Automatic Performance Tuning and Analysis of Sparse Triangular Solve

Richard Vuduc, Shoaib Kamil, Jen Hsu, Rajesh Nishtala
James W. Demmel, Katherine A. Yelick
June 22, 2002

Berkeley Benchmarking and OPtimization (BeBOP) Project

www.cs.berkeley.edu/~richie/bebop

Computer Science Division, U.C. Berkeley
Application performance dominated by a few computational kernels
- Solving PDEs (linear algebra ops)
- Google (sparse matrix-vector multiply)
- Multimedia (signal processing)

Performance tuning today
- Vendor-tuned standardized libraries (e.g., BLAS)
- User tunes by hand

Automated tuning for dense linear algebra, FFTs, …
- PHiPAC/ATLAS (dense linear algebra)
- FFTW/SPIRAL/UHFFT (signal processing)
Problem Area: Sparse Matrix Kernels

Approach to automatic tuning: for each kernel,
- Identify and generate a space of implementations
- Search (models, experiments) to find the fastest one

Performance issues in the sparse linear algebra
- High bandwidth requirements and poor instruction mix
- Depends on architecture, kernel, and matrix
- How to select data structures, algorithms?
- Need for run-time transformations

Early success: SPARSITY (Im & Yelick ’99) for sparse matrix-vector multiply (SpM×V)

This talk: Sparse triangular solve (SpTS)
Sparse Triangular Matrix Example

raefsky4
(structural problem)
+ SuperLU 2.0 +
colmmd

Dimension: 19779

No. non-zeros:
12.6 M

Dense trailing triangle:
dim=2268
20% of total nnz
Idea: Sparse/Dense Partitioning

Partition the matrix into sparse \( (L_1, L_2) \) and dense \( (L_D) \) parts:

\[
\begin{pmatrix}
L_1 & \\
L_2 & L_D
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix} =
\begin{pmatrix}
y_1 \\
y_2
\end{pmatrix}
\]

Leads to 1 SpTS, 1 SpM\(\times\)V, and 1 Dense TS:

\[
L_1x_1 = y_1 \quad (1)
\]

\[
\hat{y}_2 = y_2 - L_2x_1 \quad (2)
\]

\[
L_Dx_2 = \hat{y}_2 \quad (3)
\]

SPARSITY optimizations for (1)–(2); tuned BLAS for (3).
Register Blocking (SPARSITY)

- Store $r \times c$ dense blocks
- Multiply/solve block-by-block
- Fill in explicit zeros
- 1.3x–2.5x speedup on FEM matrices ($\text{SpM} \times \text{V}$)
- Reduced storage overhead over, e.g., CSR
- Block ops are fully unrolled – improves register reuse
- Trade-off extra computation for efficiency
Tuning Parameter Selection

First, select *switch point*, \( s \); at run-time:

- Assume matrix in CSR format on input
- Scan bottom row from diag. until two consecutive zeros found
- Fill vs. efficiency trade-off

Then, select *register block size*, \( b \)

- Maximize, over all \( b \),

\[
\frac{\text{Dense matrix in sparse format performance (Mflop/s) at } b}{\text{Estimated fill ratio at } b}
\]  

(4)

Total cost to select and reorg.: *e.g.*, 10–30 naïve solves on Itanium
Performance Bounds

- Upper-bounds on performance (Mflop/s)?
- Full latency cost model of execution time:

\[ T(b, s) = \sum_{i=1}^{\kappa-1} H_i(b, s) \alpha_i + M_\kappa(b, s) \alpha_{\text{mem}} \]  

(5)

- Lower bound on misses: ignore conflict misses on vectors

\[ M^{(i)}_{\text{lower}}(b, s) = \frac{1}{l_i} \left[ k f_b \left( 1 + \frac{1}{\gamma b^2} \right) + \frac{1}{\gamma} \left( \left\lceil \frac{n}{b} \right\rceil + 1 \right) + \left( 2n + \frac{(n-s)((n-s)+1)}{2} \right) \right] \]  

