        S_2 ← supernode_in_front_of (B_i);
        submit_gemm (S_1,S_2);
        /* sparse GEMM B_{k,k≥i} × B_i^T subtracted from S_2
           */
    end
    wait_for_all_tasks ();
end
```

PaRSEC's parameterized task graph



Task Graph

```
panel(j)

/* Execution Space */
j = 0 .. cblknbr-1

/* Task Locality (Owner Compute) */
:A(j)

/* Data dependencies */
RW A <- ( leaf ) ? A(j) : C gemm( lastbrow )
     -> A gemm(firstblock+1 .. lastblock)
     -> A(j)
```

Panel Factorization in JDF Format

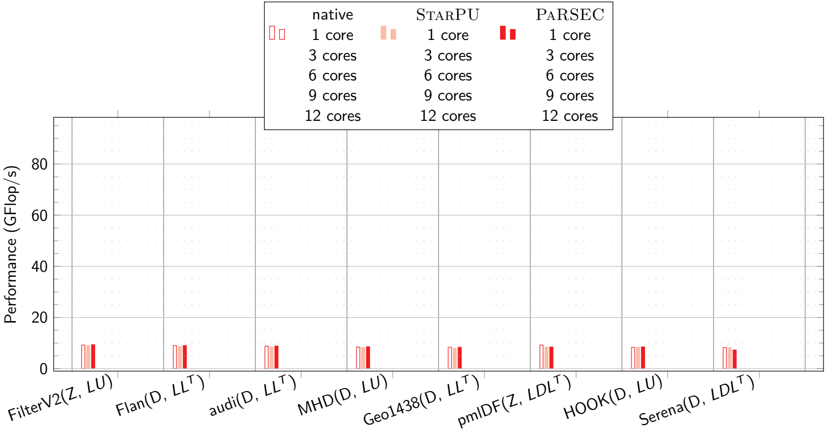
Matrices and Machines

Matrix	Prec	Method	Size	nnz_A	nnz_L	TFlop
FilterV2	Z	LU	0.6e+6	12e+6	536e+6	3.6
Flan	D	LL^T	1.6e+6	59e+6	1712e+6	5.3
Audi	D	LL^T	0.9e+6	39e+6	1325e+6	6.5
