Parallel Sparse Direct Methods and the MUMPS package

Jean-Yves L’Excellent, INRIA and LIP-ENS Lyon, France
Joint work with the MUMPS team (Lyon, Toulouse, Bordeaux).
FCA 2010
Outline

Introduction to Sparse Direct Methods

Parallel Approaches

A MUltifrontal Massively Parallel Solver (MUMPS)

Recent Research and Developments

Concluding Remarks and on-going research
Outline

Introduction to Sparse Direct Methods

Parallel Approaches

A MUltifrontal Massively Parallel Solver (MUMPS)

Recent Research and Developments

Concluding Remarks and on-going research
Sparse Direct Methods

Discretization of a physical problem (eg, finite elements) $\Rightarrow$ Solution of sparse systems $Ax = b$

Often the most expensive part of a simulation process

Sparse direct methods:

- Solve $Ax = b$ by decomposing $A$ under the form $LU, LDL^t$ or $LL^t$
  then solve triangular systems ($Ly = b$, then $Ux = y$)

Black box?

- Default (automatic/adaptive) setting of options is often available
- A better knowledge and setting of the preprocessing and algorithmic options can help the user improving:
  - size of factors and memory needed
  - operation count and computational time
  - reliability of the flops/memory estimates
  - numerical accuracy
Preprocessing - illustration

Original \((A = \text{LHR01})\)

Preprocessed matrix \((A'(\text{LHR01}))\)

Modified Problem: \(A'x' = b'\) with \(A' = P_nPD_rAQD_cP_tQ_n\)

Jean-Yves L’Excellent, FCA 2010
Three-phase scheme to solve $Ax = b$ 

1. Analysis
   - Preprocessing of $A$ (permutations, scalings)
   - Build dependency graph (tree)
   - Prepare factorization (mapping, memory estimates)

2. Factorization : $A = LU$ (or $LDL^t$, or $LL^t$)
   *dynamic pivoting for numerical stability*

3. Solve :
   - the solution $x$ is computed by means of forward and backward substitutions
   - improvement of solution (iterative refinement), error analysis
Step $k$ of $LU$ factorization ($a_{kk}$ pivot):

- For $i > k$ compute column $k$ of $L$ factors: $l_{ik} = a_{ik} / a_{kk}$ ($= a'_{ik}$).
- For $i > k, j > k$ update remaining rows/cols in matrix:

\[
    a'_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}} = a_{ij} - l_{ik} \times a_{kj}
\]

- If $a_{ik} \neq 0$ and $a_{kj} \neq 0$ then $a'_{ij} \neq 0$
- If $a_{ij}$ was zero $\rightarrow$ its non-zero value must be stored

\[
\begin{array}{ccc}
    & k & j \\
    k & x & x \\
    i & x & x \\
\end{array}
\quad
\begin{array}{ccc}
    & k & j \\
    k & x & x \\
    i & x & x \\
\end{array}
\]

\textit{fill-in}
Gaussian elimination and sparsity

• Interest of permuting a matrix:

$$\begin{pmatrix}
X & X & X & X & X & X \\
X & X & 0 & 0 & 0 & 0 \\
X & 0 & X & 0 & 0 & 0 \\
X & 0 & 0 & X & 0 & 0 \\
X & 0 & 0 & 0 & X & 0 \\
X & 0 & 0 & 0 & 0 & X \\
\end{pmatrix} \leftrightarrow 5 \begin{pmatrix}
X & 0 & 0 & 0 & 0 & X \\
0 & X & 0 & 0 & 0 & X \\
0 & 0 & X & 0 & 0 & X \\
0 & 0 & 0 & X & 0 & X \\
X & X & X & X & X & X \\
\end{pmatrix}$$

• Ordering the variables has a strong impact on
  ○ fill-in
  ○ number of operations
  ○ shape of the dependency graph (tree) and parallelism
• Fill reduction is NP-complete [Yannakakis 81]
Impact of ordering the variables

Original matrix

Factorized matrix

58202 nonzeros in factors with natural ordering
Impact of ordering the variables

Permuted matrix
(Reverse Cuthill McKee)

