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

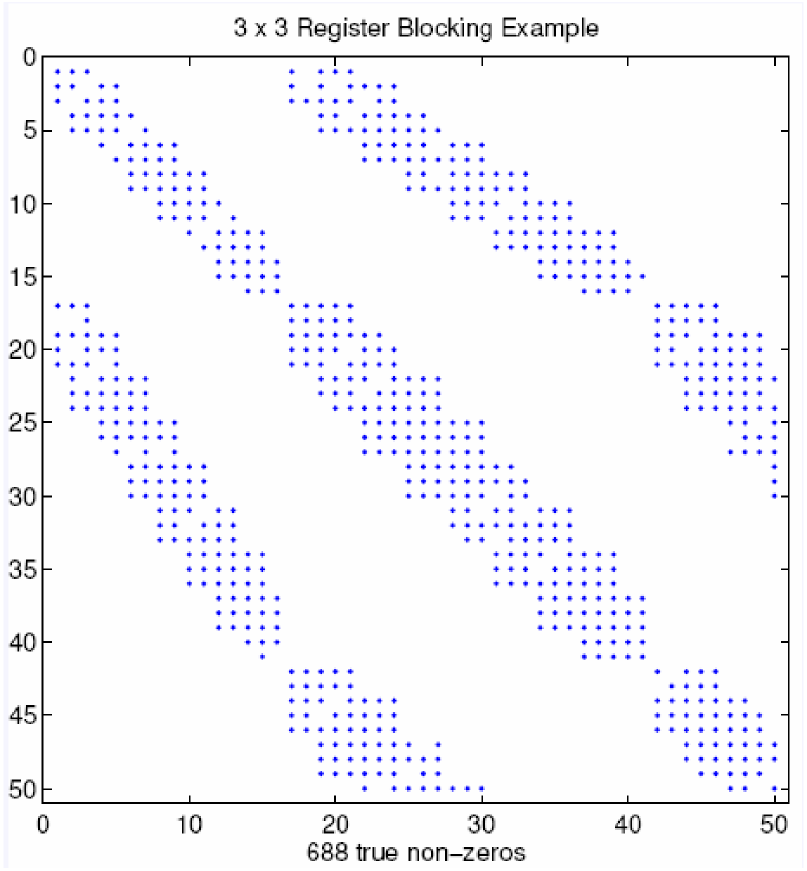
- **Performance Tuning Challenges**
- **Performance Optimizations**
  - Register Blocking
  - Matrix Symmetry
  - Multiple Vectors
- **Performance Bounds Models**
- **Experimental Evaluation**
  - 7.3x max speedup over reference (median: 4.2x)
- **Conclusions**

