Task-based linear solvers for modern architectures

E. Agullo, P. Ramet (and HiePACS team)

7th ITER International School,
High Performance Computing in Fusion Science,
Aix-en-Provence

E. Agullo, P. Ramet
HiePACS team
Inria Bordeaux Sud-Ouest
LaBRI Bordeaux University

August 28, 2014

Guideline

Guideline

Introduction

Sparse direct factorization - PaStiX

Sparse solver on heterogenous architectures

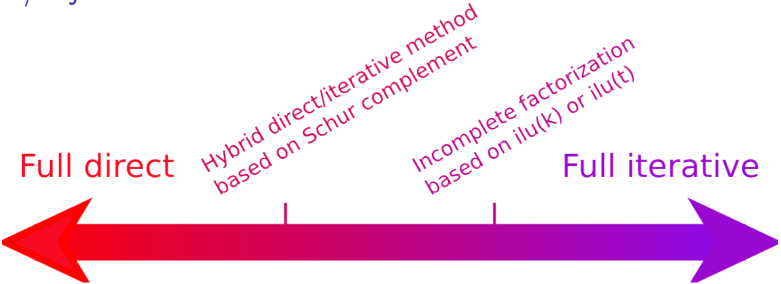
Hybrid methods - HIPS and MaPHYS

Low-rank compression - $\mathcal{H}$-PaStiX

Conclusion

1
Introduction

Mixed/Hybrid direct-iterative methods



The "spectrum" of linear algebra solvers

- Robust/accurate for general problems
- BLAS-3 based implementation
- Memory/CPU prohibitive for large 3D problems
- Limited parallel scalability

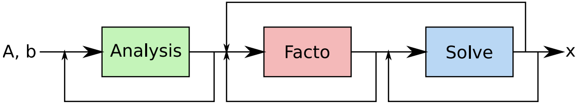
- Problem dependent efficiency/controlled accuracy
- Only mat-vec required, fine grain computation
- Less memory consumption, possible trade-off with CPU
- Attractive "build-in" parallel features

Guideline

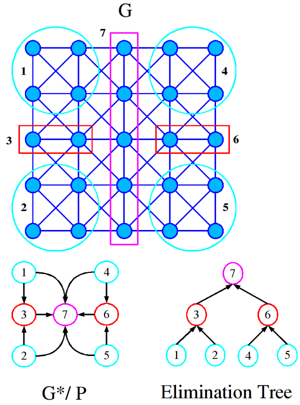
2
Sparse direct factorization - PaStiX

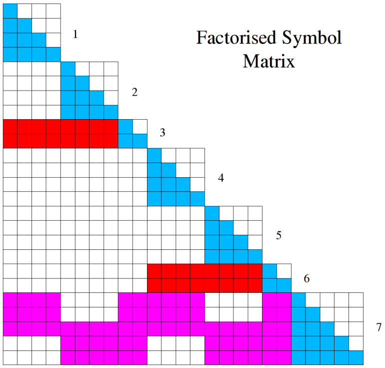
Major steps for solving sparse linear systems

1. Analysis: matrix is preprocessed to improve its structural properties ($A'x' = b'$ with $A' = P_n P D_r A D_c Q P^T$)
2. Factorization: matrix is factorized as $A = LU$, LL^T or LDL^T
3. Solve: the solution x is computed by means of forward and backward substitutions



Direct Method and Nested Dissection





Supernodal methods

Definition

A *supernode* (or supervariable) is a set of contiguous columns in the factors **L** that share essentially the same sparsity structure.

- All algorithms (ordering, symbolic factor., factor., solve) generalized to block versions.
- Use of efficient matrix-matrix kernels (improve cache usage).
- Same concept as *supervariables* for elimination tree/minimum degree ordering.
- Supernodes and pivoting: pivoting inside a supernode does not increase fill-in.

PaStiX main Features

- LLt, LDLt, LU : supernodal implementation (BLAS3)
- Static pivoting + Refinement: CG/GMRES/BiCGstab
- column-block or block mapping
- Simple/Double precision + Float/Complex operations
- **MPI/Threads (Cluster/Multicore/SMP/NUMA)**
- **Multiple GPUs using DAG runtimes**
- Support external ordering library (PT-Scotch or METIS ...)
- Multiple RHS (direct factorization)
- Incomplete factorization with ILU(k) preconditionner
- Schur complement computation
- C/C++/Fortran/Python/PETSc/Trilinos/FreeFem...