MHD	D	LU	0.5e+6	24e+6	1133e+6	6.6
Geo1438	D	LL^T	1.4e+6	32e+6	2768e+6	23
Pmldf	Z	LDL^T	1.0e+6	8e+6	1105e+6	28
Hook	D	LU	1.5e+6	31e+6	4168e+6	35
Serena	D	LDL^T	1.4e+6	32e+6	3365e+6	47

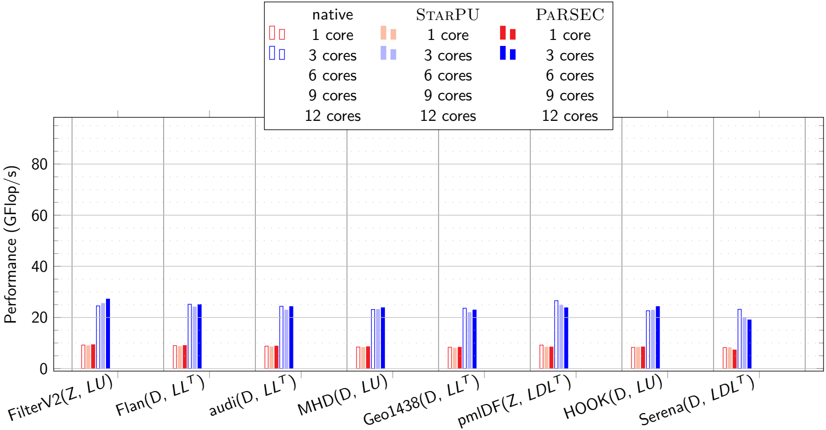
Table: Matrix description (Z: double complex, D: double).

Machine	**Processors**	**Frequency**	GPUs	**RAM**
Mirage	Westmere Intel Xeon X5650 (2×6)	2.67 GHz	Tesla M2070 ($\times 3$)	36 GB

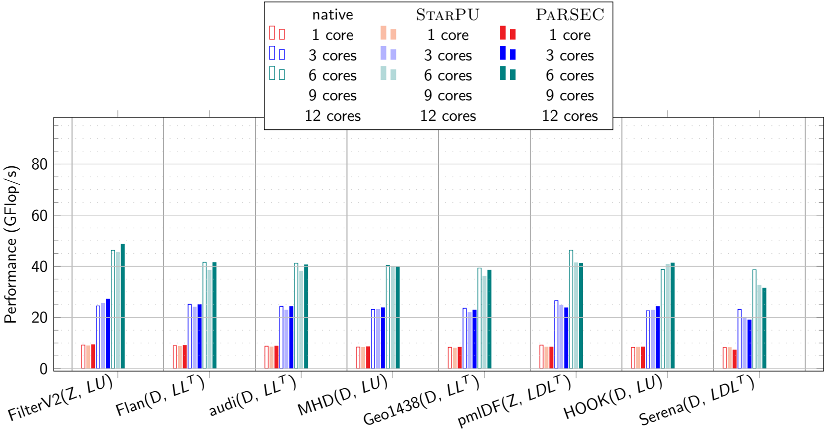
CPU scling study: GFlop/s during numerical factorization



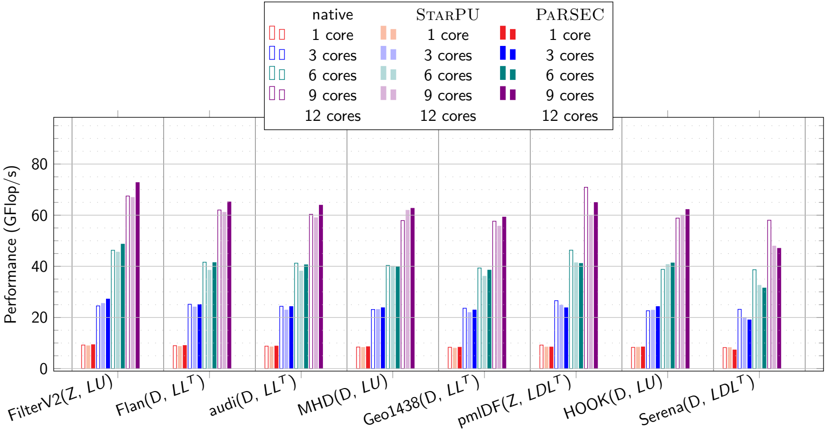
CPU scling study: GFlop/s during numerical factorization



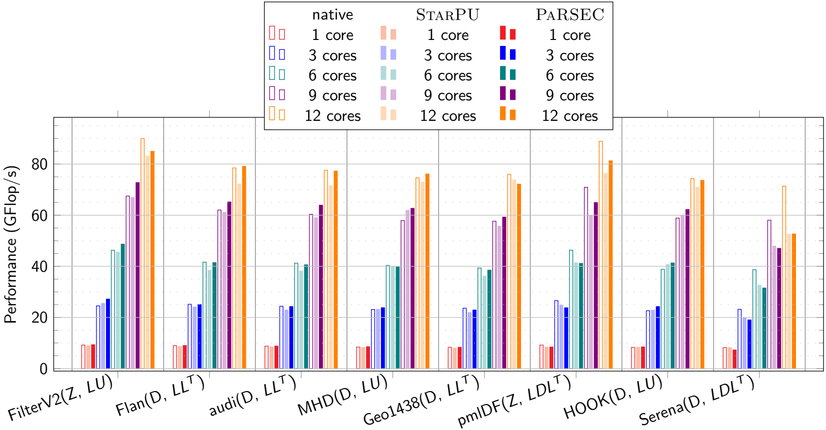
CPU scling study: GFlop/s during numerical factorization