Factorized permuted matrix

16924 nonzeros in $LU$ factors after fill-reducing ordering
Main classes of methods for minimizing fill-in during factorization

1. **Global approaches**: matrix is permuted into a matrix with a given pattern (e.g. block-bordered matrix with nested dissections)

   Graph partitioning

   ![Graph partitioning diagram](image)

   Permutated matrix

   ![Permutated matrix diagram](image)

2. **Local heuristics**: at each step of the factorization, select a pivot likely to minimize fill-in (e.g. Markovitz criterion, minimum degree)

3. **Hybrid**: once matrix is permuted to obtain a block structure, apply local heuristics within the blocks.
Impact of fill-reducing heuristics (MUMPS)

<table>
<thead>
<tr>
<th></th>
<th>METIS</th>
<th>SCOTCH</th>
<th>PORD</th>
<th>AMF</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUPTA2</td>
<td>2757.8</td>
<td>4510.7</td>
<td>4993.3</td>
<td>2790.3</td>
<td>2663.9</td>
</tr>
<tr>
<td>SHIP_003</td>
<td>83828.2</td>
<td>92614.0</td>
<td>112519.6</td>
<td>96445.2</td>
<td>155725.5</td>
</tr>
<tr>
<td>TWOTONE</td>
<td>29120.3</td>
<td>27764.7</td>
<td>37167.4</td>
<td>29847.5</td>
<td>29552.9</td>
</tr>
<tr>
<td>WANG3</td>
<td>4313.1</td>
<td>5801.7</td>
<td>5009.9</td>
<td>6318.0</td>
<td>10492.2</td>
</tr>
<tr>
<td>XENON2</td>
<td>99273.1</td>
<td>112213.4</td>
<td>126349.7</td>
<td>237451.3</td>
<td>298363.5</td>
</tr>
</tbody>
</table>

- **METIS** (Karypis and Kumar) and **SCOTCH** (Pellegrini) are global strategies (recursive nested dissection).
- **PORD** (Schulze, Paderborn Univ.) recursive dissection based on a bottom-up strategy to build the separator.
- **AMD** (Amestoy, Davis and Duff) is a local strategy based on Approximate Minimum Degree.
- **AMF** (Amestoy) is a local strategy based on Approx. Minimum Fill.
## Impact of fill-reducing heuristics (MUMPS)

### Size of factors (millions of entries)

<table>
<thead>
<tr>
<th></th>
<th>METIS</th>
<th>SCOTCH</th>
<th>PORD</th>
<th>AMF</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GUPTA2</strong></td>
<td>8.55</td>
<td>12.97</td>
<td>9.77</td>
<td>7.96</td>
<td>8.08</td>
</tr>
<tr>
<td><strong>SHIP_003</strong></td>
<td>73.34</td>
<td>79.80</td>
<td>73.57</td>
<td>68.52</td>
<td>91.42</td>
</tr>
<tr>
<td><strong>TWOTONE</strong></td>
<td>25.04</td>
<td>25.64</td>
<td>28.38</td>
<td>22.65</td>
<td>22.12</td>
</tr>
<tr>
<td><strong>WANG3</strong></td>
<td>7.65</td>
<td>9.74</td>
<td>7.99</td>
<td>8.90</td>
<td>11.48</td>
</tr>
<tr>
<td><strong>XENON2</strong></td>
<td>94.93</td>
<td>100.87</td>
<td>107.20</td>
<td>144.32</td>
<td>159.74</td>
</tr>
</tbody>
</table>

### Time for factorization (seconds, IBM Power 4)

<table>
<thead>
<tr>
<th></th>
<th>1p</th>
<th>16p</th>
<th>32p</th>
<th>64p</th>
<th>128p</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUDI</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3D, $10^6 \times 10^6$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>METIS</strong></td>
<td>2640</td>
<td>198</td>
<td>108</td>
<td>70</td>
<td>42</td>
</tr>
<tr>
<td><strong>PORD</strong></td>
<td>1599</td>
<td>186</td>
<td>146</td>
<td>83</td>
<td>54</td>
</tr>
</tbody>
</table>