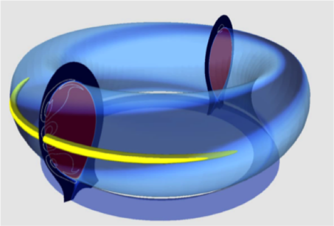
Current works

- Astrid Casadei (PhD student) : memory optimization to build a Schur complement in PaStiX, tight coupling between sparse direct and iterative solvers (HIPS) + **graph partitioning with balanced halo**
- Xavier Lacoste (PhD student) and the MUMPS team : **GPU optimizations for sparse factorizations** with StarPU
- Stojce Nakov (PhD student) : tight coupling between sparse direct and iterative solvers (MaPHYS) + **GPU optimizations for GMRES** with StarPU
- Mathieu Faverge (Assistant Professor) : **redesigned** PaStiX static/dynamic scheduling with PaRSEC in order to get a generic framework for multicore/MPI/GPU/Out-of-Core + **compression**

Direct Solver Highlights

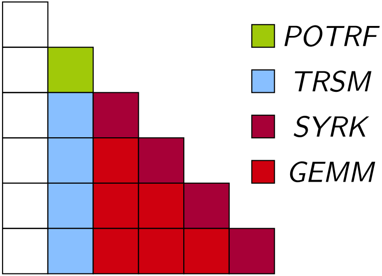
Fusion - ITER

PaStiX is used in the JOREK code developped by G. Huysmans at CEA/Cadarache in a fully implicit time evolution scheme for the numerical simulations of the ELM (Edge Localized Mode) instabilities commonly observed in the standard tokamak operating scenario.

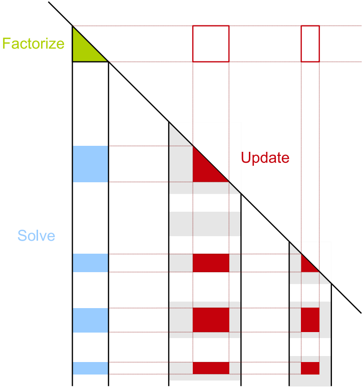


MHD pellet injection simulated with the JOREK code

Tasks algorithms

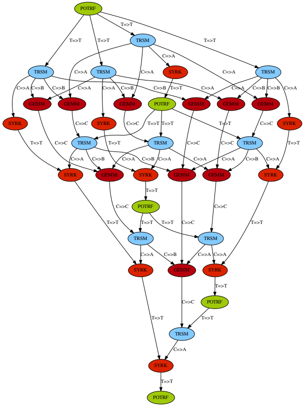


(a) Task for a dense tile

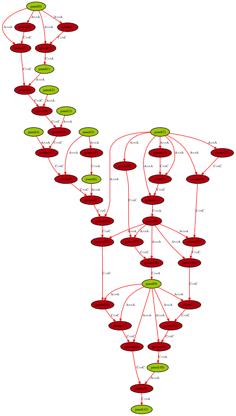


(b) Task for a sparse supernode

DAG representation



(c) Dense DAG



(d) Sparse DAG

Direct Solver Highlights (multicore)

SGI 160-cores

Name	N	NNZ_A	Fill ratio	Fact
Audi	9.44×10^5	3.93×10^7	31.28	float LL^T
10M	1.04×10^7	8.91×10^7	75.66	complex LDL^T

10M	10	20	40	80	160
Facto (s)	3020	1750	654	356	260
Mem (Gb)	122	124	127	133	146
Solve (s)	24.6	13.5	3.87	2.90	2.89

Audi	128	2x64	4x32	8x16
Facto (s)	17.8	18.6	13.8	**13.4**
Mem (Gb)	**13.4**	2x7.68	4x4.54	8x2.69
Solve (s)	0.40	0.32	0.21	**0.14**

Direct Solver Highlights (cluster of multicore)

RC3 matrix - complex double precision
N=730700 - NNA=41600758 - Fill-in=50

Facto	1 MPI	2 MPI	4 MPI	8 MPI
1 thread	6820	3520	1900	1890
6 threads	1020	639	337	287
12 threads	525	360	155	121
Mem Gb	1 MPI	2 MPI	4 MPI	8 MPI
1 thread	34	19,2	12,5	9,22
6 threads	34,3	19,5	12,8	9,66
12 threads	34,6	19,7	13	9,14
Solve	1 MPI	2 MPI	4 MPI	8 MPI
1 thread	6,97	3,75	1,93	1,03
6 threads	2,5	1,43	0,78	0,54
12 threads	1,33	0,93	0,66	0,59

Block ILU(k): supernode amalgamation algorithm

Derive a block incomplete LU factorization from the supernodal parallel direct solver

- Based on existing package PaStiX
- Level-3 BLAS incomplete factorization implementation
- Fill-in strategy based on level-fill among block structures identified thanks to the quotient graph
- **Amalgamation strategy to enlarge block size**

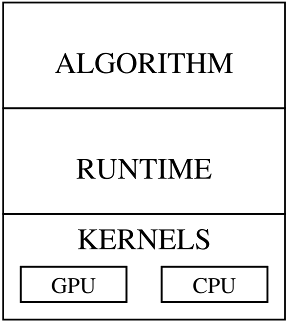
Highlights

- Handles efficiently high level-of-fill
- Solving time faster than with scalar ILU(k)
- Scalable parallel implementation

Guideline

3
Sparse solver on heterogenous architectures

Multiple layer approach



Governing ideas: Enable advanced numerical algorithms to be executed on a scalable unified runtime system for exploiting the full potential of future exascale machines.

