Performance Prediction and Optimization for Sparse Gaussian Elimination: A Case Study

Research Goals:

- Development of high performance sparse Gaussian elimination.
- Fast simulation of MPI programs on SMPs.

Talk outline:

- Schedule techniques for performance prediction and optimization.
- An application emulator based on Sparse Gaussian Elimination.
- Experience with high performance sparse Gaussian Elimination.
- Other ongoing research activities.
Sparse Gaussian Elimination

- unpredictable dependency and data structures,
- limited parallelism,
- representation and pivoting.
- Poor cache performance due to compressed data.

Challenges for high performance:

- Important for many applications.

A good case for us to study on performance prediction and optimization.

Research work on sparse GE

- S+ (Shen, Yang, Yang, SPAA '98), 11 GFlops.
- S* (Fu, Yang, SC '96), 1.3 GFlops.
- Distributed memory machines.
- SuperLU (Demmel, Gilbert, Li, 96-97), 2.5 GFlops.
- Shared memory machines.
- UMFPACK (Davis and Duff, 93-94).
- Sequential machines.
Our research strategies.

- **Further optimization:** By eliminating RAPID system software for performance prediction and optimization.
- **Code prototyping:** Use the RAPID/PRROS.
- **Symbolic factorization:** Maximize the use of BLAS-3 and eliminate dynamic and better caching performance.
- **Increase the operation count for simpler data structures:** Sequential complexity $x + 1$ may be better than $x$.

**Data partitioning after symbolic factorization**

- **BLAS-3 (and preconditioning)**: $\Lambda / T$
- **Node order partitioning and amalgamation**: Identify more dense structures to maximize the use of BLAS-3 and eliminate dynamic and better caching performance.
- **Static preprocessing to fix data structure**:
Program partitioning for sparse LU

(01) for $k = 1$ to $N$
(02)    Perform task $Factor(k)$;
(03)    for $j = k + 1$ to $N$ with $A_{k,j} \neq 0$
(04)    Perform task $Update(k,j)$;
(05)    endfor
(06)    endfor

- **Factor(k)**: Factorize and pivoting on column block $k$.
- **Update(k,j)**: Delayed updating/swapping that uses column block $k$ to modify $j$. 

Task description

- update
- swapping

Factor(k): pivoting & local updating

Update(k,j): swapping and updating

Column block $k$

Delayed pivoting information
Given the nonzero pattern of a matrix:

**Task dependencies**

RAPID provides an API and tool support for runtime parallelization of irregular codes with mixed granularities.

Dependence transformation analysis

Dependence completion task graph

Iterative asynchronous task assignments, data object owners schedules and execution.

User specifications: shared data objects, tasks and data access patterns.

RAPID derives a dependence graph and uses DAG scheduling (PR ReOS) for mapping and performance prediction.
RAPID provides an efficient schedule execution protocol using asynchronous fast remote memory access.

Runs on Cray T3E, Meiko CS-2. Tested for sparse Cholesky/LU, sparse triangular solver, Newton's method, n-body simulation (fast multipole method).

Source code: 
http://www.cs.ucsb.edu/research/rapid_sw/rapid.html

Scheduling for sparse GE

- ID data mapping: Column blocks are mapped to processors cyclically.
- Task mapping: Tasks that modify the same column block are in the same processor. Tasks are ordered within each processor.
- Performance prediction: Parallel time and space usage.

Source code: 
http://www.cs.ucsb.edu/research/rapid_sw/rapid.html

12
Effectiveness of performance prediction

Actual vs. predicted speedups:

![Graph showing the difference between predicted and actual speedups.]

\[
\text{Parallel time of our code} = \frac{\text{Operation count from SuperLU}}{\text{Mflops on 300MHz T3E}}
\]
Experience learned from RAPID prototyping:

- Fast prototyping and reasonable performance prediction of parallel sparse LU code.
- Set a new performance record for distributed memory machines [SC’96].
- RAPID can predict performance for solving larger matrices, but cannot execute schedules due to memory constraints.
- Predicted space usage is high, exceeding the memory capacity.
- Current RAPID system overhead is still substantial for runtime execution.