Jean-Yves L’Excellent, FCA 2010
The multifrontal method (Duff, Reid’83)

Memory is divided into two parts (that can overlap in time):

- the factors
- the active memory

Factors Stack of contribution blocks

Active frontal matrix

Stack of contribution blocks

Active Memory

Elimination tree represents tasks dependencies
### Impact of fill-reducing heuristics

#### Peak of active memory for multifrontal approach

<table>
<thead>
<tr>
<th></th>
<th>METIS</th>
<th>SCOTCH</th>
<th>PORD</th>
<th>AMF</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUPTA2</td>
<td>58.33</td>
<td>289.67</td>
<td>78.13</td>
<td>33.61</td>
<td>52.09</td>
</tr>
<tr>
<td>SHIP_003</td>
<td>25.09</td>
<td>23.06</td>
<td>20.86</td>
<td>20.77</td>
<td>32.02</td>
</tr>
<tr>
<td>TWOTONE</td>
<td>13.24</td>
<td>13.54</td>
<td>11.80</td>
<td>11.63</td>
<td>17.59</td>
</tr>
<tr>
<td>WANG3</td>
<td>3.28</td>
<td>3.84</td>
<td>2.75</td>
<td>3.62</td>
<td>6.14</td>
</tr>
<tr>
<td>XENON2</td>
<td>14.89</td>
<td>15.21</td>
<td>13.14</td>
<td>23.82</td>
<td>37.82</td>
</tr>
</tbody>
</table>
Theoretical limits of sparse direct methods

Regular 2D and 3D problems (e.g. finite difference)

<table>
<thead>
<tr>
<th></th>
<th>2D $n \times n$ grid</th>
<th>3D $n \times n \times n$ grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonzeros in original matrix</td>
<td>$O(n^2)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Nonzeros in factors</td>
<td>$O(n^2 \log n)$</td>
<td>$O(n^4)$</td>
</tr>
<tr>
<td>Floating-point ops</td>
<td>$O(n^3)$</td>
<td>$O(n^6)$</td>
</tr>
</tbody>
</table>

2D problems: direct solvers often preferred
3D problems: current (increasing) limit $\approx 100$ million equations? not exploiting all cores of petascale computers yet!

Algorithmic issues: parallelism, mapping irregular datastructures, scheduling for memory/for performance, memory scalability, out-of-core storage.
### Direct method vs. Iterative method

#### Direct
- Factorization of $A$
  - May be costly (memory/flops)
  - Factors can be reused for multiple/successive right-hand sides
  - Efficiency from dense blocks
- Very general/robust
  - Numerical accuracy
  - Irregular/unstructured problems

#### Iterative
- Rely on efficient Mat-Vect product
  - Memory effective
  - Successive right-hand sides is problematic
  - Smaller blocks
- Efficiency depends on:
  - Convergence preconditionning
  - Numerical prop./struct. of $A$

### Hybrid approaches
(Domain Decomposition, Schur, Block Cimmino ...)

*Often strongly rely on both iterative and direct technologies*
Outline

Introduction to Sparse Direct Methods

Parallel Approaches

A MULTifrontal Massively Parallel Solver (MUMPS)

Recent Research and Developments

Concluding Remarks and ongoing research
Several levels of parallelism can be exploited (message passing or threads):

- **At problem level**: problem can be decomposed into subproblems (e.g. domain decomposition)
- **At matrix level**: sparsity implies more independency between calculations (two branches of the dependency tree can be processed independently)
- **At submatrix level**: within dense linear algebra computations (parallel BLAS, ScaLAPACK, ... )
• The parallel algorithm is characterized by:
  ◦ its computational graph dependency
  ◦ its communication graph

• Simplificative assumptions: each column of $L$
  ◦ is assigned to a single processor
  ◦ is associated to a node in the tree representing dependencies between column eliminations
Computational strategies for parallel solvers