Basics:

- Graph of tasks
- Out-of-order scheduling
- Fine granularity

DAG schedulers considered

StarPU

- RunTime Team – Inria Bordeaux Sud-Ouest
- C. Augonnet, R. Namyst, S. Thibault.
- Dynamic Task Discovery
- Computes cost models on the fly
- Multiple kernels on the accelerators
- Heterogeneous First-Time strategy

PaRSEC (formerly DAGuE)

- ICL – University of Tennessee, Knoxville
- G. Bosilca, A. Bouteiller, A. Danalys, T. Herault
- Parameterized Task Graph
- Only the most compute intensive kernel on accelerators
- Simple scheduling strategy based on computing capabilities
- GPU multi-stream enabled

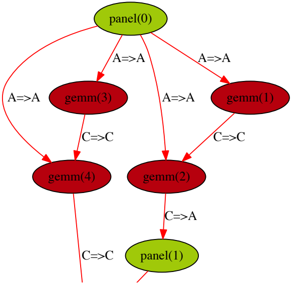
Supernodal sequential algorithm

```
forall the Supernode S1 do
    panel (S1);
    /* update of the panel                                          */
    forall the extra diagonal block Bi of S1 do
        S2 ← supernode_in_front_of (Bi);
        gemm (S1,S2);
        /* sparse GEMM B_{k,k≥i} × Bi^T substracted from
           S2                                                       */
    end
end
```

