Exploiting Multithreaded Tree Parallelism for Multicore Systems in a Parallel Multifrontal Solver

Mohamed Sid-Lakhdar
ENS Lyon, France

Joint work with: Patrick Amestoy, Alfredo Buttari, Abdou Guermouche, Jean-Yves L'Excellent

SIAM PP12

Feb 16, 2012
Problem

**Problem**

\[ A \times x = b \]

**Solution**

**Factorization**

\[ A = L \times U \]

**Resolution**

\[ L \times y = b \quad U \times x = y \]

LU factorization

Forward substitution

Backward substitution

**Sparse matrix**

**Elimination tree**

Multifrontal factorization

[Duff Reid 84]
Node parallelism and tree parallelism

Our context:
- **MUMPS**: MUltifrontal Massively Parallel Solver, MPI-based
  - Node + tree parallelism at MPI level
  - Node parallelism at thread level (threaded BLAS + OpenMP)

Question/objective:
- Is it possible to do better on multicore systems by introducing tree parallelism at thread level?
Geist-Ng algorithm
Geist-Ng algorithm
Geist-Ng algorithm

Subtree cost:
- flops
- seconds
- ...

L0
Geist-Ng algorithm

Tentative mapping
Geist-Ng algorithm

L0 acceptance criterion
Geist-Ng algorithm
Possible mapping
Geist-Ng algorithm

\[ L_0 \leftarrow \{ \text{roots of the trees} \} \]

Estimate costs of all subtrees

Repeat

Find node N with most expensive subtree

\[ L_0 \leftarrow L_0 \setminus \{N\} \cup \{ \text{children of N} \} \]

Tentative mapping of L0 subtrees

Until acceptance criterion is reached
Hybrid environment
MPI process mapping (MUMPS)
Introducing threaded-tree parallelism

```
+---+   +---+
|   |   |   |
+---+   +---+
      X   X
T1 T2 T1 T2
P1 P2 P1 P2
P3 P3
```

- MPI
- OMP

L0_MPI
L0_OMP

- Node
- Tree

P1 P2 P3
Our approach

1. Study serial & threaded dense-matrix factorization kernels to predict performance on dense matrices

2. Simulate the behavior of L0 algorithms to predict computation times on sparse matrices

3. Implement a shared L0 algorithm in MUMPS, using:
   - Under L0: tree parallelism only
   - Above L0: node parallelism only (inside MPI subtree)

4. Experiment and understand the limits of the approach
1. Performance prediction of dense-matrix factorizations

- Benchmark:
  - 1 or 8 cores
  - #pivots and Schur size in log-scale grid
    
    $[[1..10..100..1000..10000]]$
  - diagonally dominant matrices (no numerical pivoting)

- Model for dense matrices:
  - Bilinear interpolation on Gflops rate
  - Time = Gflops / Gflops rate
2. Performance prediction on sparse factorization

- **Simulator** implemented in **Python** (using discrete event simulator **SimPy**)

- Model for sparse matrices based on model for dense matrices at each node of the tree:
  - Time under L0: dense model for 1 thread (tree parallelism)
  - Time above L0: dense model for N threads (node parallelism)

- **Interest of the simulator:**
  - Predict factorization time with and without L0
  - Quickly experiment various strategies to define L0 *(cost function for subtrees, mapping, acceptance criterion)*
Validity of the simulator

Simulator vs reality

Sequential

Time (sec)

Matrix

- Wang3
- Li
- G3_circuit
- Human_gene1
- Haltere
- Conf5_4-8x8-05
- Audi

Simulation

Reality
Validity of the simulator

Simulator vs reality
Parallel (on Haltere only)

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Simulator</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>
Simulator results

3D Matrix
100x100x100 grid
(finite differences)

2D Matrix
1000x1000 grid
(finite differences)
3. Implementation within MUMPS

- MPI code → hybrid MPI/OpenMP code
- Isolate thread-private data
- Analysis phase: implement model and two algorithms to define L0
- Factorization phase:
  1. Tree parallelism under L0
  2. Synchronize all threads
  3. Node parallelism (OpenMP, threaded BLAS) above L0
Algorithm

\[ L_0 \leftarrow \{ \text{roots of the trees} \} \]

