From hybrid architectures to hybrid solvers

C2S@Exa Kickoff meeting - Nuclear Fusion

X. Lacoste, M. Faverge, P. Ramet

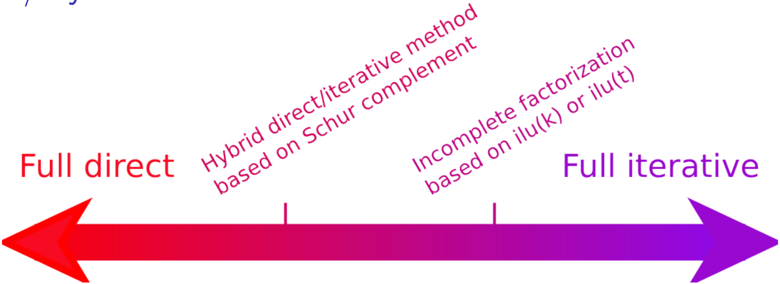
Pierre RAMET
HiePACS team
Inria Bordeaux Sud-Ouest

May 17, 2013

Guideline

1
Context and goals

Mixed/Hybrid direct-iterative methods



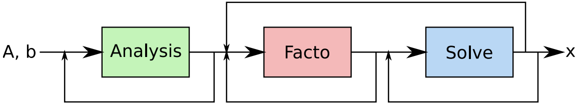
The "spectrum" of linear algebra solvers

- 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

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

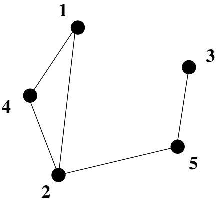


Symmetric matrices and graphs

- Assumptions: **A** symmetric, pivots are chosen on the diagonal
- Structure of **A** symmetric represented by the graph $G = (V, E)$
  - Vertices are associated to columns: $V = \{1, ..., n\}$
  - Edges E are defined by: $(i, j) \in E \leftrightarrow a_{ij} \neq 0$
  - G undirected (symmetry of **A**)
- Number of nonzeros in column $j = |Adj_G(j)|$
- Symmetric permutation $\equiv$ renumbering the graph

	1	2	3	4	5
1	×	×		×	
2	×	×		×	×
3			×		×
4	×	×		×	
5		×	×		×

Symmetric matrix



Corresponding graph

Fill-in theorem and Elimination tree

Theorem

Any $\mathbf{A}_{ij} = 0$ will become a non-null entry $\mathbf{L}_{ij}$ or $\mathbf{U}_{ij} \neq 0$ in $\mathbf{A} = \mathbf{LU}$ if and only if it exists a path in $G_A(V, E)$ from vertex i to vertex j that only goes through vertices with a lower number than i and j.

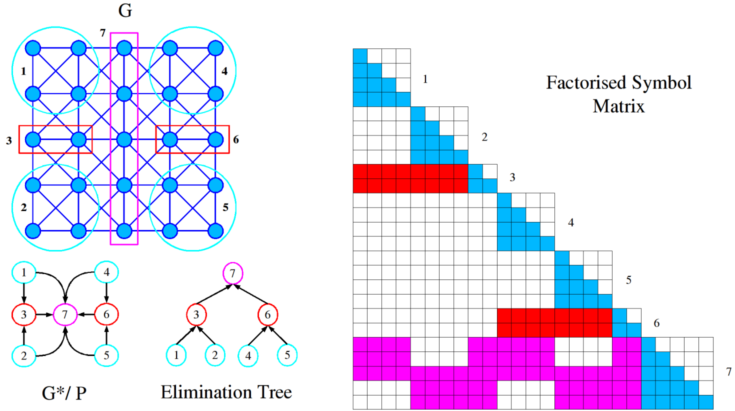
Definition

Let $\mathbf{A}$ be a symmetric positive-definite matrix, $G^+(\mathbf{A})$ is the **filled graph** (graph of $\mathbf{L} + \mathbf{L}^{\mathrm{T}}$) where $\mathbf{A} = \mathbf{LL}^{\mathrm{T}}$ (Cholesky factorization)

Definition

The **elimination tree** of $\mathbf{A}$ is a spanning tree of $G^+(\mathbf{A})$ satisfying the relation $PARENT[j] = min\{i > j | l_{ij} \neq 0\}$.