Left and right-looking approaches – sequential or multithread codes

Three classical approaches – message passing

“Fan-in” similar to left-looking, demand-driven
“Fan-out” similar to right-looking, data-driven
“Multifrontal” local communications, data-driven
Fan-in, Fan-out and Multifrontal

Figure: Communication schemes for the three approaches.
Fan-in, Fan-out and Multifrontal

Figure: Communication schemes for the three approaches.
Fan-in, Fan-out and Multifrontal

(a) Fan-in, demand-driven

(b) Fan-out, data-driven

(c) Multifrontal, data-driven

**Figure:** Communication schemes for the three approaches.
Fan-in, Fan-out and Multifrontal

(a) Fan-in, demand-driven
(b) Fan-out, data-driven
(c) Multifrontal, data-driven

**Figure:** Communication schemes for the three approaches.
**Figure:** Communication schemes for the three approaches.

(a) Fan-in, demand-driven

(b) Fan-out, data-driven

(c) Multifrontal, data-driven
<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
<th>Availability (<a href="http://www">www</a>.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCSLIB</td>
<td>Multifrontal</td>
<td>SYM/UNS</td>
<td>Boeing → Access Analytics</td>
</tr>
<tr>
<td>HSL MA87</td>
<td>Supernodal</td>
<td>SPD</td>
<td>cse.clrc.ac.uk/Activity/HSL</td>
</tr>
<tr>
<td>MA41</td>
<td>Multifrontal</td>
<td>UNS</td>
<td>cse.clrc.ac.uk/Activity/HSL</td>
</tr>
<tr>
<td>MA49</td>
<td>Multifr. QR</td>
<td>RECT</td>
<td>cse.clrc.ac.uk/Activity/HSL</td>
</tr>
<tr>
<td>PanelLLT</td>
<td>Left-looking</td>
<td>SPD</td>
<td>Ng</td>
</tr>
<tr>
<td>PARDISO</td>
<td>Left-right</td>
<td>SYM/UNS</td>
<td>Schenk</td>
</tr>
<tr>
<td>PSL†</td>
<td>Left-looking</td>
<td>SPD/UNS</td>
<td>SGI product</td>
</tr>
<tr>
<td>SuperLU_MT</td>
<td>Left-looking</td>
<td>UNS</td>
<td>nersc.gov/~xiaoye/SuperLU</td>
</tr>
<tr>
<td>SuiteSparseQR</td>
<td>Multifr. QR</td>
<td>RECT</td>
<td>cise.ufl.edu/research/sparse/SPQR</td>
</tr>
<tr>
<td>TAUCS</td>
<td>Left/Multifr.</td>
<td>SYM/UNS</td>
<td>tau.ac.il/~stoledo/taucs</td>
</tr>
<tr>
<td>WSMP†</td>
<td>Multifrontal</td>
<td>SYM/UNS</td>
<td>IBM product</td>
</tr>
</tbody>
</table>

† Only object code available.
Some distributed-memory sparse direct codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
<th>Availability (<a href="http://www">www</a>.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSCPACK</td>
<td>Multifr./Fan-in</td>
<td>SPD</td>
<td>cse.psu.edu/~raghavan/Dscpack</td>
</tr>
<tr>
<td>MUMPS</td>
<td>Multifrontal</td>
<td>SYM/UNS</td>
<td>graal.ens-lyon.fr/MUMPS and mumps.enseeht.fr</td>
</tr>
<tr>
<td>PaStiX</td>
<td>Fan-in</td>
<td>SYM/UNS</td>
<td>labri.fr/perso/ramet/pastix</td>
</tr>
<tr>
<td>PSPASES</td>
<td>Multifrontal</td>
<td>SPD</td>
<td>cs.umn.edu/~mjoshi/pspases</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Fan-in</td>
<td>SYM/UNS</td>
<td>netlib.org/linalg/spooles</td>
</tr>
<tr>
<td>SuperLU</td>
<td>Fan-out</td>
<td>UNS</td>
<td>nersc.gov/~xiaoye/SuperLU</td>
</tr>
<tr>
<td>S+</td>
<td>Fan-out</td>
<td>UNS</td>
<td>cs.ucsb.edu/research/S+</td>
</tr>
<tr>
<td>WSMP †</td>
<td>Multifrontal</td>
<td>SYM</td>
<td>IBM product</td>
</tr>
</tbody>
</table>