Introduction & Background

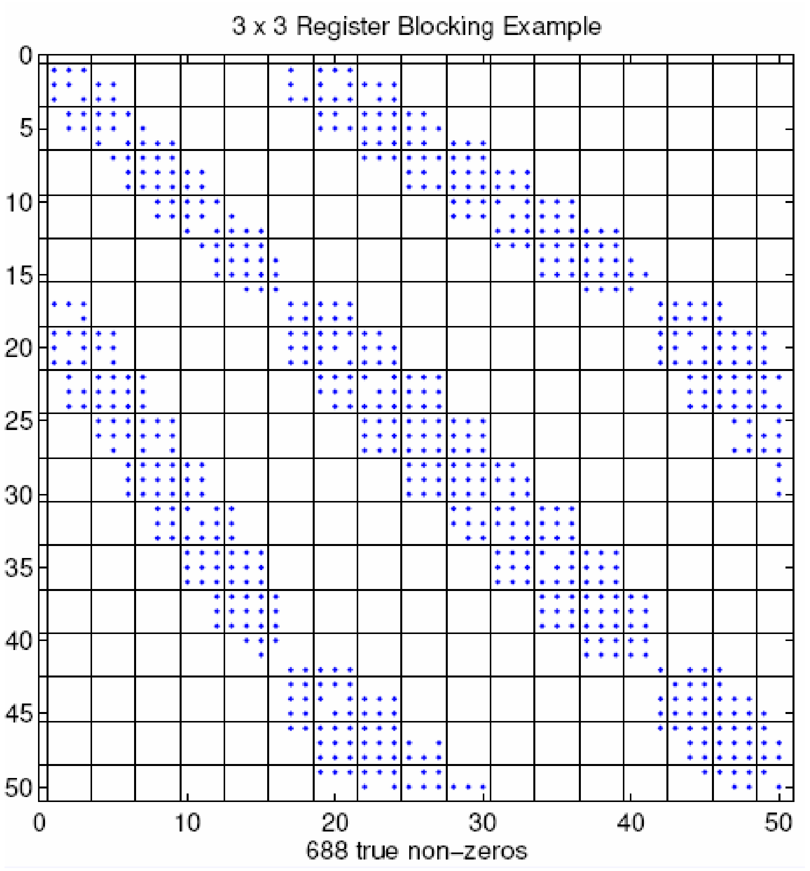
- **Computational Kernels**
  - Sparse Matrix-Vector Multiply (SpMV): $\boldsymbol{y}$ **?** $\boldsymbol{y} + \boldsymbol{A} \bullet \boldsymbol{x}$
    - $\boldsymbol{A}$: Sparse matrix, symmetric (***i.e.*** $\boldsymbol{A} = \boldsymbol{A}^T$)
    - $\boldsymbol{x},\boldsymbol{y}$: Dense vectors
  - Sparse Matrix-Multiple Vector Multiply (SpMM): $\boldsymbol{Y}$ **?** $\boldsymbol{Y} + \boldsymbol{A} \bullet \boldsymbol{X}$
    - $\boldsymbol{X},\boldsymbol{Y}$: 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: Register Blocking (1/3)

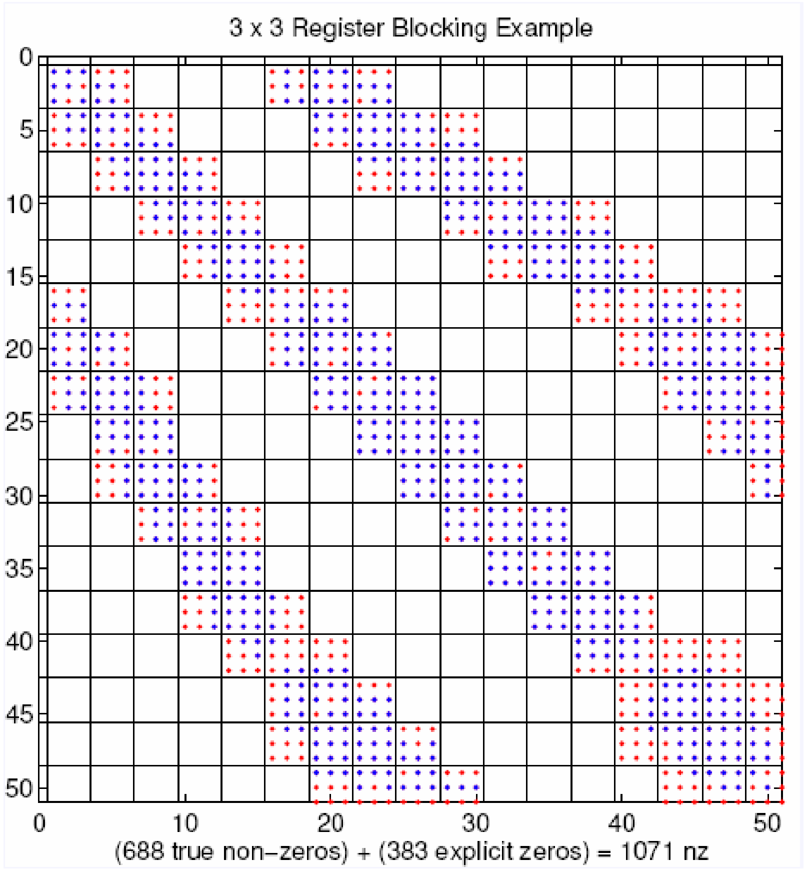


Optimizations: Register Blocking (2/3)