Direct Method



PaStiX Features

- LLt, LDLt, LU : supernodal implementation (BLAS3)
- Static pivoting + Refinement: CG/GMRES
- Simple/Double precision + Float/Complex operations
- Require only C + MPI + Posix Thread (PETSc driver)

- MPI/Threads (Cluster/Multicore/SMP/NUMA)
- **Dynamic scheduling NUMA (static mapping)**
- Support external ordering library (PT-Scotch/METIS)

- Multiple RHS (direct factorization)
- **Incomplete factorization with ILU(k) preconditionner**
- **Schur computation (hybrid method MaPHYS or HIPS)**
- Out-of Core implementation (shared memory only)

Direct Solver Highlights (MPI)

Main users

- Electomagnetism and structural mechanics at CEA-DAM
- MHD Plasma instabilities for ITER at CEA-Cadarache
- Fluid mechanics at Bordeaux

TERA CEA supercomputer

The direct solver PaStiX has been successfully used to solve a huge symmetric complex sparse linear system arising from a 3D electromagnetism code

- **45 millions unknowns**: required 1.4 Petaflops and was completed in half an hour on 2048 processors.
- **83 millions unknowns**: required 5 Petaflops and was completed in 5 hours on 768 processors.

2
Dynamic Scheduling

Dynamic Scheduling for NUMA and multicore architectures

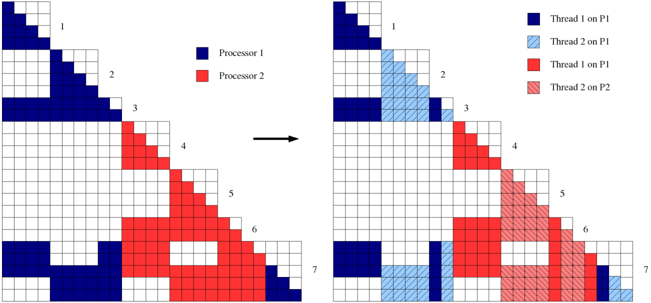
Needs

- Adapt to NUMA architectures
- Improve memory affinity (take care of memory hierarchy)
- Reduce idle-times due to I/O (communications and disk access in future works)

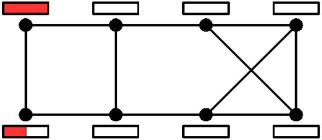
Proposed solution

- Based on a classical work stealing algorithm
- Stealing is limited to preserve memory affinity
- Use dedicated threads for I/O and communication in order to give them an higher priority

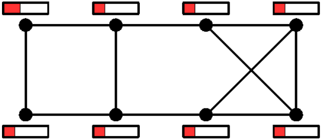
NUMA-Aware Allocation (up to 20% efficiency)



(a) Localization of new NUMA-aware allocation in the matrix

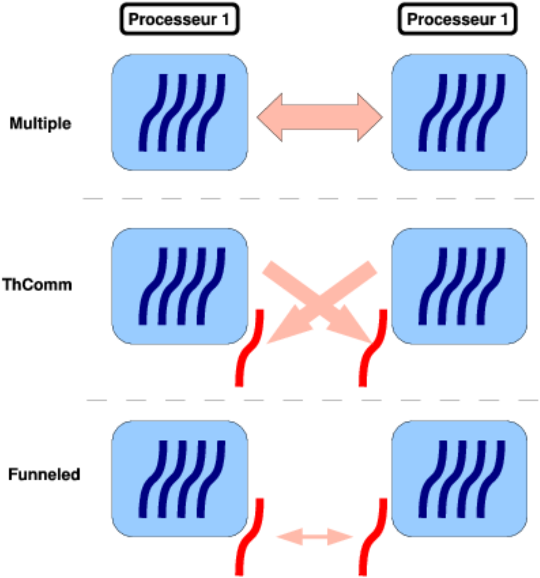


(b) Initial allocation



(c) New NUMA-aware allocation

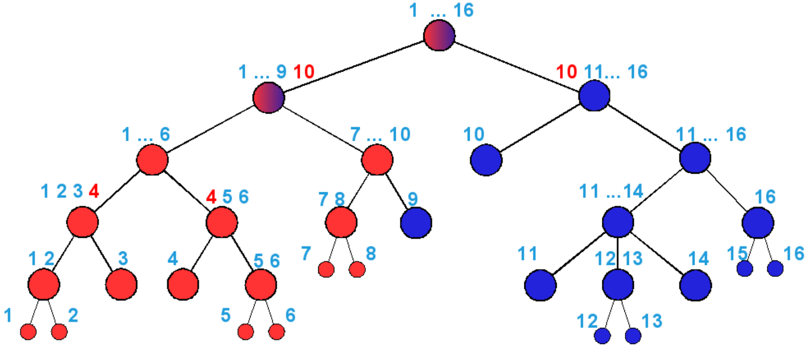
Communication schemes (upto 10% efficiency)



