Optimization of a Sparse Hypermatrix Cholesky Factorization

Josep R. Herrero, Juan J. Navarro
{josepr,juanjo}@ac.upc.es

Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona
Spain
Outline

- Introduction
- Overhead reduction techniques
- Results
- Analysis
- Conclusions
Hypermatrix structure
Overhead

Can store 0’s within data submatrices

• Storage
• Computation

Trade-off in data submatrix size

• BLAS3 efficiency
• (Useless) operation on 0’s
Reducing Overhead & Increasing Performance

- Efficient kernels which operate on small data submatrices
- Windows within data submatrices
Reducing Overhead & Increasing Performance

Efficient kernels which operate on small data submatrices
[Euro-Par’03]

• Fix parameters at compilation time
• Choose best algorithm
  – Loop unrolling factors
  – Loop orders
Matrix multiplication performance on small matrices

\[ C = C - A \cdot B^t \]

R10000 250 MHz (500 Mflop peak)
Hypermatrix Cholesky on problem PDS40

LP problem: Patient Distribution System (40 days)
Reducing Overhead & Increasing Performance

Windows within data submatrices

• Reduce storage

• Reduce computation
Reducing overhead: windows within dense data matrices

Data Submatrix

left column

right column

top row

bottom row

window

a)

b)
Context information

- MIPS R10000 @ 250 MHz (500 Mflop peak)
- Sequential code
- Data submatrices of fixed size
- Large problems solved In-Core
- Ordered using METIS
- Linear Programming problems
  - NetLib
  - Multicommodity Network Flow generators
## Matrix characteristics

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Dimension</th>
<th>NZs</th>
<th>Factor NZs</th>
<th>Density</th>
<th>Flops in factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRIDGEN1</td>
<td>330430</td>
<td>3162757</td>
<td>130586943</td>
<td>0.002</td>
<td>278891960665</td>
</tr>
<tr>
<td>QAP8</td>
<td>912</td>
<td>14864</td>
<td>193228</td>
<td>0.463</td>
<td>63764878</td>
</tr>
<tr>
<td>QAP12</td>
<td>3192</td>
<td>77784</td>
<td>2091706</td>
<td>0.410</td>
<td>2228094940</td>
</tr>
<tr>
<td>QAP15</td>
<td>6330</td>
<td>192405</td>
<td>8755465</td>
<td>0.436</td>
<td>20454297601</td>
</tr>
<tr>
<td>RMFGEN1</td>
<td>28077</td>
<td>151557</td>
<td>6469394</td>
<td>0.016</td>
<td>6323333044</td>
</tr>
<tr>
<td>TRIPART1</td>
<td>4238</td>
<td>80846</td>
<td>1147857</td>
<td>0.127</td>
<td>511884159</td>
</tr>
<tr>
<td>TRIPART2</td>
<td>19781</td>
<td>400229</td>
<td>5917820</td>
<td>0.030</td>
<td>2926231696</td>
</tr>
<tr>
<td>TRIPART3</td>
<td>38881</td>
<td>973881</td>
<td>17806642</td>
<td>0.023</td>
<td>14058214618</td>
</tr>
<tr>
<td>TRIPART4</td>
<td>56869</td>
<td>2407504</td>
<td>76805463</td>
<td>0.047</td>
<td>187168204525</td>
</tr>
<tr>
<td>pds1</td>
<td>1561</td>
<td>12165</td>
<td>37339</td>
<td>0.030</td>
<td>1850179</td>
</tr>
<tr>
<td>pds10</td>
<td>18612</td>
<td>148038</td>
<td>3384640</td>
<td>0.019</td>
<td>2519913926</td>
</tr>
<tr>
<td>pds20</td>
<td>38726</td>
<td>319041</td>
<td>10739539</td>
<td>0.014</td>
<td>13128744819</td>
</tr>
<tr>
<td>pds30</td>
<td>57193</td>
<td>463732</td>
<td>18216426</td>
<td>0.011</td>
<td>26262856180</td>
</tr>
<tr>
<td>pds40</td>
<td>76771</td>
<td>629851</td>
<td>27672127</td>
<td>0.009</td>
<td>43807548523</td>
</tr>
<tr>
<td>pds50</td>
<td>95936</td>
<td>791087</td>
<td>36321636</td>
<td>0.007</td>
<td>61180807800</td>
</tr>
<tr>
<td>pds60</td>
<td>115312</td>
<td>956906</td>
<td>46377926</td>
<td>0.006</td>
<td>81447389930</td>
</tr>
<tr>
<td>pds70</td>
<td>133236</td>
<td>1100254</td>
<td>54795729</td>
<td>0.006</td>
<td>100023696013</td>
</tr>
<tr>
<td>pds80</td>
<td>149558</td>
<td>1216223</td>
<td>64148298</td>
<td>0.005</td>
<td>125002360050</td>
</tr>
<tr>
<td>pds90</td>
<td>164944</td>
<td>1320298</td>
<td>70140993</td>
<td>0.005</td>
<td>138765323993</td>
</tr>
</tbody>
</table>
HM performance with and without windows
Reducing Overhead & Increasing Performance

Techniques used:

- Efficient kernels which operate on small data submatrices
  - Rectangular data submatrices
- Windows within data submatrices

Results:

- Improvement in Hypermatrix Cholesky
Performance: HM vs Ng-Peyton vs PSLDLT

![Graph showing performance comparison between HM, Ng-Peyton, and PSLDLT]

- **SN-LL (Ng-Peyton)**
- **HM**
- **PSLDLT**

###_legend

**Effective Mflops**

Performance of several sparse Cholesky factorization codes.
Current work

- Amalgamation

- Blocked sparse Cholesky code within SPLASH-2
  - Get it to work for large matrices
  - Get it to work in parallel
Increase in number of operations in sparse HM Cholesky (4x32 + windows).
HM flops per MxMt subroutine type
HM flops per MxMxT subroutine type

[Graph showing the percentage of flops performed by each subroutine type for different matrices and processors]

Optimization of a Sparse Hypermatrix Cholesky Factorization
Josep R. Herrero, Comp. Arch. Dept., UPC
Conclusions & Future Work

• Grouping of rows within submatrices could improve execution
  – Supernodal-HM

• Sparse HM Cholesky is competitive for large problems
  – OOC

• Good chances for exploiting parallelism
  – Parallel version