Effectiveness of graph scheduling vs. a simpler method.
**Overall sequential performance**

Cray T3E, 300MHz. Megaflop count does not include extra unnecessary operation introduced by static factorization.

**S+**: Optimized code for sparse LU

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Sequential S+</th>
<th>Sequential S+</th>
<th>Sequential S+</th>
<th>Sequential S+</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td>Time (s)</td>
<td>Time (s)</td>
<td>Time (s)</td>
</tr>
<tr>
<td></td>
<td>Mflops</td>
<td>Mflops</td>
<td>Mflops</td>
<td>Mflops</td>
</tr>
<tr>
<td></td>
<td>S+ LU</td>
<td>S+ LU</td>
<td>SuperLU</td>
<td>SuperLU</td>
</tr>
<tr>
<td>siberman4</td>
<td>0.65</td>
<td>3.88</td>
<td>0.78</td>
<td>32.4</td>
</tr>
<tr>
<td>lms30937</td>
<td>1.48</td>
<td>28.72</td>
<td>1.73</td>
<td>24.4</td>
</tr>
<tr>
<td>lms30937</td>
<td>1.58</td>
<td>28.68</td>
<td>1.84</td>
<td>24.3</td>
</tr>
<tr>
<td>siberman3</td>
<td>1.56</td>
<td>30.52</td>
<td>1.98</td>
<td>26.9</td>
</tr>
<tr>
<td>lms30937</td>
<td>1.56</td>
<td>30.52</td>
<td>1.98</td>
<td>26.9</td>
</tr>
<tr>
<td>siberman4</td>
<td>0.52</td>
<td>33.38</td>
<td>0.56</td>
<td>30.6</td>
</tr>
<tr>
<td>lms30937</td>
<td>1.60</td>
<td>39.11</td>
<td>1.53</td>
<td>26.9</td>
</tr>
<tr>
<td>siberman4</td>
<td>2.67</td>
<td>40.11</td>
<td>2.60</td>
<td>26.9</td>
</tr>
<tr>
<td>lms30937</td>
<td>2.67</td>
<td>40.11</td>
<td>2.60</td>
<td>26.9</td>
</tr>
<tr>
<td>siberman4</td>
<td>1.60</td>
<td>39.11</td>
<td>1.53</td>
<td>26.9</td>
</tr>
<tr>
<td>lms30937</td>
<td>1.60</td>
<td>39.11</td>
<td>1.53</td>
<td>26.9</td>
</tr>
<tr>
<td>siberman4</td>
<td>3.23</td>
<td>39.8</td>
<td>3.23</td>
<td>39.8</td>
</tr>
<tr>
<td>lms30937</td>
<td>3.23</td>
<td>39.8</td>
<td>3.23</td>
<td>39.8</td>
</tr>
<tr>
<td>siberman4</td>
<td>10.26</td>
<td>60.28</td>
<td>17.0</td>
<td>39.4</td>
</tr>
<tr>
<td>lms30937</td>
<td>10.26</td>
<td>60.28</td>
<td>17.0</td>
<td>39.4</td>
</tr>
<tr>
<td>siberman4</td>
<td>4.64</td>
<td>105.0</td>
<td>8.39</td>
<td>79.4</td>
</tr>
<tr>
<td>lms30937</td>
<td>4.64</td>
<td>105.0</td>
<td>8.39</td>
<td>79.4</td>
</tr>
</tbody>
</table>

**Key new ideas [SPAA’98]**

- Supernodal matrix multiplication GEMM to improve the kernel computation speed.
- Compact parallelism representation: Elimination forest to guide
- Supernode partitioning and amalgamation.
- 2D asynchronous computation scheduling with small space overhead.
### Parallel Performance of $S^+$