Thread support inside MPI librairies

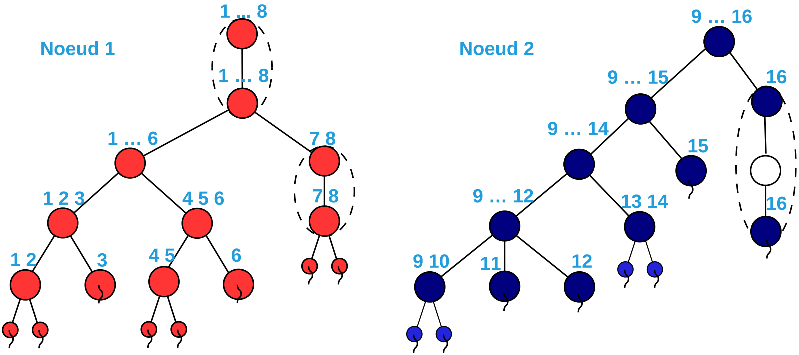
- MPI_THREAD_SINGLE
  - Only one thread will execute.
- MPI_THREAD_FUNNELED
  - The process may be multi-threaded, but only the main thread will make MPI calls
    (all MPI calls are funneled to the main thread).
- MPI_THREAD_SERIALIZED
  - The process may be multi-threaded, and multiple threads may make MPI calls, but only one at a time: MPI calls are not made concurrently from two distinct threads
    (all MPI calls are serialized).
- MPI_THREAD_MULTIPLE
  - Multiple threads may call MPI, with no restrictions.

Dynamic Scheduling : New Mapping



- Need to map data on MPI process
- Two steps :
  - **A first proportional mapping step to map data**
  - A second step to build a file structure for the work stealing algorithm

Dynamic Scheduling : New Mapping



- Need to map data on MPI process
- Two steps :
  - A first proportional mapping step to map data
  - **A second step to build a file structure for the work stealing algorithm**

Study on a large test case: 10M

Properties

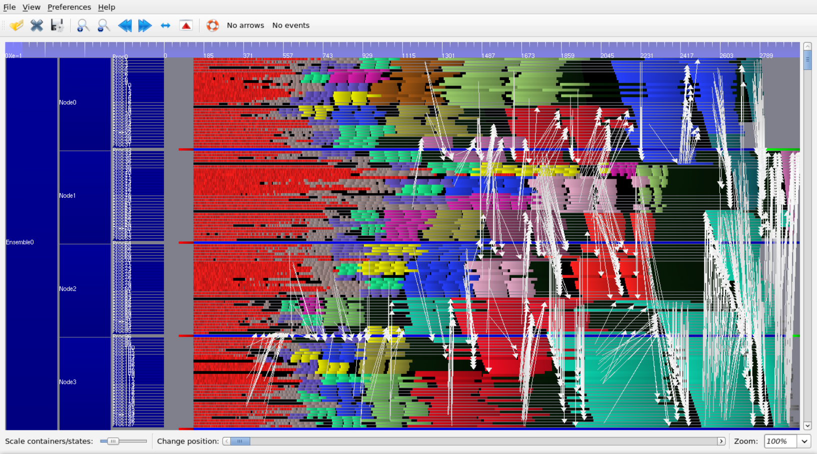
N	10 423 737
NNZ_A	89 072 871
NNZ_L	6 724 303 039
OPC	4.41834e+13