(6)
Miss Model Validation: Intel Itanium

Sparse Triangular Solve: L2 Misses — [itanium–linux–ecc]

Upper bound
PAPI
Lower bound

Sparse Triangular Solve: L3 Misses — [itanium–linux–ecc]
Performance Results: Intel Itanium

Sparse Triangular Solve: Performance Summary — [itanium–linux–ecc]

- Reference
- Reg. Blocking (RB)
- Switch-to-Dense (S2D)
- RB + S2D
- Analytic upper bound
- Analytic lower bound
- PAPI upper bound
Performance Results: Sun Ultra 2i

Sparse Triangular Solve: Performance Summary — [ultra-solaris]

- Reference
- REG. BLOCKING (RB)
- SWITCH-TO-DENSE (S2D)
- RB + S2D
- Analytic upper bound
- Analytic lower bound
- PAPI upper bound

Matrix:
- dense
- memplus
- wang4
- ex11
- raefsky4
- goodwin
- lhr10

Performance (Mflop/s)
Performance Results: IBM Power3

Sparse Triangular Solve: Performance Summary -- [power3-aix]

- Reference
- Reg. Blocking (RB)
- Switch-to-Dense (S2D)
- RB + S2D
- Analytic upper bound
- Analytic lower bound
- PAPI upper bound
Conclusions and Directions

- Limits of “low-level” tuning are near
  - Can we approach bandwidth limits?
  - Other kernels? $A^T A x$, $A^k x$, $RAR^T$
  - Other structures? multiple vectors, symmetry, reordering

- Interface from/to libraries and applications?
  - Leverage existing generators (e.g., Bernoulli)
  - Hybrid on-line, off-line optimizations

- SpTS-specific future work
  - symbolic structure; other fill-reducing orderings
  - refinements to switch point selection
  - Incomplete Cholesky and LU preconditioners
Related Work (1/R)

Automatic tuning systems
- PHiPAC/ATLAS/SPARSITY
- FFTW/SPIRAL/UHFFT
- MPI collective ops (Vadhiyar, et al., 2000)

Code generation
- FLAME (Gunnels and van de Geijn, 2000)
- Sparse compilers (Bik; Bernoulli)
- Generic programming (Blitz++, POOMA, MTL, GMCL, ...)

Sparse performance modeling
- Temam and Jalby (‘92)
- White and Sadayappan (‘97)
- Navarro (‘96); Heras (‘99); Fraguela (‘99)
Related Work (2/R)

- Compilers (analysis and models); run-time selection
  - CROPS (UCSD/Carter, Ferrante, et al.)
  - TUNE (Chatterjee, et al.)
  - Iterative compilation (O’Boyle, et al., 1998)
  - Broadway (Guyer and Lin, ’99)
  - Brewer (’95); ADAPT (Voss, 2000)

- Interfaces: Sparse BLAS; PSBLAS; PETSc

- Sparse triangular solve
  - SuperLU/MUMPS/SPOOLES/UMFPACK/PSPASES…
  - Approximation: Alvarado (’93); Raghavan (’98)
  - Scalability: Rothberg (’92;’95); Gupta (’95); Li, Coleman (’88)
<table>
<thead>
<tr>
<th>Name</th>
<th>Application Area</th>
<th>Dim.</th>
<th>Nnz in L</th>
<th>Dim.</th>
<th>Density</th>
<th>% Total Nnz</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense</td>
<td>Dense matrix</td>
<td>1000</td>
<td>500k</td>
<td>1000</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>memplus</td>
<td>Circuit simulation</td>
<td>17758</td>
<td>2.0M</td>
<td>1978</td>
<td>97.7%</td>
<td>96.8%</td>
</tr>
<tr>
<td>wang4</td>
<td>Device simulation</td>
<td>26068</td>
<td>15.1M</td>
<td>2810</td>
<td>95.0%</td>
<td>24.8%</td>
</tr>
<tr>
<td>ex11</td>
<td>Fluid flow</td>
<td>16614</td>
<td>9.8M</td>
<td>2207</td>
<td>88.0%</td>
<td>22.0%</td>
</tr>
<tr>
<td>raeisky4</td>
<td>Structural mechanics</td>
<td>19779</td>
<td>12.6M</td>
<td>2268</td>
<td>100.0%</td>
<td>20.4%</td>
</tr>
<tr>
<td>goodwin</td>
<td>Fluid mechanics</td>
<td>7320</td>
<td>1.0M</td>
<td>456</td>
<td>65.9%</td>
<td>6.97%</td>
</tr>
<tr>
<td>lhr10</td>
<td>Chemical processes</td>
<td>10672</td>
<td>369k</td>
<td>104</td>
<td>96.3%</td>
<td>1.43%</td>
</tr>
</tbody>
</table>
# Register Profile (Intel Itanium)