† Only object code available.
Outline

Introduction to Sparse Direct Methods

Parallel Approaches

A Multifrontal Massively Parallel Solver (MUMPS)

Recent Research and Developments

Concluding Remarks and on-going research
MUMPS vs other sparse direct solvers

Address wide classes of problems

- Good numerical stability (dynamic pivoting, preprocessing, postprocessing, error analysis)
  - dynamic datastructures
- Wide range of numerical features

Management of parallelism

- Mainly MPI-based
- Dynamic and asynchronous approach

MUMPS vs SuperLU_DIST
Dynamic Scheduling

- Tasks graph = tree (results from matrix structure and ordering heuristic)
- Each task = partial factorization of a dense matrix
- Most parallel tasks mapped at runtime (typically 80 %)
Dynamic Scheduling

- Tasks graph = tree (results from matrix structure and ordering heuristic)
- Each task = partial factorization of a dense matrix
- Most parallel tasks mapped at runtime (typically 80 %)
Dynamic Scheduling

- Tasks graph = tree (results from matrix structure and ordering heuristic)
- Each task = partial factorization of a dense matrix
- Most parallel tasks mapped at runtime (typically 80 %)

TIME

SUBTREES

2D static decomposition

1D pipelined factorization
P3 and P0 chosen by P2 at runtime

: STATIC

: DYNAMIC
MUMPS Team

Permanent members in 2010

Patrick Amestoy (N7-IRIT, Toulouse)

Jean-Yves L’Excellent (INRIA-LIP, Lyon)

Abdou Guermouche (LABRI, Bordeaux)

Bora Uçar (CNRS-LIP, Lyon)

Alfredo Buttari (CNRS-IRIT, Toulouse)
• **Recent post-docs:**
  Indranil Chowdhury (May 2009–March 2010)
  Bora Uçar (Jan. 2007-Dec. 2008)

• **Recent PhD Students:**
  Emmanuel Agullo (ENS Lyon, 2005-2008)
  Mila Slavova (CERFACS, Toulouse, 2005-2009)
  François-Henry Rouet (INPT-IRIT, Toulouse, 2009-)
  Clément Weisbecker (INPT-IRIT and EDF, Toulouse, 2009-)

• **Engineers**
  Aurélie Fève (INRIA, 2005-2007)
MUMPS (MUltifrontal Massively Parallel Solver)

Initially funded by European project PARASOL (1996-1999)

Platform for research

• Research projects
• PhD thesis
• Hybrid methods

Competitive software package used worldwide

• Co-developed by Lyon-Toulouse-Bordeaux
• Latest release: MUMPS 4.9.2, Nov. 2009, ≈ 250 000 lines of C and Fortran code
• Integrated within commercial and academic packages (Samcef from Samtech, Actran from Free Field Technologies, Code_Aster or Telemac from EDF, IPOPT, Petsc, Trilinos, ...).
• 1000+ downloads per year from our website, half from industries: Boeing, EADS, EDF, Petroleum industries, etc.
Download Requests from the MUMPS website
User’s distribution map

1000+ download requests per year
Main functionalities/features (1)

Fortran, C, Matlab and Scilab interfaces

Input formats
- Assembled format
- Distributed assembled format
- Sum of elemental matrices
- Sparse, multiple right-hand sides, distributed solution

Type of matrices
- Symmetric (positive definite, indefinite), Unsymmetric
- Single/Double precision with Real/Complex arithmetic
Main functionalities/features (II)

Preprocessing and Postprocessing

- Reduce fill-in: symmetric orderings interfaced: AMD, QAMD, AMF, PORD, (par)METIS, (pt)SCOTCH
- Numerical preprocessing: unsymmetric orderings and scalings
- Iterative refinement and backward error analysis