	4x32	8x32
Static Scheduler	289	195
Dynamic Scheduler	240	184

Table : Factorization time in seconds

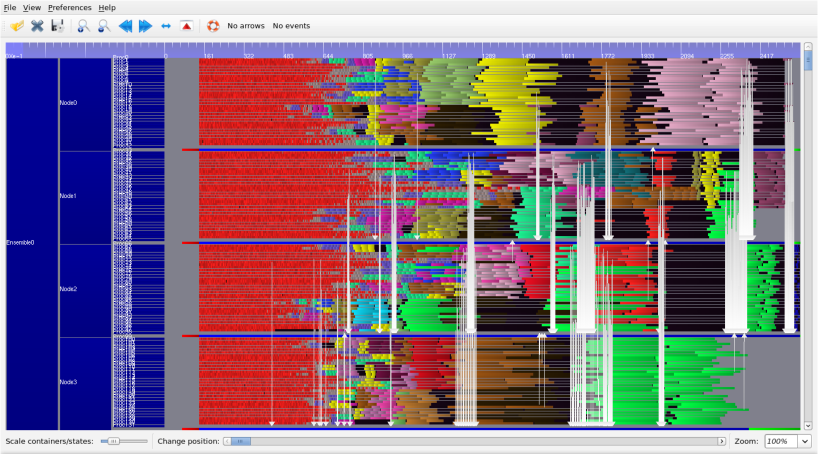
- Electromagnetism problem in double complex from CEA
- Cluster Vargas from IDRIS with 32 power6 per node

Static Scheduling Gantt Diagram



- *10Million* test case on IDRIS IBM Power6 with 4 MPI process of 32 threads (color is level in the tree)

Dynamic Scheduling Gantt Diagram



- Reduces time by 10-15%

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

Audi	8	64	128	2x64	4x32	8x16	160
Facto (s)	103	21.1	17.8	18.6	13.8	**13.4**	17.2
Mem (Gb)	11.3	12.7	**13.4**	2x7.68	4x4.54	8x2.69	14.5
Solve (s)	1.16	0.31	0.40	0.32	0.21	**0.14**	0.49

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

3
Generic Runtimes

Panel factorization

- Factorization of the diagonal block (xxTRF);
- TRSM on the extra-diagonal blocks (ie. solves $X \times b_d = b_{i,i>d}$ – where b_d is the diagonal block).

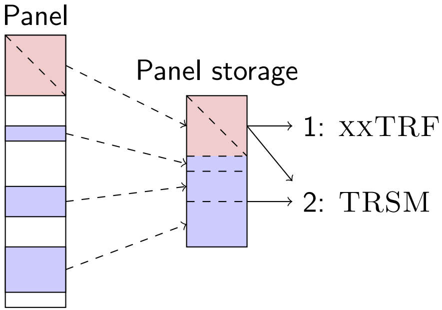


Figure : Panel update

Trailing supernodes update

- One global GEMM in a temporary buffer;
- Scatter addition (many AXPY).

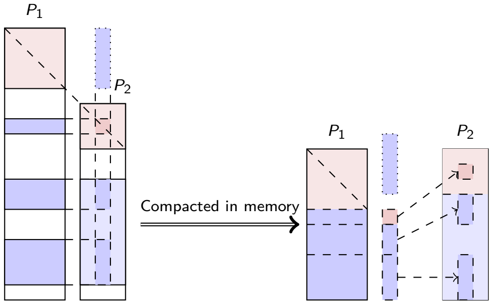


Figure : Panel update

GPU kernel performance

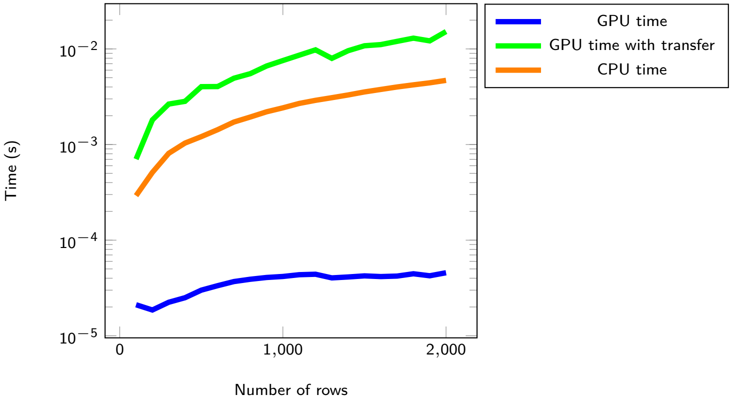


Figure : Sparse kernel timing with 100 columns.

Generic Runtimes

- Task-based programming model;
- Tasks scheduled on computing units (CPUs, GPUs, ...);
- Data transfers management;
- Dynamicaly build models for kernels;
- Add new scheduling strategies with plugins;
- Get informations on idle times and load balances.