StarPU Tasks submission

```
forall the Supernode S1 do
    submit_panel (S1);
    /* update of the panel                                        */
    forall the extra diagonal block Bi of S1 do
        S2 ← supernode_in_front_of (Bi);
        submit_gemm (S1,S2);
        /* sparse GEMM B_{k,k≥i} × Bi^T subtracted from S2
            */
    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

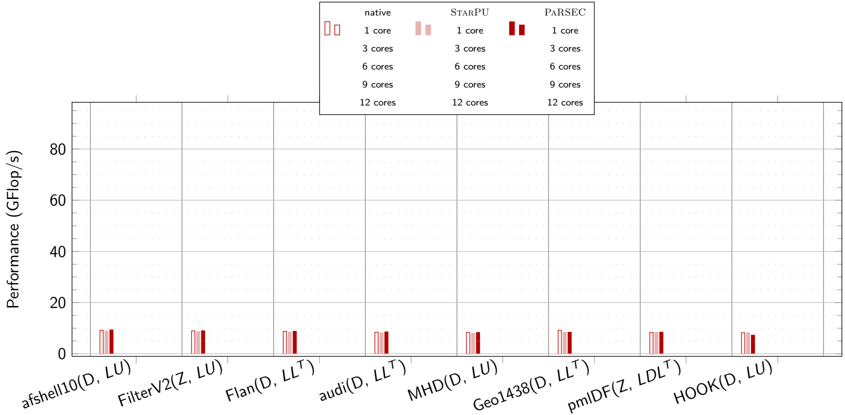
Matrices and Machines

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

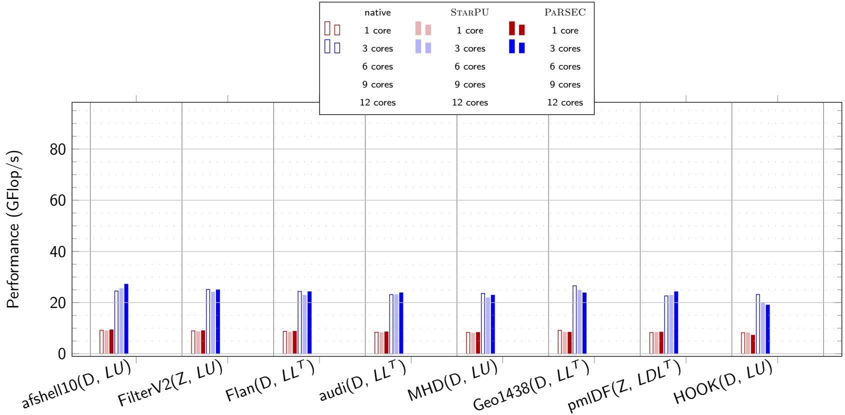
Table: Matrix description (Z: double complex, D: double).

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

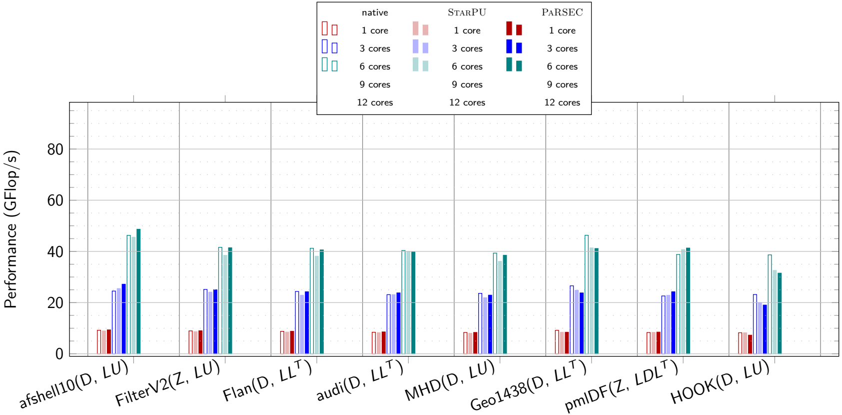
CPU scaling study: GFlop/s for numerical factorization



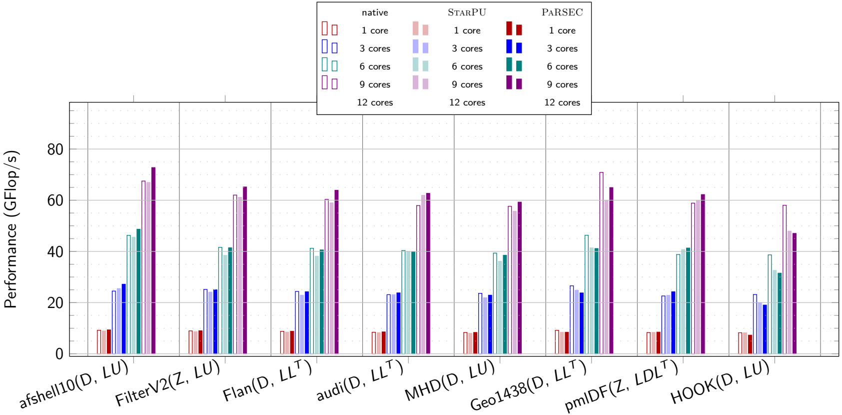
CPU scaling study: GFlop/s for numerical factorization



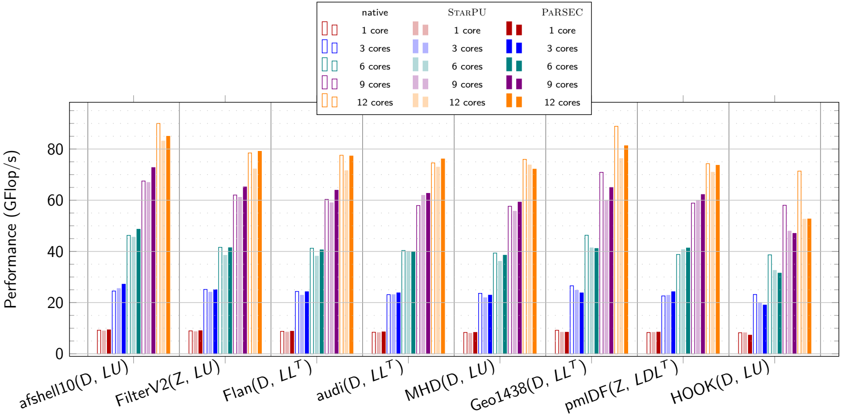
CPU scaling study: GFlop/s for numerical factorization



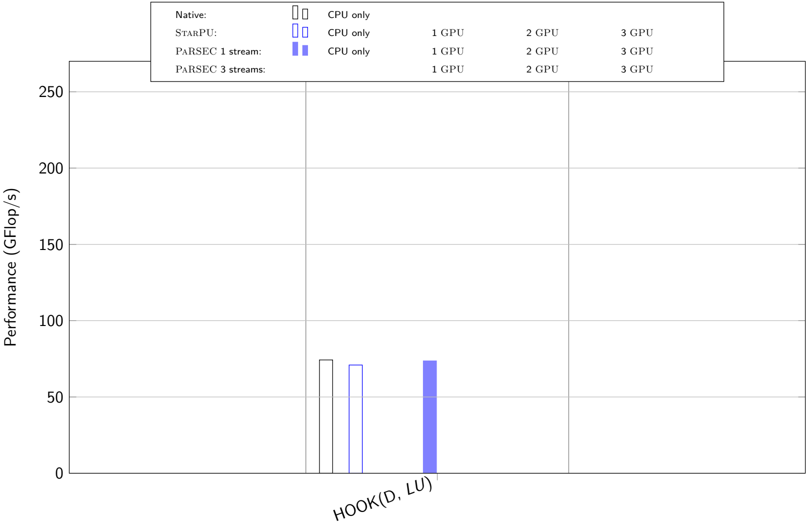
CPU scaling study: GFlop/s for numerical factorization



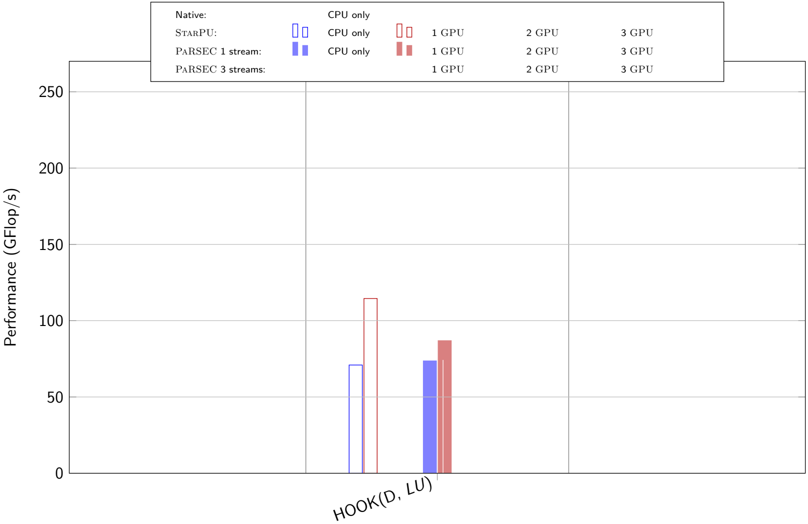
CPU scaling study: GFlop/s for numerical factorization



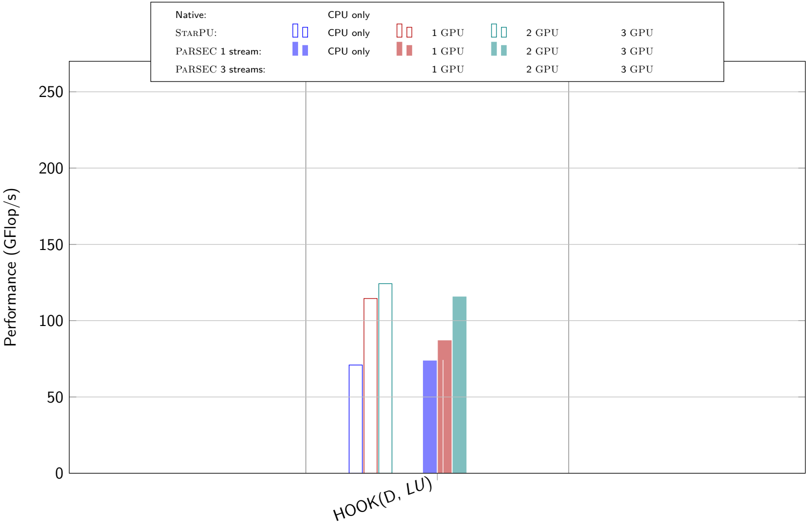
GPU scaling study : GFlop/s for numerical factorization



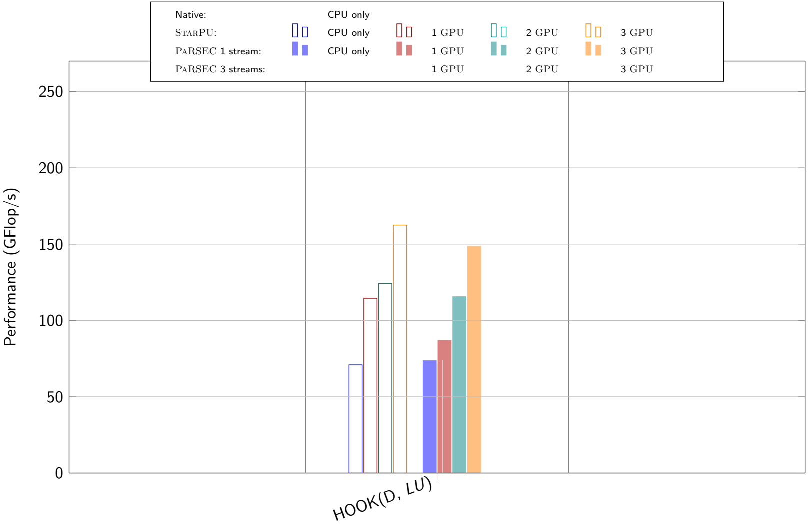
GPU scaling study : GFlop/s for numerical factorization



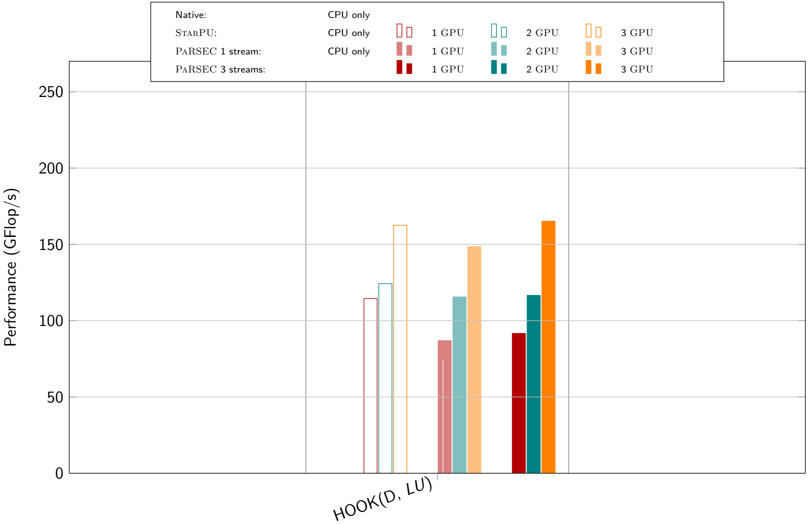
GPU scaling study : GFlop/s for numerical factorization



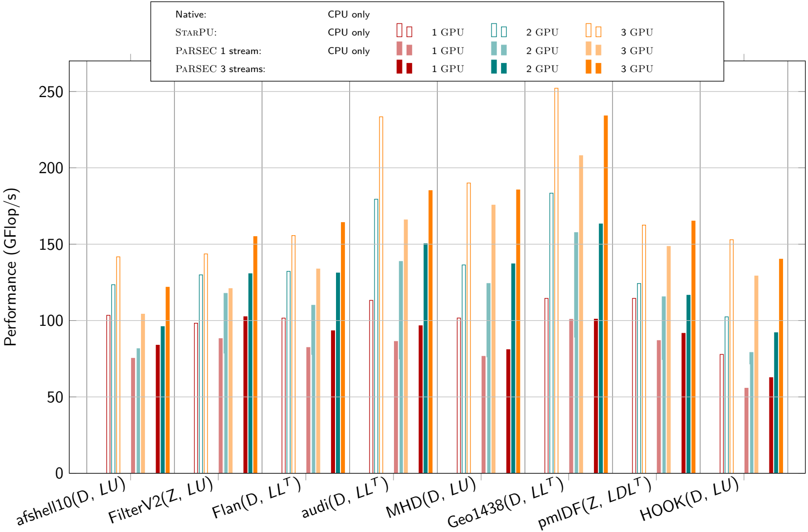
GPU scaling study : GFlop/s for numerical factorization



GPU scaling study : GFlop/s for numerical factorization



GPU scaling study : GFlop/s for numerical factorization