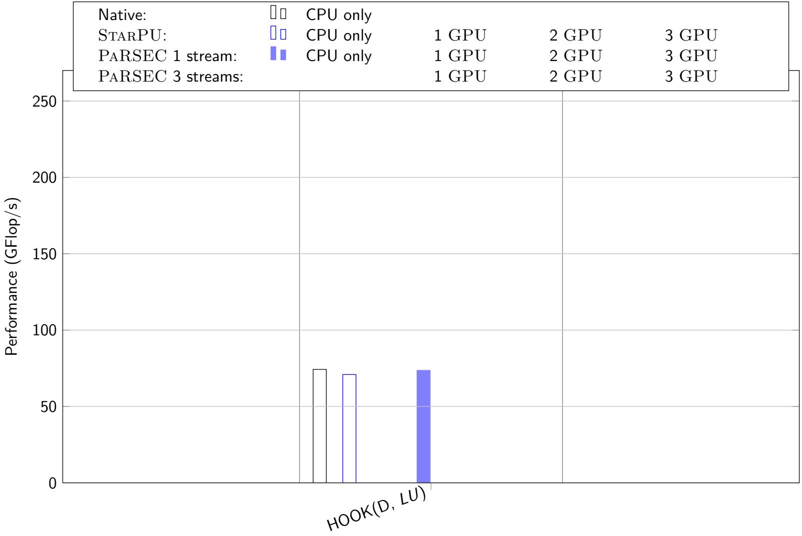
CPU scling study: GFlop/s during numerical factorization



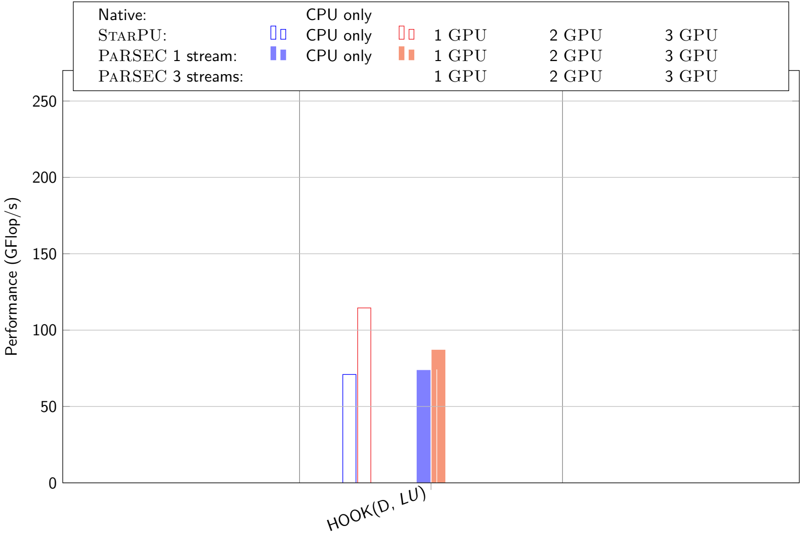
CPU scling study: GFlop/s during numerical factorization



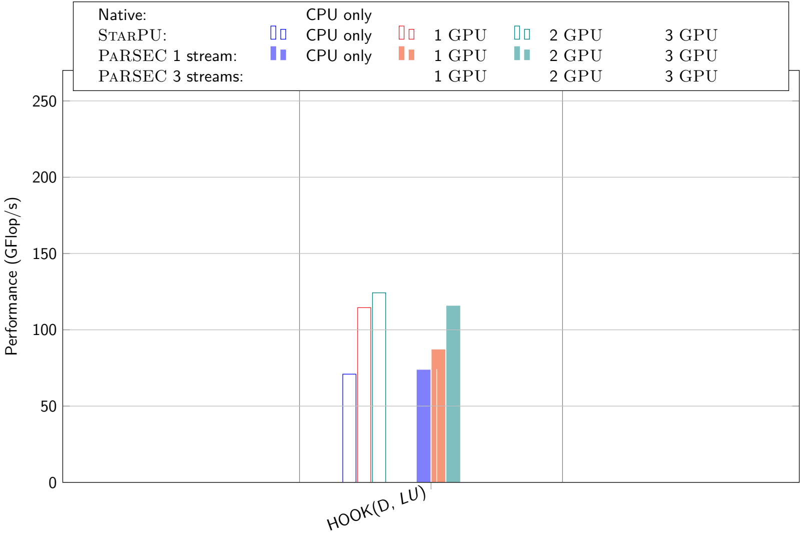
GPU scaling study



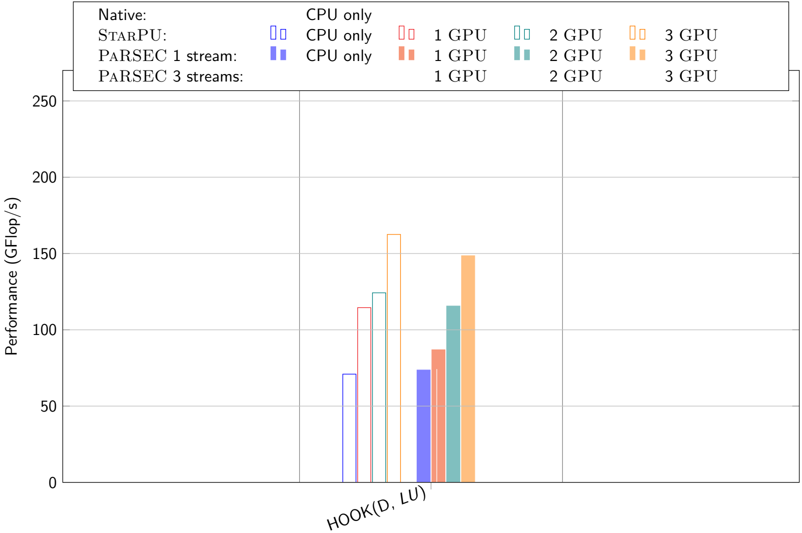
GPU scaling study



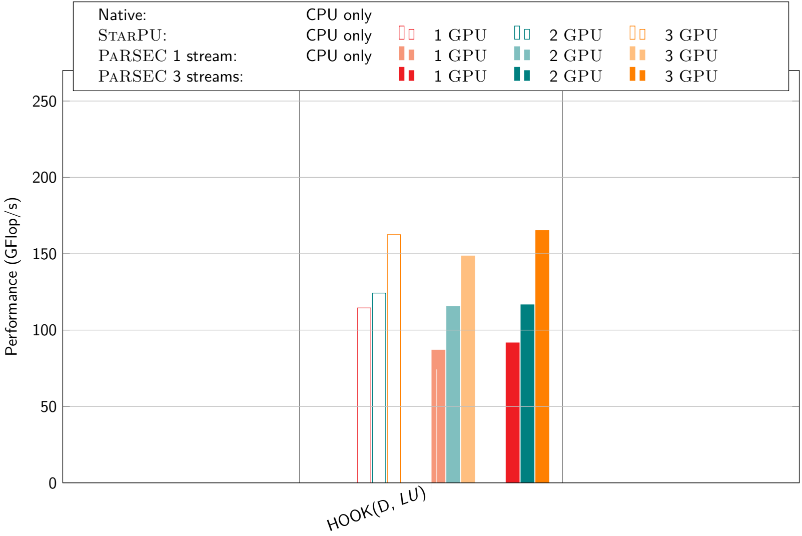
GPU scaling study



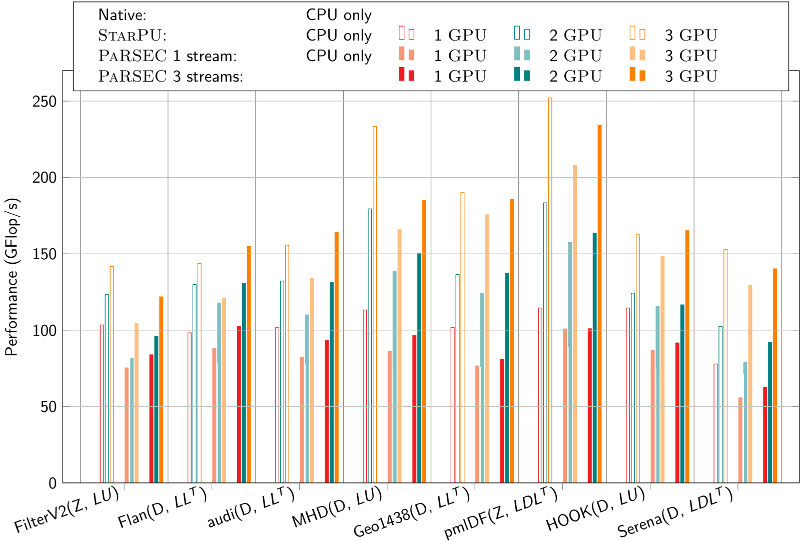
GPU scaling study



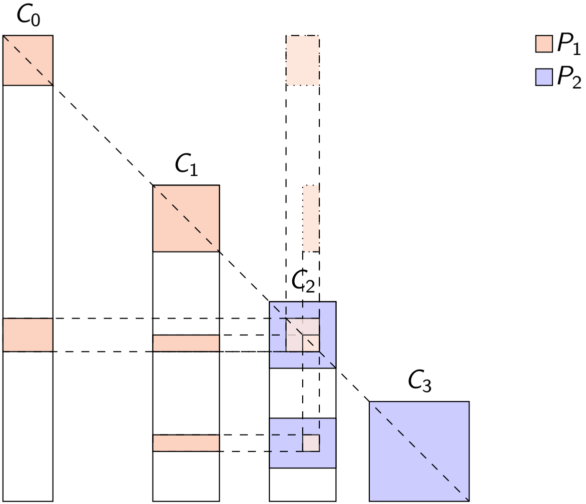
GPU scaling study



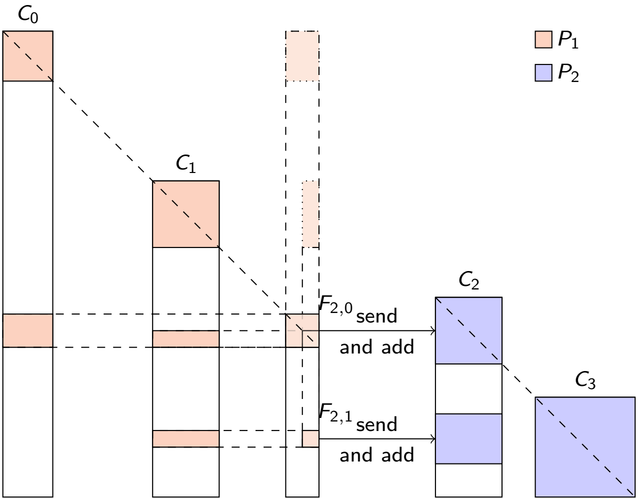
GPU scaling study



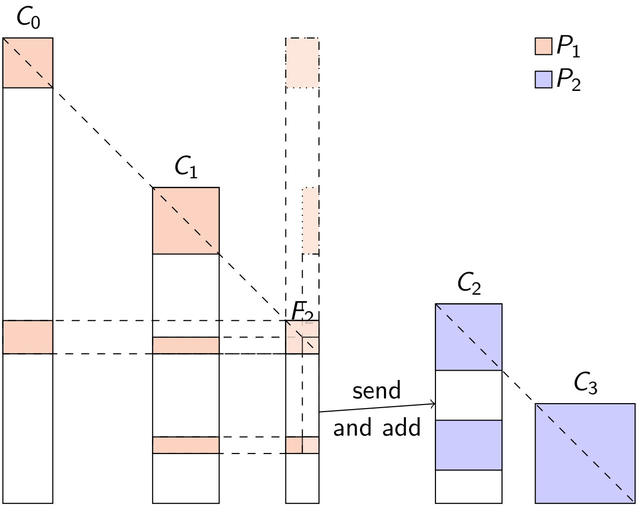
Distributed implementation



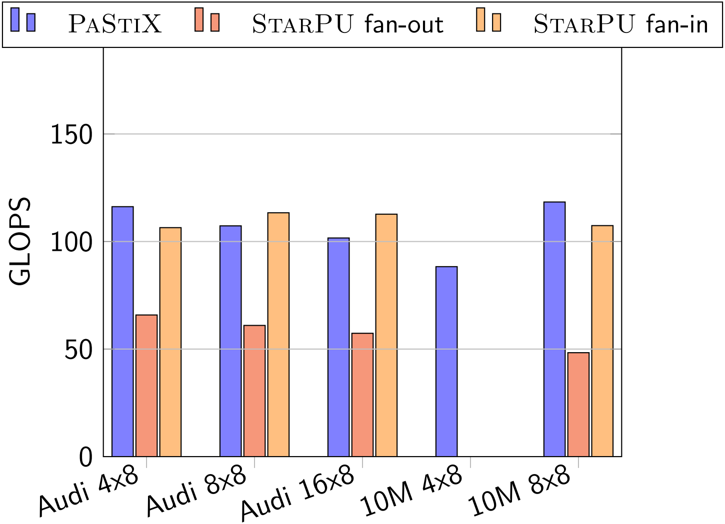
Fanin version (PaStiX)



