Reducing Overhead in 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
Introduction

Cholesky factorization

- Symmetric Positive Definite (SPD) matrix

Sparse matrices

- Plenty of 0’s
- Avoid storage and computation on 0’s
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
- Bit Vectors associated to data submatrices
- Windows within data submatrices
Efficient operation on small data submatrices

Goal:

- Reduce data submatrix size while keeping good BLAS3 performance

Idea [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 Mflops peak)
HM Cholesky: pds40

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

Operation on small data submatrices ... still has overhead

Goal: reduce the overhead further

- Bit Vectors associated to data submatrices
- Windows within data submatrices
Bit Vectors: Goal

Reduce unnecessary computation

- Avoid matrix multiplication when two submatrices produce no update upon a third one
Bit Vectors: Definition

One bit associated to a column in a data submatrix

- Value = 0 $\iff$ column is full of 0’s
- Value = 1 $\iff$ $\exists \geq 1$ NZ in column
Bit Vectors: Usage (I)

$BV_A \& BV_B = 0$: Operation can be skipped
Bit Vectors: Usage (II)

\[ \text{BV}_A \& \text{BV}_B \neq 0: \text{Operation must be performed} \]
Windows within data submatrices: Goal

Store and use only a part of a data submatrix

- Reduce unnecessary computation
- Reduce storage
Windows: Definition

Data Submatrix

top row

bottom row

left column

right column

Window: subset of data submatrix
Windows: Usage (I)

Operation can be reduced
Windows: Usage (II)

Operation can be skipped
Windows: Usage (III)

Unnecessary operation performed (Could be avoided with BVs)
Results: Context information

- MIPS R10000 @ 250 MHz (500 Mflops 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>NZs in L&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Density</th>
<th>Flops to factor&lt;sup&gt;b&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRIDGEN1</td>
<td>330430</td>
<td>3162757</td>
<td>130586943</td>
<td>0.002</td>
<td>278891</td>
</tr>
<tr>
<td>QAP8</td>
<td>912</td>
<td>14864</td>
<td>193228</td>
<td>0.463</td>
<td>63</td>
</tr>
<tr>
<td>QAP12</td>
<td>3192</td>
<td>77784</td>
<td>2091706</td>
<td>0.410</td>
<td>2228</td>
</tr>
<tr>
<td>QAP15</td>
<td>6330</td>
<td>192405</td>
<td>8755465</td>
<td>0.436</td>
<td>20454</td>
</tr>
<tr>
<td>RMFGEN1</td>
<td>28077</td>
<td>151557</td>
<td>6469394</td>
<td>0.016</td>
<td>6323</td>
</tr>
<tr>
<td>TRIPART1</td>
<td>4238</td>
<td>80846</td>
<td>1147857</td>
<td>0.127</td>
<td>511</td>
</tr>
<tr>
<td>TRIPART2</td>
<td>19781</td>
<td>400229</td>
<td>5917820</td>
<td>0.030</td>
<td>2926</td>
</tr>
<tr>
<td>TRIPART3</td>
<td>38881</td>
<td>973881</td>
<td>17806642</td>
<td>0.023</td>
<td>14058</td>
</tr>
<tr>
<td>TRIPART4</td>
<td>56869</td>
<td>2407504</td>
<td>76805463</td>
<td>0.047</td>
<td>187168</td>
</tr>
<tr>
<td>pds1</td>
<td>1561</td>
<td>12165</td>
<td>37339</td>
<td>0.030</td>
<td>1</td>
</tr>
<tr>
<td>pds10</td>
<td>18612</td>
<td>148038</td>
<td>3384640</td>
<td>0.019</td>
<td>2519</td>
</tr>
<tr>
<td>pds20</td>
<td>38726</td>
<td>319041</td>
<td>10739539</td>
<td>0.014</td>
<td>13128</td>
</tr>
<tr>
<td>pds30</td>
<td>57193</td>
<td>463732</td>
<td>18216426</td>
<td>0.011</td>
<td>26262</td>
</tr>
<tr>
<td>pds40</td>
<td>76771</td>
<td>629851</td>
<td>27672127</td>
<td>0.009</td>
<td>43807</td>
</tr>
<tr>
<td>pds50</td>
<td>95936</td>
<td>791087</td>
<td>36321636</td>
<td>0.007</td>
<td>61180</td>
</tr>
<tr>
<td>pds60</td>
<td>115312</td>
<td>956906</td>
<td>46377926</td>
<td>0.006</td>
<td>81447</td>
</tr>
<tr>
<td>pds70</td>
<td>133326</td>
<td>1100254</td>
<td>54795729</td>
<td>0.006</td>
<td>100023</td>
</tr>
<tr>
<td>pds80</td>
<td>149558</td>
<td>1216223</td>
<td>64148298</td>
<td>0.005</td>
<td>125002</td>
</tr>
<tr>
<td>pds90</td>
<td>164944</td>
<td>1320298</td>
<td>70140993</td>
<td>0.005</td>
<td>138765</td>
</tr>
</tbody>
</table>

- <sup>a</sup> Number of non-zeros in factor L (matrix ordered using METIS).
- <sup>b</sup> Number of floating point operations (in Millions) necessary to obtain L from the original matrix (ordered with METIS).
Performance of several HM codes
HM vs SN (Ng-Peyton): QAP matrix family
HM vs SN: PDS matrix family
HM vs SN performance: summary

SN vs HM

Effective Mflops

GRIDGEN1 QAP8 QAP12 QAP15 RMFGEN1 TRIPART1 TRIPART2 TRIPART3 TRIPART4 pds1 pds10 pds20 pds30 pds40 pds50 pds60 pds70 pds80 pds90

SN
HM
Summary

Techniques which improve Hypermatrix Cholesky performance:

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

Techniques which do not improve its performance:

- Bit Vectors
Conclusions & Future Work

- Sparse HM Cholesky is competitive for large problems
  - OOC
- Good chances for exploiting parallelism
  - Parallel version
Increase in number of operations in sparse HM Cholesky (4x32 + windows).
HM performance with and without windows
Performance of several sparse Cholesky factorization codes.
Current work

- Amalgamation
- Compare with Blocked sparse Cholesky code within SPLASH-2
  - Get it to work for large matrices
  - Get it to work in parallel