- BCSR with uniform, aligned grid

Optimizations: Register Blocking (3/3)



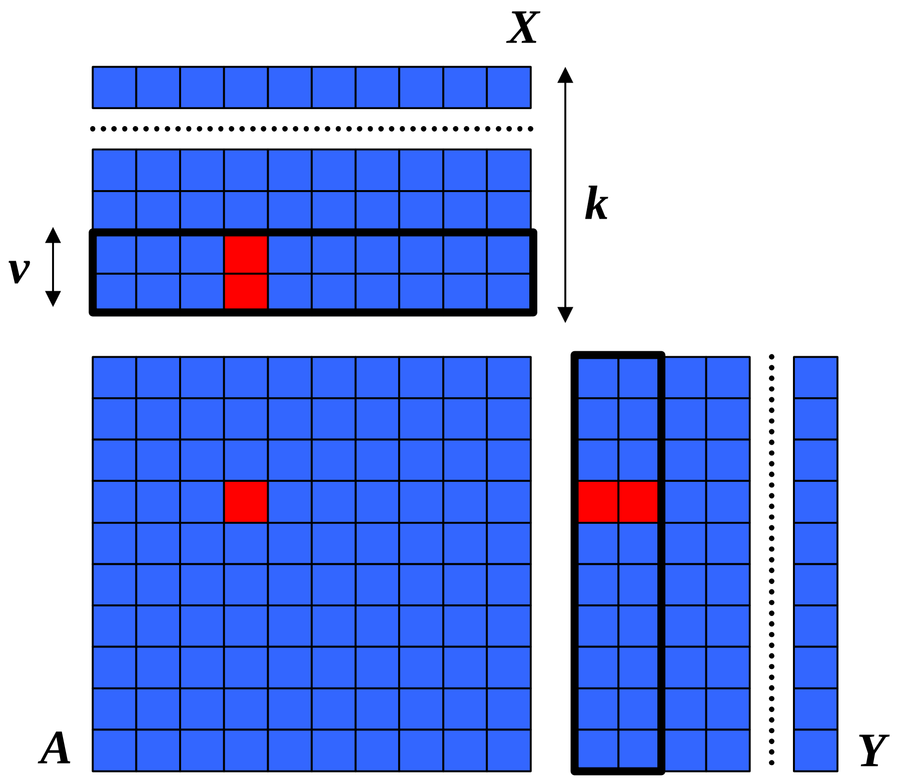
- Fill-in zeros: Trade extra flops for better blocked efficiency

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 $A(i, j)$
      - $y(i) \mathrel{+}= A(i, j) * x(j)$
      - $y(j) \mathrel{+}= A(i, j) * x(i)$
  - Special consideration of diagonal elements