Fanin version (StarPU)



Distributed preliminary results



Schur complement computation in PaStiX

1. User provides a list of schur unknowns;
2. PaStiX moves this unknown to the last block of the matrix which is centralized;
3. PaStiX perform the factorization and stop before Schur block factorization;
4. PaStiX returns the computed Schur complement;
5. User can call PaStiX to solve using non Schur unknowns.

Improvements on Schur complement

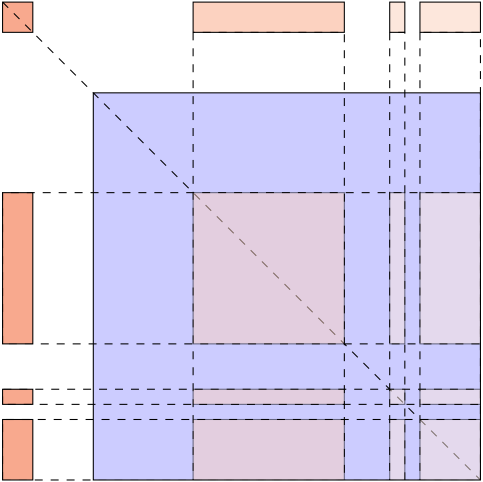
Next release

- Temporary buffer size optimization:
  - Do not take into account last block (doesn't produce GEMM);
  - Split Schur facing blocks as if Schur was splitted;
- Multi-threaded update of the Schur complement.