Xeon Phi (from Max-Planck-Institut for Plasmaphysic) [Phi=4CPUs, GPU=6CPUs]

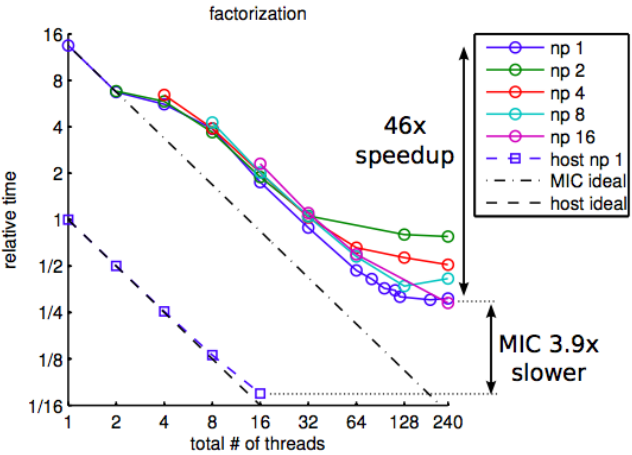


Fig. 21 Strong scaling of the “simple” PaStiX example

Guideline

4
Hybrid methods - HIPS and MaPHYS

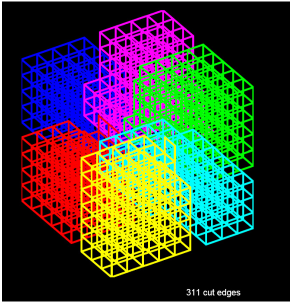
Hybrid Linear Solvers

Develop robust scalable parallel hybrid direct/iterative linear solvers

- Exploit the efficiency and robustness of the sparse direct solvers
- Develop robust parallel preconditioners for iterative solvers
- Take advantage of scalable implementation of iterative solvers

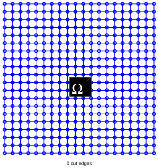
Domain Decomposition (DD)

- Natural approach for PDE's
- Extend to general sparse matrices
- Partition the problem into subdomains
- Use a direct solver on the subdomains
- Robust preconditioned iterative solver

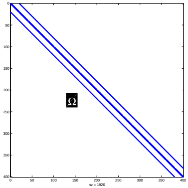


Method used in MAPHYS

- Partitioning the global matrix in several local matrices



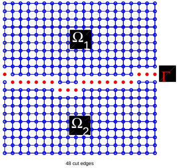
- Local factorization



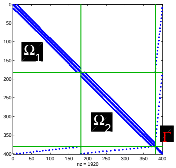
- Constructing of the preconditioner
- Solving the reduced system

Method used in MaPHYS

- Partitioning the global matrix in several local matrices



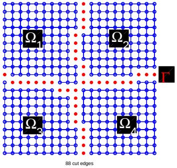
- Local factorization



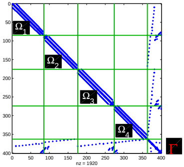
- Constructing of the preconditioner
- Solving the reduced system

Method used in MAPHYS

- Partitioning the global matrix in several local matrices



- Local factorization

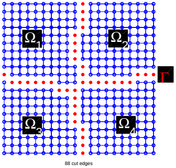


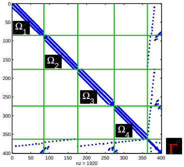
- Constructing of the preconditioner

- Solving the reduced system

Method used in MaPHYS

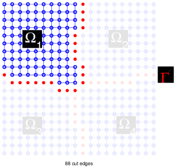