Numerical pivoting

- Partial pivoting and two-by-two pivots (symmetric)
- Static pivoting, "null" pivot detection, null-space basis
Main functionalities/features (III)

Solving larger problems

- Hybrid scheduling
- Out-of-core
- Parallel analysis

Miscellaneous

- Partial factorization and Schur complement (for hybrid solvers)
- Inertia, determinant, entries of $A^{-1}$ ...
Outline

Introduction to Sparse Direct Methods

Parallel Approaches

A MULTifrontal Massively Parallel Solver (MUMPS)

Recent Research and Developments

Concluding Remarks and on-going research
Recent Research and Developments

Out-of-core: use disks when memory not big enough

2 PhD completed

- Emmanuel AGULLO (ENS Lyon, 2005-2008) *On the Out-of-core Factorization of Large Sparse Matrices*
- Mila Slavova (CERFACS, Toulouse, 2005-2009) *Parallel triangular solution in the out-of-core multifrontal approach*

- Task scheduling, prefetching, low-level I/O’s, synchronous vs asynchronous approaches . . .
- Lots of developments by MUMPS team
- Core memory requirements for “Epicure” matrix (3D problem, 1 million equations, EDF, Code_Aster)
  - Total memory (in-core) = 20.8 GBytes
  - Active memory (out-of-core) = 3.7 GBytes
Goal: compute a set of entries $(A^{-1})_{ij} = e_i^T A^{-1} e_j$

PhD F.-H. Rouet, 2009- (extension of PhD work of M. Slavova)

- $x = L^{-1} e_j$: Exploit sparsity of right-hand sides (RHS)
- $(A^{-1})_{ij} = e_i^T U^{-1} x$: Exploit sparsity of requested entries of solution

- Combinatorial problem: how to partition the right-hand-sides?
- Illustration on an application from astrophysics (CESR, Toulouse)

| No exploit sparsity of RHS | 43 396 sec |
| Natural ordering           | 721 sec    |
| Postordering               | 199 sec    |

time to compute **ALL** diagonal of $A^{-1}$, OOC factors, N=148k

- On-going work: hypergraph partitionning, in-core, parallel aspects
Out-of-core related work: entries of $A^{-1}$

Goal: compute a set of entries $(A^{-1})_{ij} = e_i^T A^{-1} e_j$

PhD F.-H. Rouet, 2009- (extension of PhD work of M. Slavova)

- $x = L^{-1} e_j$ : Exploit sparsity of right-hand sides (RHS)
- $(A^{-1})_{ij} = e_i^T U^{-1} x$ Exploit sparsity of requested entries of solution
- Combinatorial problem: how to partition the right-hand-sides?
- Illustration on an application from astrophysics (CESR, Toulouse)

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No exploit sparsity of RHS</td>
<td>43 396</td>
</tr>
<tr>
<td>Natural ordering</td>
<td>721</td>
</tr>
<tr>
<td>Postordering</td>
<td>199</td>
</tr>
</tbody>
</table>

time to compute **ALL** diagonal of $A^{-1}$, OOC factors, $N=148k$

- **On-going work**: hypergraph partitionning, in-core, parallel aspects
Recent Research and Developments (cont’)

- Parallel analysis (Buttari et al.)
  - Critical when graph of matrix cannot be centralized on one processor
  - Use parallel graph partitioning tools PT-Scotch (LaBRI, Bordeaux) or ParMETIS (Univ. Minnesota)

- Parallel scalings (Ruiz, Uçar et al.):
  \[ A^{(k+1)} = D_r^{(k)} A^{(k)} D_c^{(k)} \]
  where \( D_r^{(k)} = \text{diag}(\sqrt{||A_i^{(k)}||_p})^{-1} \), \( D_c^{(k)} = \text{diag}(\sqrt{||A_{i*}^{(k)}||_p})^{-1} \)

<table>
<thead>
<tr>
<th></th>
<th>Flops ((×10^6))</th>
<th># of entries in factors ((×10^6))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>estimated</td>
<td>effective</td>
</tr>
<tr>
<td>scaling</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>a0nsdsil</td>
<td>7.7</td>
<td>2.5</td>
</tr>
<tr>
<td>C-54</td>
<td>281</td>
<td>209</td>
</tr>
</tbody>
</table>
Multicores : mixing MPI and OpenMP