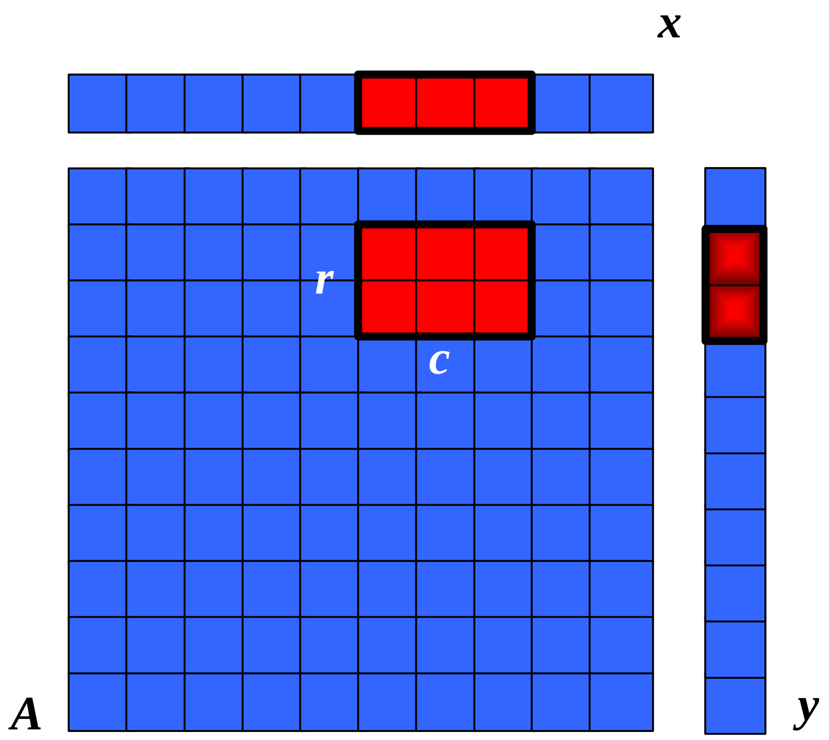
Optimizations: Multiple Vectors

- **Performance Implications**
  - Reduces loop overhead
  - Amortizes the cost of reading A for v vectors



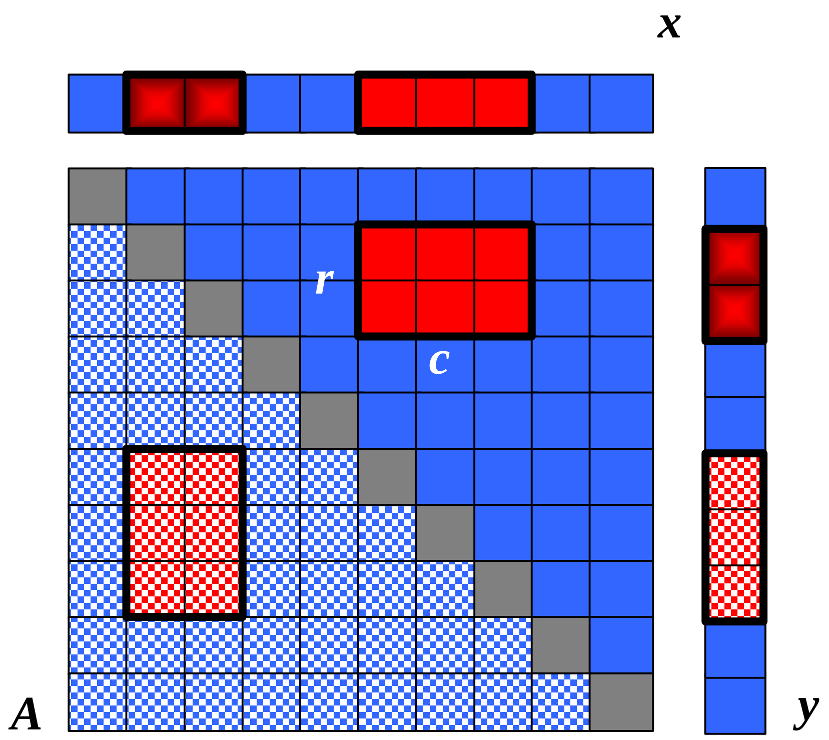
Optimizations: Register Usage (1/3)

- **Register Blocking**
  - Assume column-wise unrolled block multiply
  - Destination vector elements in registers (*r*)