- Partitioning the global matrix in several local matrices
  - METIS [G. Karypis and V. Kumar]
  - SCOTCH [Pellegrini and al.]
- Local factorization
- Constructing of the preconditioner
- Solving the reduced system

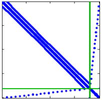




Method used in MaPHYS

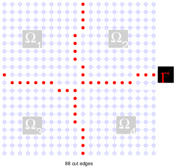
- Partitioning the global matrix in several local matrices
  - METIS [G. Karypis and V. Kumar]
  - SCOTCH [Pellegrini and al.]
- Local factorization
  - MUMPS [P. Amestoy and al.] (with Schur option)
  - PASTIX [P. Ramet and al.] (with Schur option and multi-threaded version)
- Constructing of the preconditioner
- Solving the reduced system

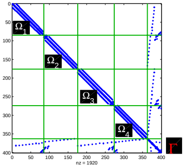




Method used in MAPHYS

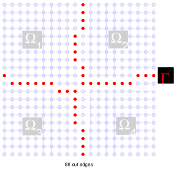
- Partitioning the global matrix in several local matrices
  - METIS [G. Karypis and V. Kumar]
  - SCOTCH [Pellegrini and al.]
- Local factorization
  - MUMPS [P. Amestoy and al.] (with Schur option)
  - PASTIX [P. Ramet and al.] (with Schur option and multi-threaded version)
- Constructing of the preconditioner
  - MKL library
- Solving the reduced system

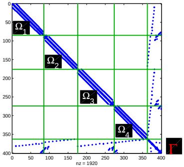




Method used in MaPHYS

- 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 and multi-threaded version)
- Constructing of the preconditioner
  - Mkl library
- Solving the reduced system
  - Cg/Gmres/FGmres on the reduced system





Experimental set up

Hopper platform (Hardware)

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

Matrices

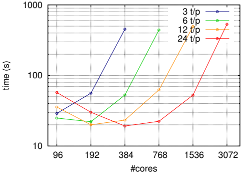
Matrix	Tdr455K	Nachos4M
N	2,738K	4,147K
Nnz	112,7M	256,4M

Table: Overview of sparse matrices used on the Hopper platform