(work started with 1-year postdoc Indranil Chowdhury, funded by ANR)
Main goal : understand limits of MPI + threaded BLAS + OpenMP

- Use TAU and Intel Vtune profilers
- Insert OpenMP directives in critical kernels of MUMPS
- Experiment with various matrices on various architectures

Difficulties :

- Understand speed-down problems (use OMP IF directives)
- Side effects of threaded BLAS libraries (MKL ok)
- Dependence on OpenMP implementations
- Threaded BLAS within OpenMP parallel regions
Hybrid MPI + multithreaded version of MUMPS

Initial results:

- good compromise: use 4 threads per MPI process
  typical speed-up: 6 on 8 cores
- unsymmetric and Cholesky versions more efficient than $LDL^t$

Recent illustrative results:

- 96-core machine, INRIA Bordeaux Sud-Ouest
- Runs on a medium-size problem (ULTRASOUND80)
  - Order: 531,441, non-zeroes: $330 \times 10^6$
  - Factors: $981 \times 10^6$ entries, $3.9 \times 10^{12}$ flops
Experimental results

![Bar chart showing execution time in seconds for MPI only, MPI+Thread MKL (4 threads per process), and MPI+Thread MKL + OMP (4 threads per process) for different numbers of processes (1, 4, 8, 16, 32, 64).]
Remark on memory usage

More threads per MPI process is good for memory
Recent Developments: software issues

- **MUMPS**: 15-year old research code
- **Software engineering/maintenance/support** not so easy in the context of academic research
- **Combination of options difficult to maintain** (asynchronous pipelined factorizations + out-of-core + many pivoting strategies + various types of matrices and input formats + dynamic distributed datastructures . . .)
- **Recent initiatives**:
  - 1-year engineer funded by CNRS (2008): Philippe Combes
    - taught us better practices, cvs to svn migration, trunk and release branches, improved makefiles, scripts for non regression tests, . . .
  - Action of Technological Development funded by INRIA (2009-2012)
    - 2-year funding for a young engineer (G. Joslin)
    - part-time of a permanent INRIA engineer (M. Brémond)
    - Objective: ensure durability and evolutivity of MUMPS
Outline

Introduction to Sparse Direct Methods

Parallel Approaches

A MUltifrontal Massively Parallel Solver (MUMPS)

Recent Research and Developments

Concluding Remarks and on-going research
Direct methods often preferred to iterative methods in industrial applications (robustness)... as long as computer is big enough!

- **Main originalities of MUMPS:**
  - wide range of functionalities
  - numerical stability in distributed-memory environments

- **Several other direct solvers available,** fitting various needs
  
  [GridTLSE expertise website, http://gridtlse.org: users can experiment and combine different algorithms and codes to get advice and statistics on best algorithmic choices for a given purpose/type of matrix]

- **Evolution of platforms and applications** ⇒ considerable efforts on algorithms and code development
Some On-going Research

- **More parallelism**
  - Many efforts towards **multicores** (Pastix, WSMP, HSL MA87)
  - Threaded BLAS vs single-threaded BLAS and portability issues
  - MPI parallelism on thousands of nodes
    (performance and **memory scalability**)

- **GPGPU’s**
  - High potential from multifrontal methods (large blocks)
  - Experiments with MUMPS by A. Guermouche (LaBRI, Bordeaux)
  - See also Daniel Pierce et al. (multifrontal solver in BCSLIB-GPU) and Bob Lucas et al. (also on multifrontal approaches)

- **Evolution of sparse direct solvers**
  - Tool for **hybrid direct-iterative methods**: domain decomposition, Schur-complement methods (Giraud et al.), block-Cimmino approaches (Ruiz et al.),
  - Exploit **low rank** of off-diagonal factor blocks in some PDE’s (Xia et al., Berkeley; PhD Clément Weisbecker, Toulouse)