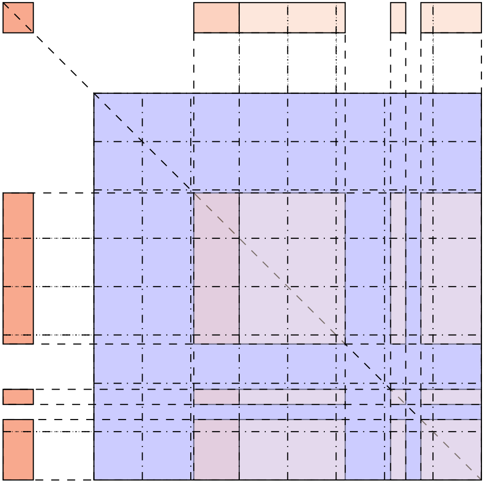
To do

- Distributed schur complement;
- Integrate Astrid Casadei work on memory consumption:
  - Allocate data only when required;
  - Drop coupling data:
    - Adapted data structure;
    - Mix left/right-looking algorithm.

Temporary buffer optimization



Temporary buffer optimization



2
MaPHYS - MPI+threads parallelization

Motivations

Goal: solving $\mathcal{A}x = b$, where A is sparse

Usual trades off

Direct

- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large $3D$ problems

Iterative

- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster

Motivations

Goal: solving $\mathcal{A}x = b$, where A is sparse

Usual trades off

Direct

- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large $3D$ problems

Iterative

- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster

Motivations

Goal: solving $\mathcal{A}x = b$, where A is sparse

Usual trades off

Direct

- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large $3D$ problems

Iterative

- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster

Motivations

Goal: solving $\mathcal{A}x = b$, where A is sparse

Usual trades off

Direct

- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large $3D$ problems

Iterative

- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster

Motivations

Goal: solving $\mathcal{A}x = b$, where A is sparse

Usual trades off

Direct

- Robust/accurate for general problems
- BLAS-3 based implementations
- Memory/CPU prohibitive for large $3D$ problems

Iterative

- Problem dependent efficiency / accuracy
- Sparse computational kernels
- Less memory requirements and possibly faster

Sparse hybrid (direct/iterative) Schur complement solvers based on domain decomposition

The main idea behind these methods is to permute the linear problem $\mathcal{A}x = b$ in a form:

$$\begin{pmatrix} \mathcal{A}_{\mathcal{I}\mathcal{I}} & \mathcal{A}_{\mathcal{I}\Gamma} \\ \mathcal{A}_{\Gamma\mathcal{I}} & \mathcal{A}_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} x_{\mathcal{I}} \\ x_{\Gamma} \end{pmatrix} = \begin{pmatrix} b_{\mathcal{I}} \\ b_{\Gamma} \end{pmatrix}$$

where $\mathcal{A}_{\mathcal{I}\mathcal{I}}$ is a blog diagonal matrix. Eliminating $x_{\mathcal{I}}$ from the second block row leads to the reduced system

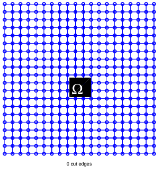
$$\mathcal{S}x_{\Gamma} = f$$

where

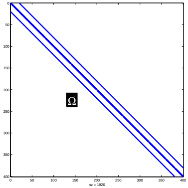
$$\mathcal{S} = \mathcal{A}_{\Gamma\Gamma} - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I}\Gamma} \quad \text{and} \quad f = b_{\Gamma} - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}b_{\mathcal{I}}.$$

Method used in MaPHyS

- Partitioning the global matrix in several local matrices



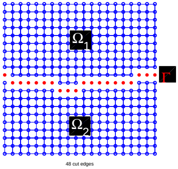
- Local factorization



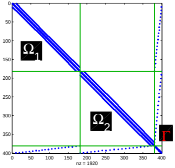
- Constructing of the preconditioner
- Solving the reduced system

Method used in MaPHyS

- Partitioning the global matrix in several local matrices



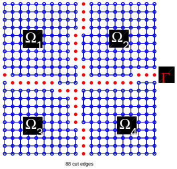
- Local factorization



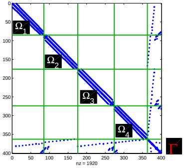
- Constructing of the preconditioner
- Solving the reduced system

Method used in MaPHyS

- Partitioning the global matrix in several local matrices
- Local factorization



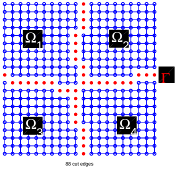
- Constructing of the preconditioner



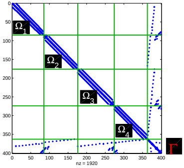
- Solving the reduced system

Method used in MaPHyS

- Partitioning the global matrix in several local matrices
  - Metis [G. Karypis and V. Kumar]
  - Scotch [Pellegrini and al.]
- Local factorization

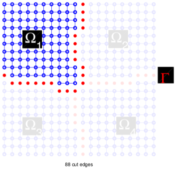


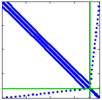
- Constructing of the preconditioner
- Solving the reduced system



Method used in MaPHyS

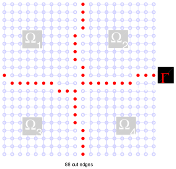
- Partitioning the global matrix in several local matrices
  - Metis [G. Karypis and V. Kumar]
  - Scotch [Pellegrini and al.]
