Toward a supernodal sparse direct solver over DAG runtimes

CEMRA@CS’2012 Luminy

X. Lacoste, P. Ramet, M. Faverge, I. Yamazaki, G. Bosilca, J. Dongarra

Pierre RAMET
BACCHUS team
Inria Bordeaux Sud-Ouest
Guideline

Context and goals

Kernels

- Panel factorization
- Trailing supernodes update (CPU version)
- Sparse GEMM on GPU

Runtimes

Results

- Matrices and Machines
- Multicore results
- GPU results

Conclusion and extra tools
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
Possible solutions for Many-Cores

- Multi-Cores: PaStiX already finely tuned to use MPI and P-Threads;
- Multiple-GPU and many-cores, two solutions:
  - Manually handle GPUs:
    - lot of work;
    - heavy maintenance.
  - Use dedicated runtime:
    - May lose the performance obtained on many-core;
    - Easy to add new computing devices.

Elected solution, runtime:

- StarPU: RUNTIME – Inria Bordeaux Sud-Ouest;
- DAGuE: ICL – University of Tennessee, Knoxville.
Major steps for solving sparse linear systems

1. **Analysis:** matrix is preprocessed to improve its structural properties (\( A'x' = b' \) with \( A' = P_nPD_rAD_cQP^T \))

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: $\mathbf{A}$ symmetric, pivots are chosen on the diagonal
- Structure of $\mathbf{A}$ symmetric represented by the graph $G = (V, E)$
  - Vertices are associated to columns: $V = \{1, \ldots, n\}$
  - Edges $E$ are defined by: $(i, j) \in E \iff a_{ij} \neq 0$
  - $G$ undirected (symmetry of $\mathbf{A}$)

- Number of nonzeros in column $j = |\text{Adj}_G(j)|$
- Symmetric permutation $\equiv$ renumbering the graph
Theorem
Any $A_{ij} = 0$ will become a non-null entry $L_{ij}$ or $U_{ij} \neq 0$ in $A = LU$ 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 $A$ be a symmetric positive-definite matrix, $G^+(A)$ is the filled graph (graph of $L + L^T$) where $A = LL^T$ (Cholesky factorization).

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

Context and goals
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

- Electromagnetism 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.
## Direct Solver Highlights (multicore)

### SGI 160-cores

<table>
<thead>
<tr>
<th>Name</th>
<th>(N)</th>
<th>(\text{NNZ}_A)</th>
<th>Fill ratio</th>
<th>Fact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audi</td>
<td>(9.44 \times 10^5)</td>
<td>(3.93 \times 10^7)</td>
<td>31.28</td>
<td>float (LL^T)</td>
</tr>
<tr>
<td>10M</td>
<td>(1.04 \times 10^7)</td>
<td>(8.91 \times 10^7)</td>
<td>75.66</td>
<td>complex (LDL^T)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Audi</th>
<th>8</th>
<th>64</th>
<th>128</th>
<th>2×64</th>
<th>4×32</th>
<th>8×16</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facto (s)</td>
<td>103</td>
<td>21.1</td>
<td>17.8</td>
<td>18.6</td>
<td>13.8</td>
<td>13.4</td>
<td>17.2</td>
</tr>
<tr>
<td>Mem (Gb)</td>
<td>11.3</td>
<td>12.7</td>
<td>13.4</td>
<td>2×7.68</td>
<td>4×4.54</td>
<td>8×2.69</td>
<td>14.5</td>
</tr>
<tr>
<td>Solve (s)</td>
<td>1.16</td>
<td>0.31</td>
<td>0.40</td>
<td>0.32</td>
<td>0.21</td>
<td>0.14</td>
<td>0.49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10M</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facto (s)</td>
<td>3020</td>
<td>1750</td>
<td>654</td>
<td>356</td>
<td>260</td>
</tr>
<tr>
<td>Mem (Gb)</td>
<td>122</td>
<td>124</td>
<td>127</td>
<td>133</td>
<td>146</td>
</tr>
<tr>
<td>Solve (s)</td>
<td>24.6</td>
<td>13.5</td>
<td>3.87</td>
<td>2.90</td>
<td>2.89</td>
</tr>
</tbody>
</table>
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% (will increase with NUMA factor)
2
Kernels
Panel factorization (CPU only)

- 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).

\[ X \times b_d = b_{i,i>d} \]

**Figure:** Panel update
Trailing supernodes update

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

Figure: Panel update
Why a new kernel?

