Sparse Hypermatrix Cholesky: In Search for High Performance

Josep Ramon Herrero

Computer Architecture Department
Universitat Politècnica de Catalunya

Talk at the School of Mathematics
The University of Edinburgh
May 12th, 2006
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization

Compiler-optimized inner kernels

Conclusions and future work
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization

Compiler-optimized inner kernels

Conclusions and future work
Motivation & General Goals

Motivation

- Matrix computations lie at the heart of many applications.

General Goal:

- Obtain efficient implementations of frequent matrix operations.
Specific Goals

Identify the key points for obtaining high performance.

Obtain efficient implementations ......

> of some frequent operations:
  > Sparse Cholesky factorization.
  > Dense Cholesky factorization.
  > Dense Matrix Multiplication.
  > Nearest Neighbor (NN) Classification.

> On different platforms.

Note:

> Focus on sequential code.
Overview

In search for high performance:

- **Efficiency of inner kernel** is of paramount importance.

Usual approach:

- Ad-hoc codes written in assembler.

Our approach:

- Compiler-optimized inner kernel for operation on small matrices
  - Collection of codes written in high level language;
  - Use compiler to generate optimized object code.
  - Insert best code in library: Small Matrix Library (SML).

Use SML routines for general codes.
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization

- SML kernels which operate on small data submatrices
- Bit Vectors associated to data submatrices
- Windows within data submatrices
- Amalgamation
- Ordering
- Results

Compiler-optimized inner kernels

Conclusions and future work
Hypermatrix (HM) Structure

Matrix

HyperMatrix
Hypermatrix (HM) Structure

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
- Amalgamation
Introduction

Sparse Hypermatrix Cholesky Factorization

SML kernels which operate on small data submatrices
Bit Vectors associated to data submatrices
Windows within data submatrices
Amalgamation
Ordering
Results

Compiler-optimized inner kernels

Conclusions and future work
Hypermatrix Cholesky on problem PDS40
LP problem: Patient Distribution System (40 days)

Effective Mflops = \( \frac{\# \text{flops (excluding operations on zeros)}}{\text{Time (including operations on zeros)}} \) \times 10^{-6}
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
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization

SML kernels which operate on small data submatrices

Bit Vectors associated to data submatrices

Windows within data submatrices

Amalgamation

Ordering

Results

Compiler-optimized inner kernels

Conclusions and future work
**Bit Vectors: Goal**

Reduce unnecessary computation

- Avoid matrix multiplication when two submatrices produce no update upon a third one

![Diagram](attachment:image.png)
Bit Vectors: Definition

One bit associated to a column in a data submatrix

- Value = 0 ⇔ column is full of 0’s
- Value = 1 ⇔ ∃ ≥ 1 NZ in column
Bit Vectors: Usage (I)

$B V_A \& B V_B = 0$: Operation can be skipped
Bit Vectors: Usage (II)

\[ BV_A \& BV_B \neq 0: \text{Operation must be performed} \]
Introduction

Sparse Hypermatrix Cholesky Factorization
- SML kernels which operate on small data submatrices
- Bit Vectors associated to data submatrices
- Windows within data submatrices
- Amalgamation
- Ordering
- Results

Compiler-optimized inner kernels

Conclusions and future work
Windows within data submatrices: Goal

Store and use only a part of a data submatrix

- Reduce unnecessary computation
- Reduce storage
Windows: Definition

Data Submatrix

left column

right column

top row

bottom row

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
- Large problems solved In-Core
- Ordered using METIS
- Post-order of Elimination Tree
- Linear Programming problems
  - NetLib
  - Multicommodity Network Flow generators
- Applications of Finite Element Method
  - NetLib
  - PERMAS
  - PARASOL
Performance: Block Size vs BVs vs Windows

HM performance

<table>
<thead>
<tr>
<th>Effective Mflops</th>
<th>QAP12</th>
<th>QAP15</th>
<th>TRIPART2</th>
<th>TRIPART4</th>
<th>pds30</th>
<th>pds60</th>
<th>pds90</th>
</tr>
</thead>
<tbody>
<tr>
<td>HM_8x8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HM_8x8+BV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HM_4x32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HM_4x32+BV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HM_4x32+win</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HM_4x32+win+BV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compiler-optimized inner kernels

Conclusions and future work
Increase in number of floating point operations in sparse HM Cholesky w.r.t. the minimum: windows reduce the number of operations on zeros.
Introduction