- Local factorization
  - Mumps [P. Amestoy and al.] (with Schur option)
  - PaStiX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
- Solving the reduced system

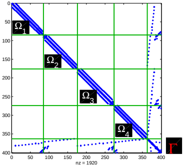




Method used in MaPHyS

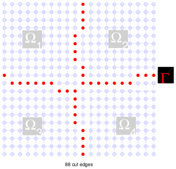
- Partitioning the global matrix in several local matrices
  - Metis [G. Karypis and V. Kumar]
  - Scotch [Pellegrini and al.]
- Local factorization
  - Mumps [P. Amestoy and al.] (with Schur option)
  - PaStiX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
  - Mkl library
- Solving the reduced system

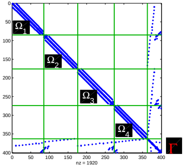




Method used in MaPHyS

- Partitioning the global matrix in several local matrices
  - Metis [G. Karypis and V. Kumar]
  - Scotch [Pellegrini and al.]
- Local factorization
  - Mumps [P. Amestoy and al.] (with Schur option)
  - PaStiX [P. Ramet and al.] (with Schur option)
- Constructing of the preconditioner
  - Mkl library
- Solving the reduced system
  - Cg/Gmres/FGmres [V.Fraysse and L.Giraud] using Mkl library for the reduced system





Software used in MaPHyS (before)

Partitioners

- Scotch
- Metis

Dense direct solver

- Mkl library

Sparse direct solvers

- Mumps
- PaStiX

Iterative Solvers

- Cg/Gmres/FGmres using Mkl library

Software used in MaPHyS (this study)

Dense direct solver

- Multi-threaded Mkl library

Sparse direct solvers

- Mumps
- Multi-threaded PaStiX

Iterative Solvers

- Cg/Gmres/FGmres using multi-threaded Mkl library

Software used in MaPHyS (this study)

Dense direct solver

- Multi-threaded Mkl library

Sparse direct solvers

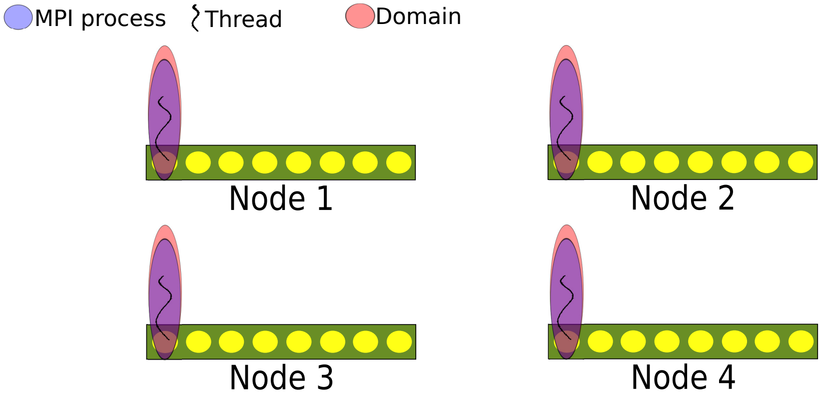
- Mumps
- Multi-threaded PaStiX

Iterative Solvers

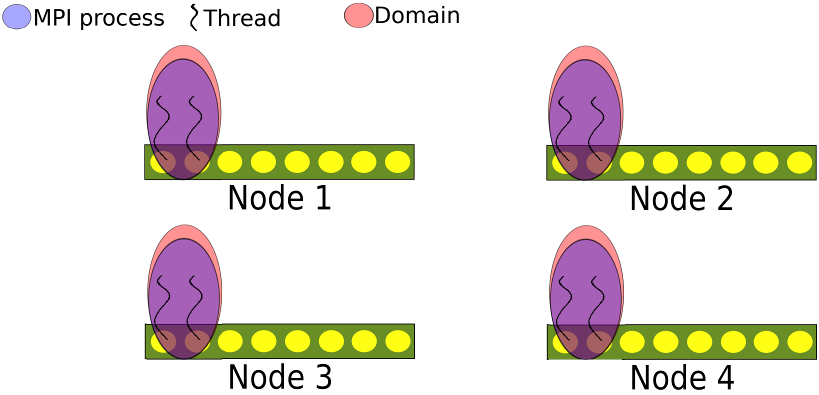
- Cg/Gmres/FGmres using multi-threaded Mkl library