- A BLAS call ⇒ a CUDA startup paid;
- Many AXPY calls ⇒ loss of performance.

⇒ need a GPU kernel to compute all the updates from $P_1$ on $P_2$ at once.
How?

auto-tunning GEMM CUDA kernel

▶ Auto-tunning done by the framework ASTRA developed by Jakub Kurzak for MAGMA and inspired from ATLAS;
▶ computes $C \leftarrow \alpha AX + \beta B$, $AX$ split into a 2D tiled grid;
▶ a block of threads computes each tile;
▶ each thread computes several entries of the tile in the shared memory and subtract it from $C$ in the global memory.

Sparse GEMM cuda kernel

▶ Based on auto-tunning GEMM CUDA kernel;
▶ Added two arrays giving first and last line of each blocks of $P_1$ and $P_2$;
▶ Computes an offset used when adding to the global memory.
Sparse GEMM on GPU

Figure: Panel update on GPU
GPU kernel experimentation

Parameters

- $Ncol_A = 100$;
- $Ncol_B = Nrow_{A_11} = 100$;
- $Nrow_A$ varies from 100 to 2000;
- Random number and size of blocks in $A$;
- Random blocks in $B$ matching $A$;
- Get mean time of 10 runs for a fixed $Nrow_A$ with different blocks distribution.

Figure: GPU kernel experimentation
GPU kernel performance

![Graph showing GPU kernel performance with line plots for GPU time, GPU time with transfer, and CPU time against the number of rows.]

**Figure:** Sparse kernel timing with 100 columns.
3
Runtimes
Runtimes

- Task-based programming model;
- Tasks scheduled on computing units (CPUs, GPUs, ...);
- Data transfers management;
- Dynamically 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 $B_{k,k \geq i} \times B_i^T$ subtracted 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
# Matrices and Machines

## Matrices

<table>
<thead>
<tr>
<th>Name</th>
<th>$N$</th>
<th>$\text{NNZ}_A$</th>
<th>Fill ratio</th>
<th>$\text{OPC}$</th>
<th>Fact</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHD</td>
<td>$4.86 \times 10^5$</td>
<td>$1.24 \times 10^7$</td>
<td>61.20</td>
<td>$9.84 \times 10^{12}$</td>
<td>Float $LU$</td>
</tr>
<tr>
<td>Audi</td>
<td>$9.44 \times 10^5$</td>
<td>$3.93 \times 10^7$</td>
<td>31.28</td>
<td>$5.23 \times 10^{12}$</td>
<td>Float $LL^T$</td>
</tr>
<tr>
<td>10M</td>
<td>$1.04 \times 10^7$</td>
<td>$8.91 \times 10^7$</td>
<td>75.66</td>
<td>$1.72 \times 10^{14}$</td>
<td>Complex $LDL^T$</td>
</tr>
</tbody>
</table>

## Machines

<table>
<thead>
<tr>
<th>Processors</th>
<th>Frequency</th>
<th>GPUs</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Opteron 6180 SE (4 × 12)</td>
<td>2.50 GHz</td>
<td>Tesla T20 ($\times 2$)</td>
<td>256 GiB</td>
</tr>
</tbody>
</table>
CPU only results on Audi

Figure: $LL^T$ decomposition on Audi (double precision)
CPU only results on MHD

![Graph showing factorization time for different numbers of threads using PaStiX, PaStiX with StarPU, and PaStiX with DAGUE.]

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 on Romulus (double precision)
MHD: GPU results on Romulus (StarPU)

Figure: MHD $LU$ decomposition with GPU on Romulus (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).

Future 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;
- Bit-compatibility for a same number of processors?
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

![Graph showing factorization time vs fill-in for AUDI matrix with different values of k.](image-url)
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
BACCHUS 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 Bacchus team
CEMRACS’2012 - Luminy