Register Blocking Performance (Mflop/s) [Dense (n=1000); itanium–linux–ecc]

<table>
<thead>
<tr>
<th>Column Block Size (c)</th>
<th>Row Block Size (r)</th>
<th>Performance (Mflop/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.49</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1.49</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.49</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.49</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1.55</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1.55</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1.55</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.55</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1.55</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1.55</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1.48</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1.53</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1.48</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1.42</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1.45</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1.40</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1.45</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1.46</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1.46</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1.45</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1.45</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1.46</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1.46</td>
</tr>
</tbody>
</table>
Register Profile (IBM Power3)

Register Blocking Performance (Mflop/s) [Dense (n=1000); power3-aix]
Register Profile (Sun Ultra 2i)

Register Blocking Performance (Mflop/s) [ Dense (n=1000); ultra-solaris]

<table>
<thead>
<tr>
<th>row block size (r)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>column block size (c)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Values in the grid represent performance in Mflop/s.
Register Profile (Intel Pentium III)

Register Blocking Performance (Mflop/s) [ Dense (n=1000); pentium3–linux–icc]
Miss Model Validation: Sun Ultra 2i

Sparse Triangular Solve: L1 Misses --- [ultra-solaris]

Upper bound
PAPI
Lower bound

No. of misses (millions)

Matrix

dense memplus wang4 ex11 Matrix raefsky4 goodwin lhr10

Sparse Triangular Solve: L2 Misses --- [ultra-solaris]

Upper bound
PAPI
Lower bound

No. of misses (millions)

Matrix

dense memplus wang4 ex11 Matrix raefsky4 goodwin lhr10
Miss Model Validation: IBM Power3

Sparse Triangular Solve: L1 Misses — [power3-aix]

Sparse Triangular Solve: L2 Misses — [power3-aix]
L1 Cache Misses: Intel Itanium

Sparse Triangular Solve: L1 Misses -- [itanium-linux-ecc]

- Upper bound
- PAPI
- Lower bound

- No. of misses (millions)

- dense  memplus  wang4  ex11  Matrix  raefsky4  goodwin  lhr10

- 0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 3 3.2 3.4

Automatic Performance Tuning and Analysis of Sparse Triangular Solve -- p.24/31
Dense Triangle Density: Dense Matrix

Density of Trailing Submatrix [dense-L]

- Trailing submatrix density (%)
- Heuristic switch point

Column number (normalized)
Dense Triangle Density: memplus

Density of Trailing Submatrix \([\text{memplus} - L]\)

- Column number (normalized)
- Density

Heuristic switch point
Dense Triangle Density: \textit{wang4}

![Graph showing density of trailing submatrix](image-url)
Dense Triangle Density: \textbf{ex11}
Dense Triangle Density: $raefsky4$

![Graph](image)
Dense Triangle Density: goodwin

Density of Trailing Submatrix [goodwin-L]

- Trailing submatrix density (%)
- Heuristic switch point

Column number (normalized)

Density
Dense Triangle Density: *lhr10*

![Graph showing the density of trailing submatrix for *lhr10*](image)