Sparse Hypermatrix Cholesky
In Search for High Performance

J.R. Herrero

HM vs SN (Ng-Peyton): QAP matrix family

HM_4x32+win
SN_cache2M_unr8
SN_cache1M_unr8
SN_cache512K_unr8
SN_cache32K_unr8
SN_cache32K_unr4

Compiler-optimized inner kernels

Conclusions and future work
Sparse Hypermatrix Cholesky: In Search for High Performance

J.R. Herrero

Introduction
Sparse Hypermatrix Cholesky
SML routines
Bit Vectors
Dense Windows
Amalgamation
Ordering
Results
Compiler-optimized inner kernels
Conclusions and future work

HM vs SN: TRIPART matrix family

Effective Mflops

TRIPART1
TRIPART2
TRIPART3
TRIPART4
Sparse Hypermatrix Cholesky: In Search for High Performance

J.R. Herrero

Introduction

Sparse Hypermatrix Cholesky

SML routines
Bit Vectors
Dense Windows
Amalgamation
Ordering
Results

Compiler-optimized inner kernels

Conclusions and future work

HM vs SN: PDS matrix family
HM vs SN performance: summary

SN vs HM

Effective Mflops

GRIDGEN1  QAP8  QAP12  QAP15  RMF/GEN1  TRIPART1  TRIPART2  TRIPART3  TRIPART4  pds1  pds10  pds20  pds30  pds40  pds50  pds60  pds70  pds80  pds90

SN  HM
Matrix multiplication: efficiency of codes

Less efficient

Most efficient
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization
  SML kernels which operate on small data submatrices
  Bit Vectors associated to data submatrices
  Windows within data submatrices
  Amalgamation
  Ordering
  Results

Compiler-optimized inner kernels

Conclusions and future work
Intra-Block Amalgamation: Original window

Data Submatrix

left column

right column

top row

bottom row

window
Intra-Block Amalgamation: column-wise

Data Submatrix

top row

bottom row

left column

right column

window

Compiler-optimized inner kernels
Intra-Block Amalgamation: row-wise

Data Submatrix

top row

bottom row

left column

right column

window
Intra-Block Amalgamation: row and column-wise

Data Submatrix

top row

bottom row

left column

right column

window

36
Results: QAP8
Intra-Block Amalgamation

Effective Mflops

Columns amalgamated

- amr=0
- amr=1
- amr=2
- amr=3
Results: QAP12
Intra-Block Amalgamation

Effective Mflops

Columns amalgamated

- amr=0
- amr=1
- amr=2
- amr=3
Results: TRIPART1
Intra-Block Amalgamation

Effective Mflops

Columns amalgamated
Results: TRIPART2
Intra-Block Amalgamation

Effective Mflops

Columns amalgamated

- amr=0
- amr=1
- amr=2
- amr=3

J.R. Herrero

Introduction
Sparse Hypermatrix Cholesky: In Search for High Performance
SML routines
Bit Vectors
Dense Windows
Amalgamation
Ordering
Results
Compiler-optimized inner kernels
Conclusions and future work
Results: pds10
Intra-Block Amalgamation
Results: pds20
Intra-Block Amalgamation
Results: Original (without amalgamation) vs Intra-block amalgamation

![Bar chart showing effective Mflops comparison between Original and Intra-block amalgamation for various problems with different sizes and dimensions.](image)
Intra-Block Amalgamation on Itanium2: Matrix pds20
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization
- SML kernels which operate on small data submatrices
- Bit Vectors associated to data submatrices
- Windows within data submatrices
- Amalgamation
- Ordering

Results

Compiler-optimized inner kernels

Conclusions and future work
Ordering sparse matrices with METIS

Number of iterations necessary to amortize cost of improved ordering.
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization
  SML kernels which operate on small data submatrices
  Bit Vectors associated to data submatrices
  Windows within data submatrices
  Amalgamation
  Ordering

Results

Compiler-optimized inner kernels

Conclusions and future work
Performance of several sparse Cholesky codes: IPM

Introduction

Sparse Hypermatrix Cholesky: In Search for High Performance

J.R. Herrero

Results

Compiler-optimized inner kernels

Conclusions and future work
Performance of several sparse Cholesky codes: FEM

![Graph showing performance comparison of various codes]

- **SN-LL (Taucs)**
- **SN-MF (Taucs)**
- **HM**

**Details**

**Introduction**

**Sparse Hypermatrix Cholesky: In Search for High Performance**

**J.R. Herrero**

**Results**

- Compiler-optimized inner kernels

**Conclusions and future work**
Overhead in number of operations in sparse HM Cholesky (4x32 + windows)
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization

Compiler-optimized inner kernels

Conclusions and future work
Compiler-optimized inner kernels

Facts:

- Need for high performance inner kernels
- High cost in creation of such kernels by hand
- Compiler Optimization is a mature field

Approach:

- Smooth the way to the compiler
  - Creation of a Small Matrix Library (SML)
Generalization

\[ C = \beta C + \alpha \text{op}(A) \times \text{op}(B) \]

- \(\alpha\) and \(\beta\) are scalars
- \(\text{op}(A)\) is \(A\) or \(A^t\).

**Table:** Peak Mflops of inner kernel on a Pentium 4 Xeon Northwood.

<table>
<thead>
<tr>
<th></th>
<th>(A \times B^t)</th>
<th>(A^t \times B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No align</td>
<td>3334</td>
<td>3220</td>
</tr>
<tr>
<td>Align</td>
<td>3457</td>
<td>3810</td>
</tr>
</tbody>
</table>
Efficient Inner Kernels: Conclusions

\[ C = C + \alpha A^t \times B \] is appealing:
- access to all three matrices with stride one;
- stores to matrix C can be hoisted from the inner loop

Compilers can perform efficient optimizations on regular codes. We can facilitate this by:
- Providing matrix leading dimensions and loop trip counts at compilation time;
- Trying several variants of code: different loop orders, unroll factors.

The resulting code can be more efficient if:
- Matrices are aligned;
- All matrices are accessed with stride one;
- Store operations are removed from the inner kernel.
Outline

Introduction

Sparse Hypermatrix Cholesky Factorization

Compiler-optimized inner kernels

Conclusions and future work
Conclusions

- Current compilers can generate very efficient codes when working on simple and **regular codes**.

- There is a trade-off between the speed of an algorithm and:
  - computation of non productive operations.
  - more regular codes

- Fundamental aspects to achieve high performance:
  - data accessed with stride one;
  - data properly aligned;
  - store operations removed from the innermost loop.
Conclusions

Sparse hypermatrix Cholesky factorization

- The most effective overhead reduction techniques have been:
  - Use of SML routines working on rectangles
  - Use of dense windows within data submatrices
  - Use of Intra-Block Amalgamation
Future Work

- Improve creation of SML routines using more realistic access patterns;
- Implement HM Cholesky on matrix $U$ to increase number of accesses with stride one;
- Allow for storage of data submatrices as supernodes;
- Experiment with an Incomplete Cholesky factorization;
- ...
Sparse Hypermatrix Cholesky: In Search for High Performance

Josep Ramon Herrero

Computer Architecture Department
Universitat Politècnica de Catalunya

Talk at the School of Mathematics
The University of Edinburgh
May 12th, 2006
Further Details I

Further Details
## IPM: 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).
## FEM: 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>bear</td>
<td>25906</td>
<td>412447</td>
<td>3278225</td>
<td>0.009</td>
<td>847</td>
</tr>
<tr>
<td>rail</td>
<td>11783</td>
<td>799545</td>
<td>3768886</td>
<td>0.054</td>
<td>1594</td>
</tr>
<tr>
<td>methan</td>
<td>48162</td>
<td>1234332</td>
<td>16631801</td>
<td>0.014</td>
<td>10493</td>
</tr>
<tr>
<td>nasasrb</td>
<td>54870</td>
<td>1366097</td>
<td>10489476</td>
<td>0.007</td>
<td>3496</td>
</tr>
<tr>
<td>cfd1</td>
<td>70656</td>
<td>949510</td>
<td>20910296</td>
<td>0.008</td>
<td>13523</td>
</tr>
<tr>
<td>cfd2</td>
<td>123440</td>
<td>1605669</td>
<td>37696869</td>
<td>0.005</td>
<td>31218</td>
</tr>
<tr>
<td>inline_1</td>
<td>503712</td>
<td>18660027</td>
<td>174608135</td>
<td>0.001</td>
<td>150974</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).
Sparse Hypermatrix Cholesky: In Search for High Performance

J.R. Herrero

Appendix
Further Details

HM flops per MxMt subroutine type
Calls, flops & time per $A \times B^T$ subroutine type
QAP8 and QAP12

IPM matrices
Sparse HM Cholesky: flops per $M \times M^t$

subroutine type

IPM & FEM