StarPU Tasks submission

Algorithm 1: 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**
        $S_2 \leftarrow$ `supernode_in_front_of` (B_i);
        `submit_gemm` (S_1, S_2);
        `/* sparse` GEMM $B_{k,k \geq i} \times B_i^T$ `substracted from` S_2 `*/`
    end
end

DAGUE's parametrized taskgraph

```
panel(j) [high_priority = on]
/* execution space */
j = 0 .. cblknbr-1
/* Extra parameters */
firstblock = diagonal_block_of( j )
lastblock = last_block_of( j )
lastbrow = last_brow_of( j ) /* Last block generating an update on j */
/* Locality */
:A(j)
RW A ← leaf ? A(j) : C gemm(lastbrow)
       → A gemm(firstblock+1..lastblock)
       → A(j)
```

Figure : Panel factorization description in DAGUE

4
Results on Manycore Architectures

Matrices and Machines

Matrices

Name	N	NNZ_A	Fill ratio	OPC	Fact
MHD	4.86×10^5	1.24×10^7	61.20	9.84×10^{12}	Float LU
Audi	9.44×10^5	3.93×10^7	31.28	5.23×10^{12}	Float LL^T
10M	1.04×10^7	8.91×10^7	75.66	1.72×10^{14}	Complex LDL^T

Machines

Processors	Frequency	GPUs	RAM
AMD Opteron 6180 SE (4×12)	2.50 GHz	Tesla T20 ($\times 2$)	256

CPU only results on Audi

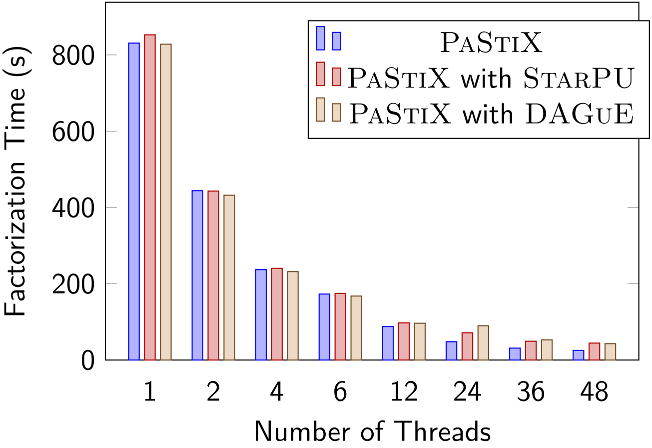


Figure : LL^T decomposition on Audi (double precision)

CPU only results on MHD

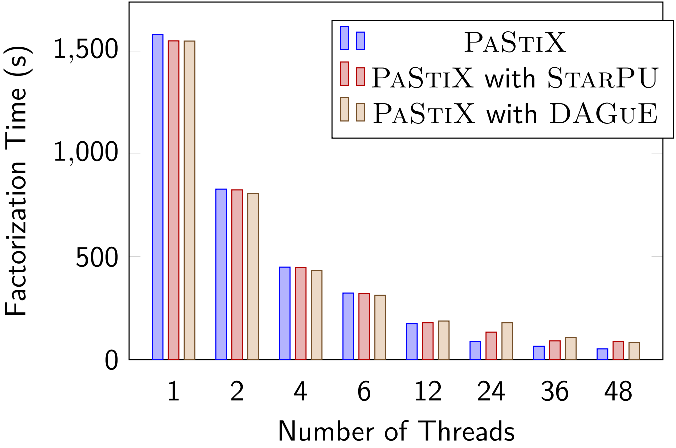


Figure : *LU* decomposition on MHD (double precision)

CPU only results on 10 Millions

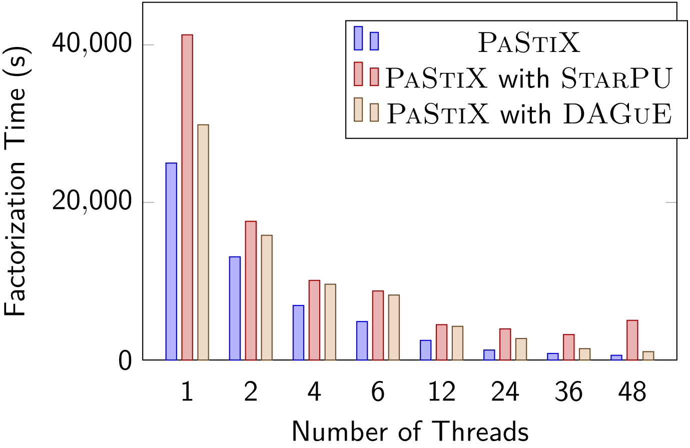


Figure : LDL^T decomposition on 10M (double complex)