- Challenge
  - Composability
  - Performance

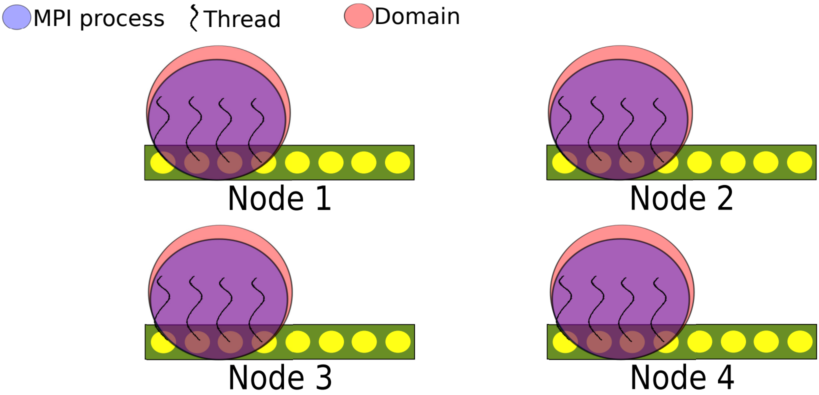
Scalability of MaPHyS on multicore nodes



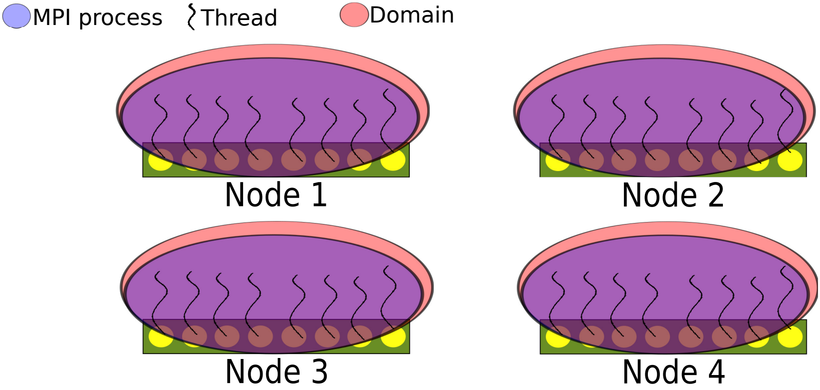
Scalability of MaPHyS on multicore nodes



Scalability of MaPHyS on multicore nodes



Scalability of MaPHyS on multicore nodes



Experimental set up

Hardware (on each node)

- Two Quad-core Nehalem Intelő Xeonő X5550
- Memory: 24 GB GDDR3
- Double precision

Matrices

Matrix	Matrix211k	Audi	Nachos	Haltere	Amande
Id	1	2	3	4	5
N	801K	943K	1,120K	1,288K	6,994K
Nnz	129,4M	39,29M	39,9M	10,47M	58,47M

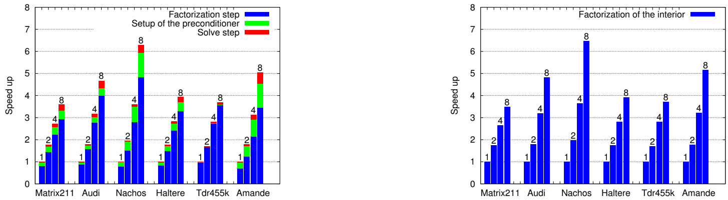
Table: Overview of sparse matrices used in this study.

Scalability of MaPHyS on multicore nodes

Achieved performance on four nodes with dense preconditioner

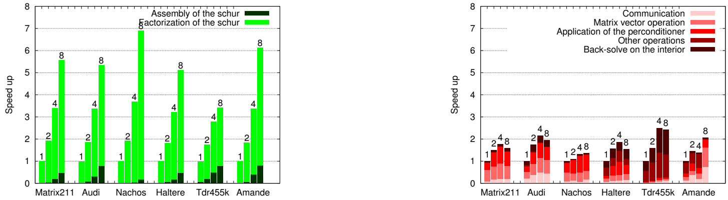
All computational steps

Factorization step

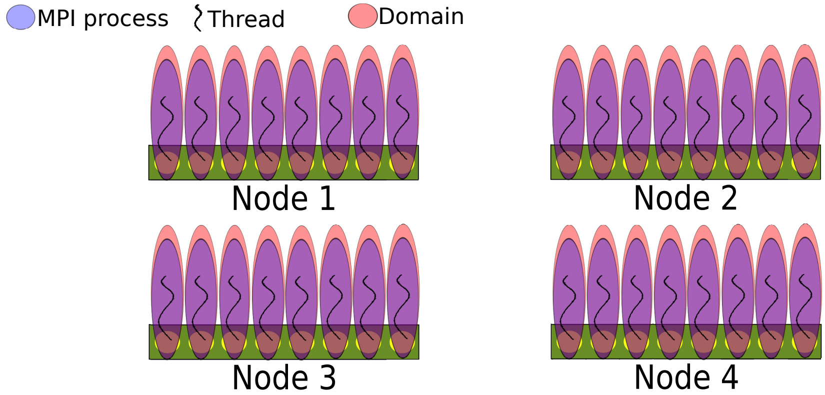


Preconditioning step

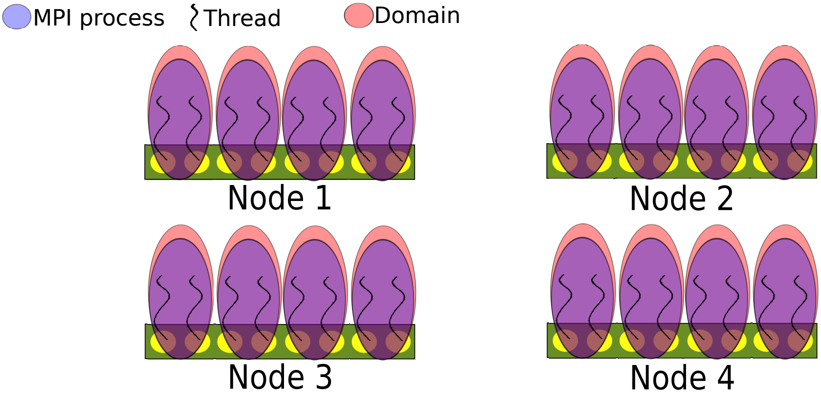
Solve step



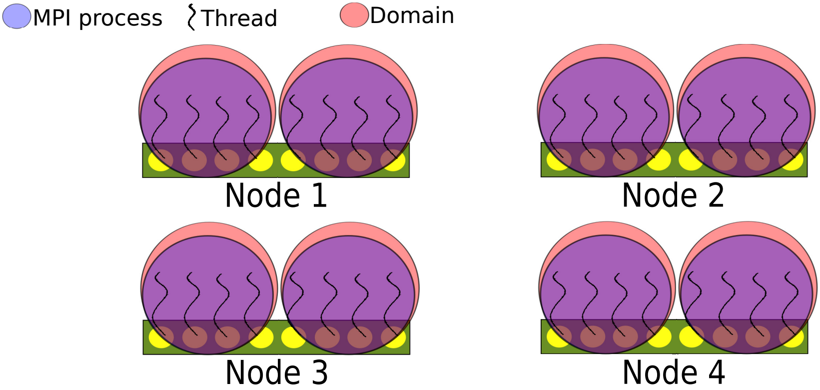
Flexibility to exploit entire multicore nodes



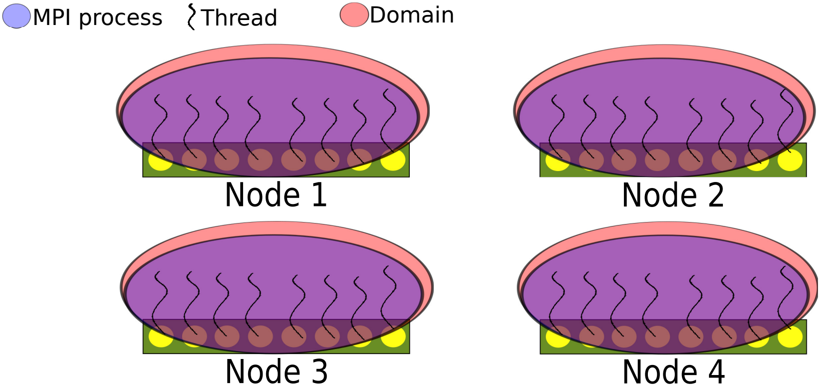
Flexibility to exploit entire multicore nodes



Flexibility to exploit entire multicore nodes



Flexibility to exploit entire multicore nodes



Experimental set up

Hardware (on each node)

- Two Quad-core Nehalem Intelő Xeonő X5550
- Memory: 24 GB GDDR3
- Double precision

Matrices

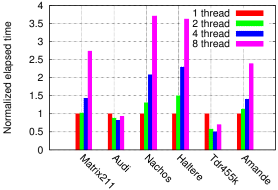
Matrix	Matrix211	Audi	Nachos	Haltere	Tdr455k	Amande
Nb.	1	2	3	4	5	6
N	801K	943K	1,120K	1,288K	2,738K	6,994K
Nnz	129,4M	39,29M	39,9M	10,47M	112,7M	58,47M
Nb_nodes	8	4	16	4	8	16
Preconditioner	dense	sparse03	sparse02	sparse03	dense	sparse03

Table: Overview of sparse matrices used in this study.

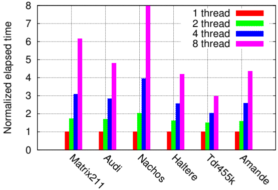