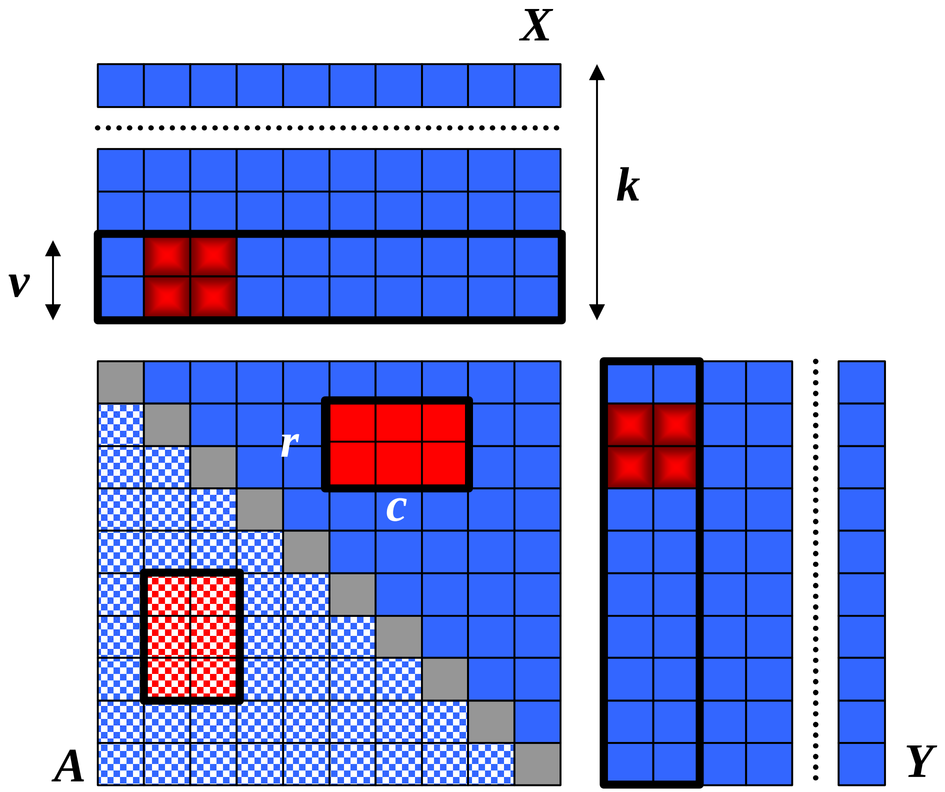
Optimizations: Register Usage (2/3)