Audi: GPU results on Romulus (STARPU)

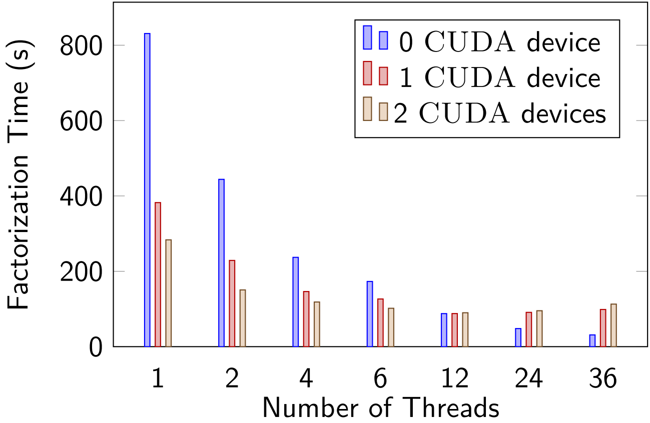


Figure : Audi LL^t decomposition with GPU (double precision)

MHD: GPU results on Romulus (StarPU)

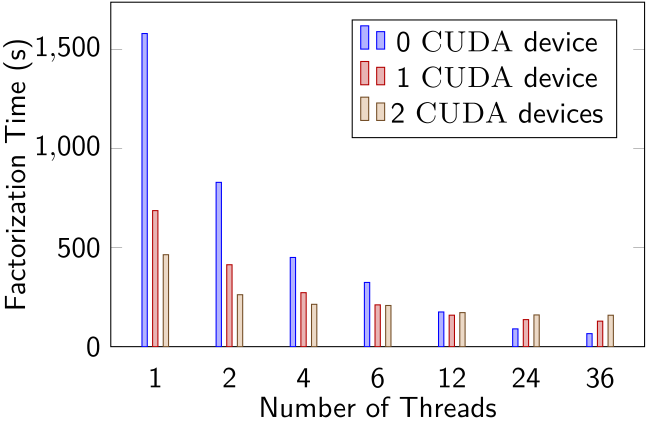


Figure : MHD *LU* decomposition with GPU (double precision)

5
Conclusion and extra tools

Conclusion

- Timing equivalent to PaStiX with medium size test cases;
- Quite good scaling;
- Speedup obtained with one GPU and little number of cores;
- released in PaStiX 5.2 (`http://pastix.gforge.inria.fr`).

Futur works

- Study the effect of the block size for GPUs;
- Write solve step with runtime;
- Distributed implementation (MPI);
- Panel factorization on GPU;
- Add context to reduce the number of candidates for each task;

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**

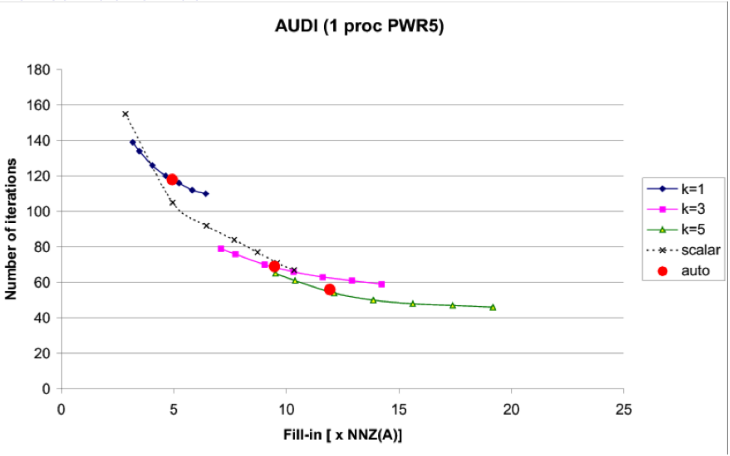
Highlights

- Handles efficiently high level-of-fill
- Solving time can be 2-4 faster than with scalar ILU(k)
- Scalable parallel implementation