#### on 300 MHz Cray T3E

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$p = 8$</th>
<th>$p = 16$</th>
<th>$p = 64$</th>
<th>$p = 128$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodwin</td>
<td>11.2</td>
<td>7.3</td>
<td>0.35</td>
<td>0.15</td>
</tr>
<tr>
<td>Reeksly</td>
<td>12.4</td>
<td>8.1</td>
<td>0.52</td>
<td>0.24</td>
</tr>
<tr>
<td>MacGyver</td>
<td>13.9</td>
<td>9.6</td>
<td>0.61</td>
<td>0.30</td>
</tr>
</tbody>
</table>

#### on 450 MHz Cray T3E

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$p = 8$</th>
<th>$p = 32$</th>
<th>$p = 128$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodwin</td>
<td>21.5</td>
<td>6.9</td>
<td>0.30</td>
</tr>
<tr>
<td>Reeksly</td>
<td>11.4</td>
<td>8.0</td>
<td>0.50</td>
</tr>
<tr>
<td>MacGyver</td>
<td>12.5</td>
<td>9.3</td>
<td>0.60</td>
</tr>
</tbody>
</table>

### Parallel Performance of $S^+$

#### on 300 MHz Cray T3E

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$p = 8$</th>
<th>$p = 16$</th>
<th>$p = 64$</th>
<th>$p = 128$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodwin</td>
<td>11.2</td>
<td>7.3</td>
<td>0.35</td>
<td>0.15</td>
</tr>
<tr>
<td>Reeksly</td>
<td>12.4</td>
<td>8.1</td>
<td>0.52</td>
<td>0.24</td>
</tr>
<tr>
<td>MacGyver</td>
<td>13.9</td>
<td>9.6</td>
<td>0.61</td>
<td>0.30</td>
</tr>
</tbody>
</table>

#### on 450 MHz Cray T3E

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$p = 8$</th>
<th>$p = 32$</th>
<th>$p = 128$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodwin</td>
<td>21.5</td>
<td>6.9</td>
<td>0.30</td>
</tr>
<tr>
<td>Reeksly</td>
<td>11.4</td>
<td>8.0</td>
<td>0.50</td>
</tr>
<tr>
<td>MacGyver</td>
<td>12.5</td>
<td>9.3</td>
<td>0.60</td>
</tr>
</tbody>
</table>
Summary on sparse GE:

- Effective time/space performance prediction and code prototyping by RAPID.
- Time and space efficient implementation for parallel sparse LU factorization on distributed memory machines.
- Achieve the highest megaflop rate. 11.04Gflops on 128-node 450MHz Cray T3E.
- Comparison: 2.5 Gflops by SuperLU on 8 processors of Cray C90.

Current work and research plans

- Redesign and implement S+ using MPI as a performance benchmark (application emulator).
- Fast simulation of MPI codes on SMPs
  - Step 1: Provide compiler and runtime support for thread-safe execution of MPI codes (ThrMPI).
  - Step 2: Integrate some of Howsim, Gigasim, Dumsim, Petasim, and PYRROS+ work for performance prediction without direct execution.

Yang/UCSB/18/99
Techniques for task graphs scheduling and performance prediction:

- Integrate RAPID with PYRROS+ and use S* code as a test benchmark.
- Parameterized task graphs.
- Symbolic scheduling and performance prediction for PPoPP'97.
- Continue to work on time/space performance prediction.

Performance prediction:

• Techniques for task graph scheduling and performance prediction.

---

<table>
<thead>
<tr>
<th>System on a 4-processor Power Challenge: Performance improvement of ThirMPI over SCI MPI runtime</th>
</tr>
</thead>
</table>

| Matrix size | GE (1024 x 1024) | (4,8%) |
| 4 MPI nodes | Wm (1000 x 1000) | 16 MPI nodes |
| % | GE (512 x 512) | (32%) |
| % | Wm (500 x 500) | (32%) |

---

Current ThirMPI performance