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

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

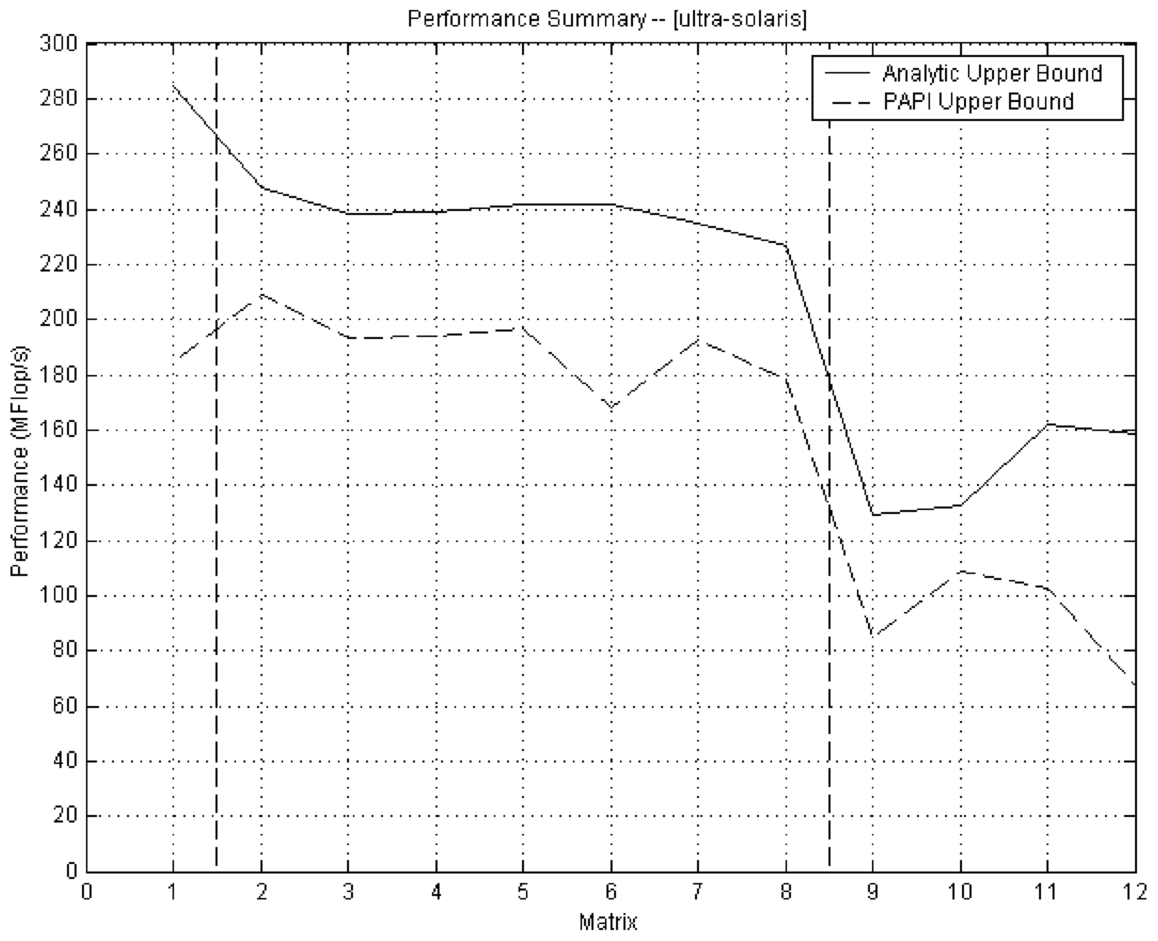
Evaluation: Methodology

- **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