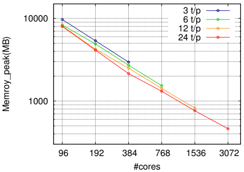
Results on the Hopper platform

Achieved performance for the Tdr455K matrix

All computational steps

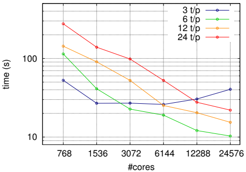


Memory used per node

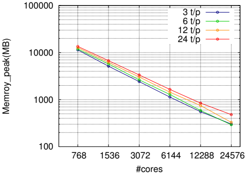


Achieved performance for the Nachos4M matrix

All computational steps



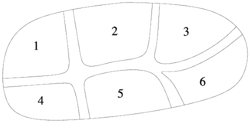
Memory used per node



HIPS : hybrid direct-iterative solver

Based on a **domain decomposition** : interface one node-wide (no overlap in DD lingo)

$$\begin{pmatrix} A_B & F \\ E & A_C \end{pmatrix}$$

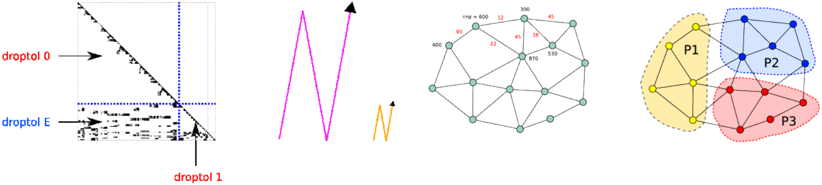


B : Interior nodes of subdomains (direct factorization).

C : Interface nodes.

Special decomposition and ordering of the subset C :
Goal : Building a global Schur complement preconditioner (ILU) from the local domain matrices only.

HIPS: preconditioners



Main features

- Iterative or "hybrid" direct/iterative method are implemented.
- Mix direct supernodal (BLAS-3) and sparse ILUT factorization in a seamless manner.
- Memory/load balancing : distribute the domains on the processors (domains > processors).

Guideline

5
Low-rank compression - $\mathcal{H}$-PaStiX

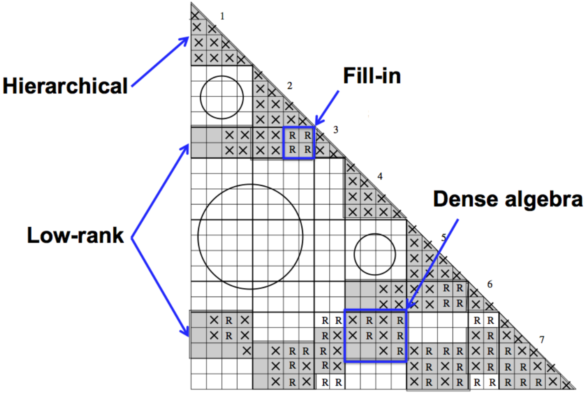
Toward low rank compressions in supernodal solver

Many works on hierarchical matrices and direct solvers

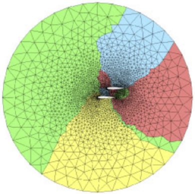
- Eric Darve : Hierarchical matrices classifications (*Building O(N) Linear Solvers Using Nested Dissection*)
- Sherry Li : Multifrontal solver + HSS (*Towards an Optimal-Order Approximate Sparse Factorization Exploiting Data-Sparseness in Separators*)
