Optimizations & Bounds for Sparse Symmetric Matrix-Vector Multiply

Berkeley Benchmarking and Optimization Group (BeBOP)
http://bebop.cs.berkeley.edu

Benjamin C. Lee, Richard W. Vuduc, James W. Demmel, Katherine A. Yelick
University of California, Berkeley

27 February 2004
Outline

- Performance Tuning Challenges
- Performance Optimizations
  - Matrix Symmetry
  - Register Blocking
  - Multiple Vectors
- Performance Bounds Models
- Experimental Evaluation
  - 7.3x max speedup over reference (median: 4.2x)
- Conclusions
Computational Kernels

- Sparse Matrix-Vector Multiply (SpMV): \( y \leftarrow y + A \cdot x \)
  - \( A \): Sparse matrix, symmetric (i.e. \( A = A^T \))
  - \( x, y \): Dense vectors

- Sparse Matrix-Multiple Vector Multiply (SpMM): \( Y \leftarrow Y + A \cdot X \)
  - \( X, Y \): Dense matrices

Performance Tuning Challenges

- Sparse code characteristics
  - High bandwidth requirements (matrix storage overhead)
  - Poor locality (indirect, irregular memory access)
  - Poor instruction mix (low ratio of flops to memory operations)

- SpMV performance less than 10% of machine peak
- Performance depends on kernel, matrix and architecture
Optimizations: Matrix Symmetry

Symmetric Storage
- Assume compressed sparse row (CSR) storage
- Store half the matrix entries (e.g., upper triangle)

Performance Implications
- Same flops
- Halves memory accesses to the matrix
- Same irregular, indirect memory accesses
  - For each stored non-zero $A(i, j)
    - $y(i) += A(i, j) * x(j)$
    - $y(j) += A(i, j) * x(i)$
- Special consideration of diagonal elements
Optimizations: Register Blocking (1/3)
Optimizations: Register Blocking (2/3)

- BCSR with uniform, aligned grid
Fill-in zeros: Trade extra flops for better blocked efficiency

- In this example: 1.5x speedup with 50% fill on Pentium III
Optimizations: Multiple Vectors

Performance Implications

- Reduces loop overhead
- Amortizes the cost of reading $A$ for $v$ vectors
Optimizations: Register Usage (1/3)

- Register Blocking
  - Assume column-wise unrolled block multiply
  - Destination vector elements in registers (r)
Optimizations: Register Usage (2/3)

- Symmetric Storage
  - Doubles register usage (2r)
    - Destination vector elements for stored block
    - Source vector elements for transpose block
Optimizations: Register Usage (3/3)

- **Vector Blocking**
  - Scales register usage by vector width ($2rv$)
Performance Models

- **Upper Bound on Performance**
  - Evaluate quality of optimized code against bound

- **Model Characteristics and Assumptions**
  - Consider only the cost of memory operations
  - Accounts for minimum effective cache and memory latencies
  - Consider only compulsory misses (i.e. ignore conflict misses)
  - Ignores TLB misses

- **Execution Time Model**
  - Cache misses are modeled and verified with hardware counters
  - Charge $\alpha_i$ for hits at each cache level
    - $T = (L1\ hits) \ \alpha_1 + (L2\ hits) \ \alpha_2 + (Mem\ hits) \ \alpha_{mem}$
    - $T = (Loads) \ \alpha_f + (L1\ misses)(\alpha_2 - \alpha_f) + (L2\ misses)(\alpha_{mem} - \alpha_2)$
Evaluation: Methodology

- Four Platforms
  - Sun Ultra 2i, Intel Itanium, Intel Itanium 2, IBM Power 4

- Matrix Test Suite
  - Twelve matrices
  - Dense, Finite Element, Assorted, Linear Programming

- Reference Implementation
  - Non-symmetric storage
  - No register blocking (CSR)
  - Single vector multiplication
Evaluation: Observations

- **Performance**
  - 2.6x max speedup (median: 1.1x) from symmetry
    - {Symmetric BCSR Multiple Vector} vs. {Non-Symmetric BCSR Multiple Vector}
  - 7.3x max speedup (median: 4.2x) from combined optimizations
    - {Symmetric BCSR Multiple Vector} vs. {Non-symmetric CSR Single Vector}