Estimate costs of all subtrees

Repeat

Find node N with most expensive subtree

\[ L_0 \leftarrow L_0 \setminus \{ N \} \cup \{ \text{children of N} \} \]

Tentative mapping of L0 subtrees

Until acceptance criterion is reached
Flop-based algorithm

- **Subtree cost** based on #flops
- **Mapping** the subtrees under L0 using a DFF (Decreasing First Fit) scheduling algorithm
- **Acceptance criterion:**
  \[ \frac{T_{min}}{T_{max}} \geq \text{Threshold} \]
- Threshold experimentally tuned: **0.9**
Benchmark-based algorithm

- **Subtree cost** based on model
- **Mapping** as before → $T_{under}=T_{max}$
- Simulate $T_{above}$ L0 using the model for NbThreads/node
- **Acceptance criterion**: Minimize $T_{total} = T_{under} + T_{above}$
  $T_{total}$ smaller than all the ones before and after (10-100 next)
4. Experimental results (Nehalem)

- Test problems: Univ. Florida & Gridlise collections

<table>
<thead>
<tr>
<th>Matrix</th>
<th>N</th>
<th>NZ</th>
<th>Application field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mono_500Hz</td>
<td>169410</td>
<td>5033796</td>
<td>Acoustics</td>
</tr>
<tr>
<td>Haltere</td>
<td>1288825</td>
<td>19664725</td>
<td>3D Electromagnetics</td>
</tr>
<tr>
<td>3Dspectralwave2</td>
<td>292008</td>
<td>12935272</td>
<td>Materials</td>
</tr>
<tr>
<td>3D 100x100x50</td>
<td>500000</td>
<td>5500000</td>
<td>Finite differences</td>
</tr>
<tr>
<td>conf5_4-8x8-05</td>
<td>49152</td>
<td>1916928</td>
<td>Quantum chemistry</td>
</tr>
<tr>
<td>AUDI</td>
<td>943695</td>
<td>77651847</td>
<td>Structural mechanics</td>
</tr>
<tr>
<td>G3_circuit</td>
<td>150102</td>
<td>726674</td>
<td>Circuit simulation</td>
</tr>
<tr>
<td>QIMONDA07</td>
<td>8613291</td>
<td>66900289</td>
<td>Circuit simulation</td>
</tr>
</tbody>
</table>

- Next slide: compare speed-ups
  - With threaded BLAS only
  - With threaded BLAS + OpenMP directives
  - With flop-based L0 algorithm
  - With benchmark-based L0 algorithm
Speed-Ups on 8-core Intel Nehalem

1 MPI 8 threads
Speed-Ups on 8-core Intel Nehalem

L0_OMP VS MPI
4. Experimental results (Istambul)

- Test problems: Univ. Florida & Gridlise collections

<table>
<thead>
<tr>
<th>Matrix</th>
<th>N</th>
<th>NZ</th>
<th>Application field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haltere</td>
<td>1288825</td>
<td>20953550</td>
<td>3D Electromagnetics</td>
</tr>
<tr>
<td>Flux2M</td>
<td>2001728</td>
<td>212885526</td>
<td>Fluid dynamics</td>
</tr>
<tr>
<td>fem_hireq_circuit</td>
<td>491100</td>
<td>20239237</td>
<td>Electromagnetics</td>
</tr>
</tbody>
</table>

- Next slide: compare speed-ups
  - With threaded BLAS only
  - With threaded BLAS + OpenMP directives
  - With flop-based L0 algorithm
  - With benchmark-based L0 algorithm
Speed-Ups on 24-core AMD Istambul
Conclusion

- Threaded tree parallelism implemented
- Synchronization on L0 seems acceptable (MPI can help on more threads)
- Approach can efficiently exploit threaded dense linear algebra libraries
- Promising results
Perspectives

- Work on threaded dense-matrix factorization kernels
- Improve multi-threaded memory management
- MPI + OpenMP scaling on large problems (more MPI processes, more threads per MPI process)
- Design and implement an efficient multi-threaded solve