- David Bindel : CHOLMOD + Low Rank (*An Efficient Solver for Sparse Linear Systems Based on Rank-Structured Cholesky Factorization*)
- Jean-Yves L'Excellent : MUMPS + Block Low Rank

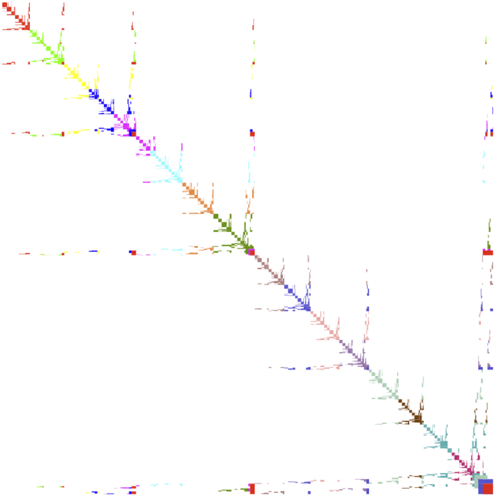
Symbolic factorization

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \Rightarrow \begin{bmatrix} LU & L^{-1}B \\ CU^{-1} & D - CA^{-1}B \end{bmatrix}$$



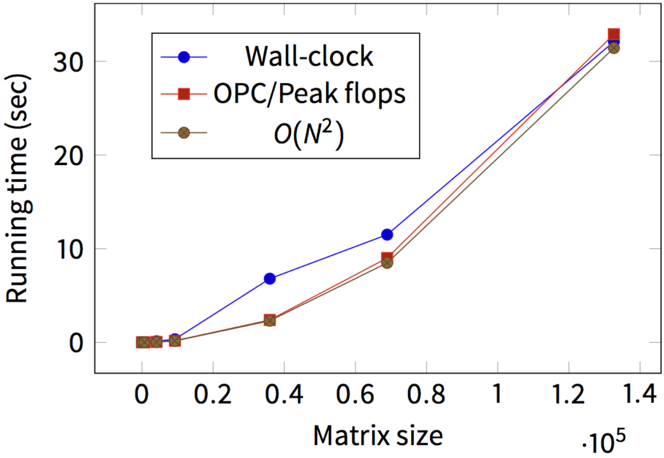
Nested dissection, 2D mesh/matrix





Computational cost

Cost grows like $O(N^2)$ for 3D PDE matrix. Benchmarks obtained using PaStiX (Pierre Ramet).



FastLA associate team between INRIA/Berkeley/Stanford

Supernodal Solver - Hierarchical Matrices $O(N.log^a(N))$

1. Check the potential compression ratio on top level blocks
2. Develop a prototype with:
   - low-rank compression on the larger supernodes
   - compression tree built at each update
   - complexity analysis of the approach
3. Study coupling between nested dissection and compression tree ordering

Which algorithm to find low-rank approximation ?

SVD, RR-LU, RR-QR, ACA, CUR, Random ...

Which family of hierarchical matrix ?

$\mathcal{H}$, $\mathcal{H}^2$, HODLR ...

Guideline

6
Conclusion

Softwares

Graph/Mesh partitioner and ordering :

http://scotch.gforge.inria.fr

Sparse linear system solvers :

http://pastix.gforge.inria.fr

http://hips.gforge.inria.fr

https://wiki.bordeaux.inria.fr/maphys/doku.php

Softwares

Fast Multipole Method :

SCALFMM

C++ Fast Multipole Method Library for HPC

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

Matrices Over Runtime Systems (with University of Tenessee):

MORSE

http://icl.cs.utk.edu/projectsdev/morse

Thank You

Pierre Ramet
HiePACS
http://www.labri.fr/ ramet