- **Storage**
  - 64.7% max savings (median: 56.5%) in storage
    - Savings > 50% possible when combined with register blocking
  - 9.9% increase in storage for a few cases
    - Increases possible when register block size results in significant fill

- **Performance Bounds**
  - Measured performance achieves 68% of PAPI bound, on average
Performance Results: Sun Ultra 2i
Performance Results: Sun Ultra 2i
Performance Results: Sun Ultra 2i
Performance Results: Sun Ultra 2i
Performance Results: Intel Itanium 1

[Graph showing performance results for Intel Itanium 1 with various markers representing different performance metrics and matrix sizes.]
Performance Results: Intel Itanium 2
Performance Results: IBM Power 4
Conclusions

- **Matrix Symmetry Optimizations**
  - Symmetric Performance: 2.6x speedup (median: 1.1x)
    - {Symmetric BCSR Multiple Vector} vs. {Non-Symmetric BCSR Multiple Vector}
  - Overall Performance: 7.3x speedup (median: 4.15x)
    - {Symmetric BCSR Multiple Vector} vs. {Non-symmetric CSR Single Vector}
  - Symmetric Storage: 64.7% savings (median: 56.5%)
  - Cumulative performance effects
  - Trade-off between optimizations for register usage

- **Performance Modeling**
  - Models account for symmetry, register blocking, multiple vectors
  - Gap between measured and predicted performance
    - Measured performance is 68% of predicted performance (PAPI)
    - Model refinements are future work
Current & Future Directions

- **Heuristic Tuning Parameter Selection**
  - Register block size and vector width chosen independently
  - Heuristic to select parameters simultaneously

- **Automatic Code Generation**
  - Automatic tuning techniques to explore larger optimization spaces
  - Parameterized code generators

- **Related Optimizations**
  - Symmetry (Structural, Skew, Hermitian, Skew Hermitian)
  - Cache Blocking
  - Field Interlacing
Related Work

- **Automatic Tuning Systems and Code Generation**
  - PHiPAC [BACD97], ATLAS [WPD01], SPARSITY[Im00]
  - FFTW [FJ98], SPIRAL[PSVM01], UHFFT[MMJ00]
  - MPI collective ops (Vadhiyar, et al. [VFD01])
  - Sparse compilers (Bik [BW99], Bernoulli [Sto97])

- **Sparse Performance Modeling and Tuning**
  - Temam and Jalby [TJ92]
  - Toledo [Tol97], White and Sadayappan [WS97], Pinar [PH99]
  - Navarro [NGLPJ96], Heras [HPDR99], Fraguela [FDZ99]
  - Gropp, et al. [GKKS99], Geus [GR99]

- **Sparse Kernel Interfaces**
  - Sparse BLAS Standard [BCD+01]
  - NIST SparseBLAS [RP96], SPARSKIT [Saa94], PSBLAS [FC00]
  - PETSc
Symmetric Register Blocking

Square Diagonal Blocking
- Adaptation of register blocking for symmetry
- Register blocks – $r \times c$
  - Aligned to the right edge of the matrix
- Diagonal blocks – $r \times r$
  - Elements below the diagonal are not included in diagonal block
- Degenerate blocks – $r \times c'$
  - $c' < c$ and $c'$ depends on the block row
  - Inserted as necessary to align register blocks

Register Blocks – 2 x 3
Diagonal Blocks – 2 x 2
Degenerate Blocks – Variable
Multiple Vectors Dispatch Algorithm

- **Dispatch Algorithm**
  - $k$ vectors are processed in groups of the vector width ($v$)
    - SpMM kernel contains $v$ subroutines: $SR_i$ for $1 \leq i \leq v$
    - $SR_i$ unrolls the multiplication of each matrix element by $i$
  - Dispatch algorithm, assuming vector width $v$
    - Invoke $SR_v \lfloor k/v \rfloor$ times
    - Invoke $SR_{k \mod v}$ once, if $k \mod v > 0$
References (1/3)


References (2/3)


References (3/3)