Flexibility to exploit entire multicore nodes

Achieved performance when all cores are used

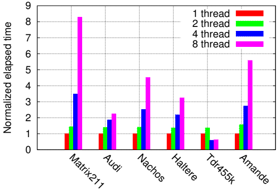
All computational steps



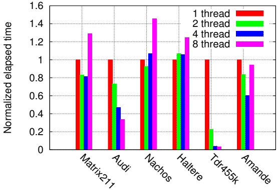
Factorization step



Preconditioning step



Solve step



Experimental set up

Hopper platform (Hardware)

- Two twelve-core AMD 'MagnyCours' 2.1-GHz
- Memory: 32 GB GDDR3
- Double precision

Matrices

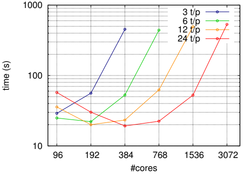
Matrix	Tdr455K	Nachos4M
Nb.	1	2
N	2,738K	4,147K
Nnz	112,7M	256,4M
Preconditioner	dense	sparse02

Table: Overview of sparse matrices used on the Hopper platform.

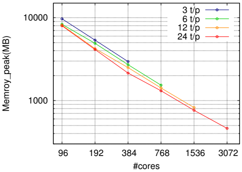
Results on the Hopper platform.

Achieved performance for the Tdr455K matrix

All computational steps

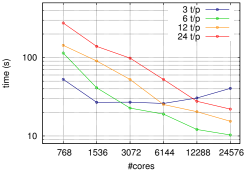


Memory used per node

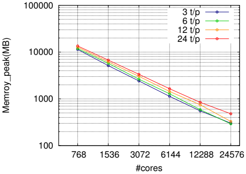


Achieved performance for the Nachos4M matrix.

All computational steps



Memory used per node



PaStiX

- Recent developpements:
  - Heterogeonous direct sparse linear solver using generic runtime;
  - Schur complement computation
  - Generic finite element oriented matrix assembly interface.
- Current developpements:
  - Distributed heterogeonous direct sparse linear solver;
  - Distributed Schur and optimizations;
  - Low-rank compression - H-Matrix.
  - Rewritten code for better upgradability, CMake compilation tools.

MaPHyS

- Recent study:
  - With the two-level parallelism we are able to efficiently exploits multicore architectures
  - Presentation of Julien Pedron (tomorrow)
- Current work:
  - Comparison with the PDSLin solver
  - Further experiments in the collaboration with TOTAL
  - Full task-based hybrid prototype

Thanks !

Xavier LACOSTE
Stojce NAKOV
INRIA HiePACS team
July 10, 2014