Block ILU(k): some results on AUDI matrix

$(N = 943,695,\ NNZ = 39,297,771)$

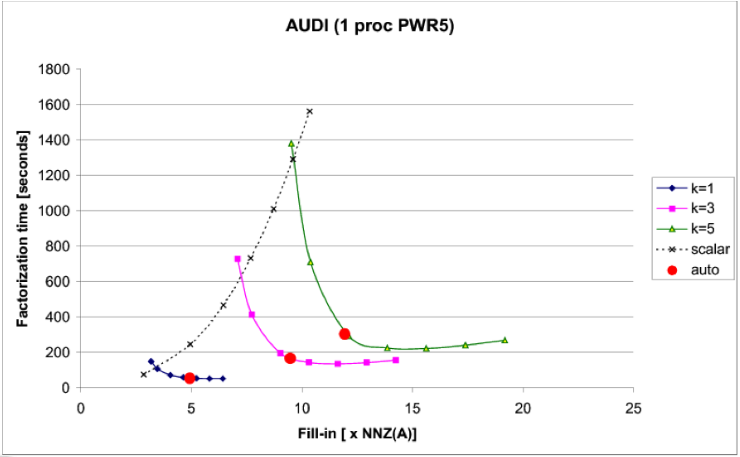
Numerical behaviour



Block ILU(k): some results on AUDI matrix

$(N = 943,695,\ NNZ = 39,297,771)$

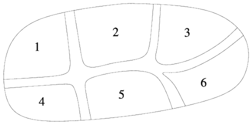
Preconditioner setup time



HIPS : hybrid direct-iterative solver

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

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

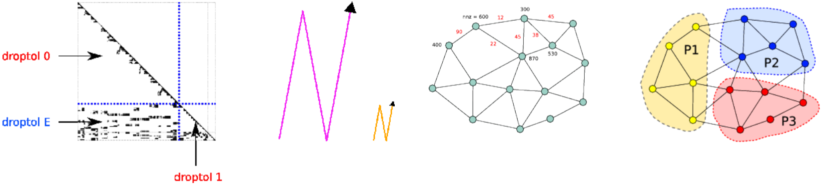


B : Interior nodes of subdomains (direct factorization).

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).

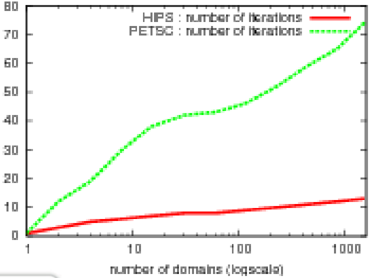
HIPS vs Additive Schwarz (from PETSc)

Experimental conditions

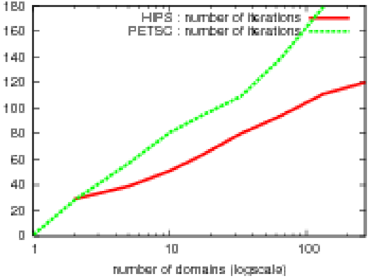
These curves compare HIPS (Hybrid) with Additive Schwarz from PETSc.

Parameters were tuned to compare the result with a very similar fill-in

Haltere



MHD



MURGE: a common API to the sparse linear solvers of BACCHUS

http://murge.gforge.inria.fr

Features

- Through one interface, access to many solver strategies
- Enter a graph/matrix in a centralized or distributed way
- Simple formats : coordinate, CSR or CSC
- Very easy to implement an assembly step

General structure of the code

```
MURGE_Initialize(idnbr, ierror)
MURGE_SetDefaultOptions(id, MURGE_ITERATIVE)  /* Choose general strategy */
MURGE_SetOptionInt(id, MURGE_DOF, 3)  /* Set degrees of freedom */
..
MURGE_Graph_XX(id..) /* Enter the graph : several possibilities */
DO
 MURGE_SetOptionReal(id, MURGE_DROPTOL1, 0.001) /* Threshold for ILUT */
 MURGE_SetOptionReal(id, MURGE_PREC, 1e-7)  /* Precision of solution */
 ...
 /** Enter new coefficient for the matrix **/
  MURGE_AssemblyXX(id..) /* Enter the matrix coefficients */
 DO
   MURGE_SetRHS(id, rhs) /* Set the RHS */
   MURGE_GetSol(id, x)   /* Get the solution */
 END
  MURGE_MatrixReset(id) /* Reset matrix coefficients */
END
MURGE_Clean(id)  /* Clean-up for system "id" */
MURGE_Finalize() /* Clean-up all remaining structure */
```

BACCHUS/HiePACS 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`

Thanks !

Pierre RAMET
INRIA HiePACS team

C2S@Exa, Nuclear Fusion, Sophia-Antipoli