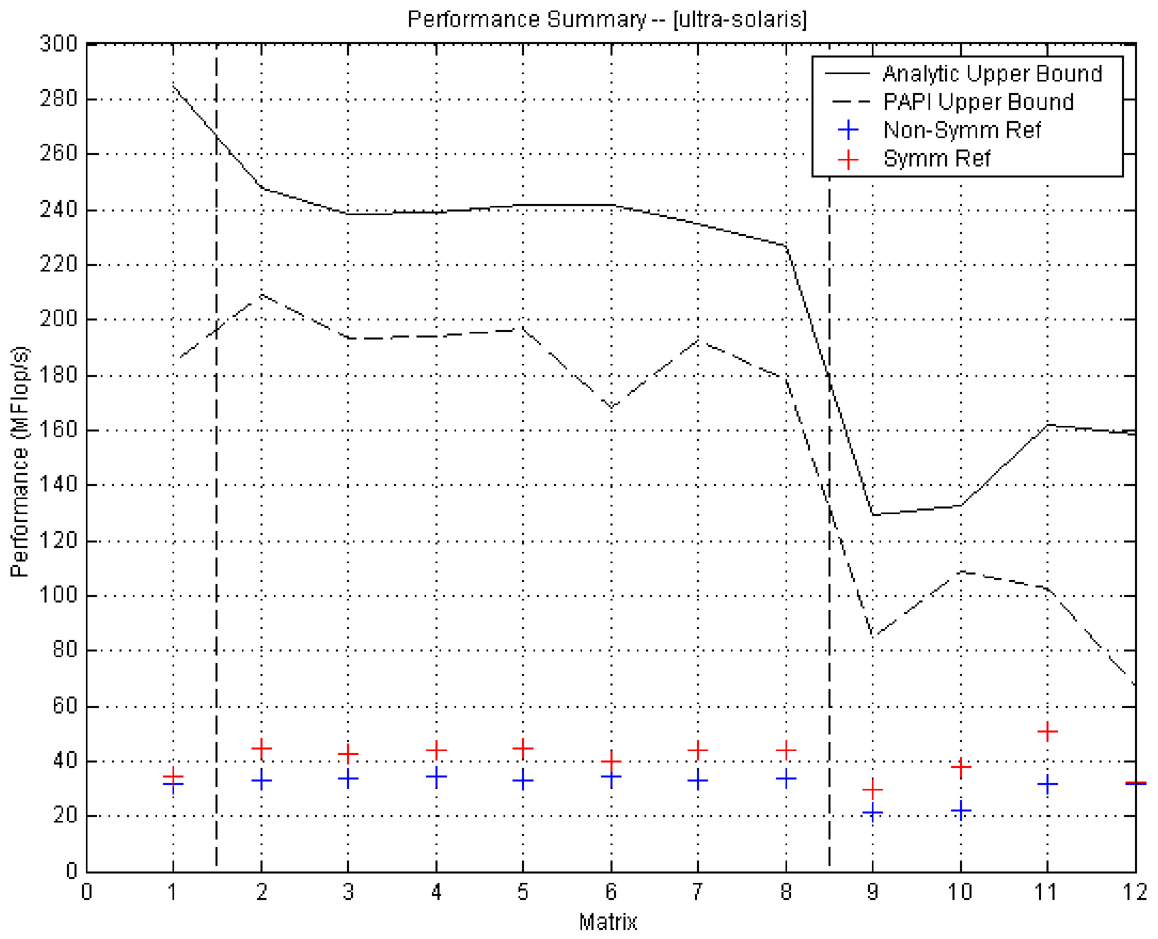
Evaluation: Observations

- **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

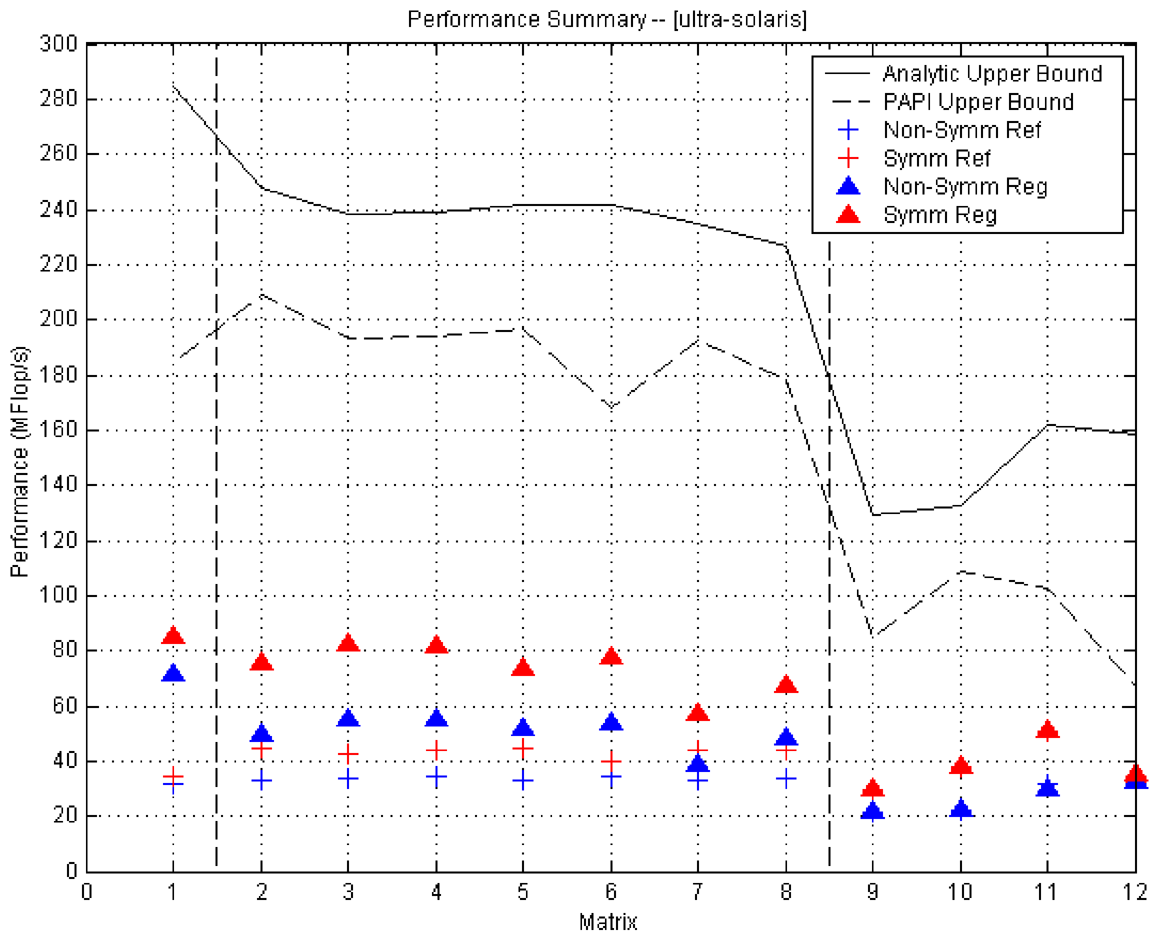
Performance Results: Sun Ultra 2i



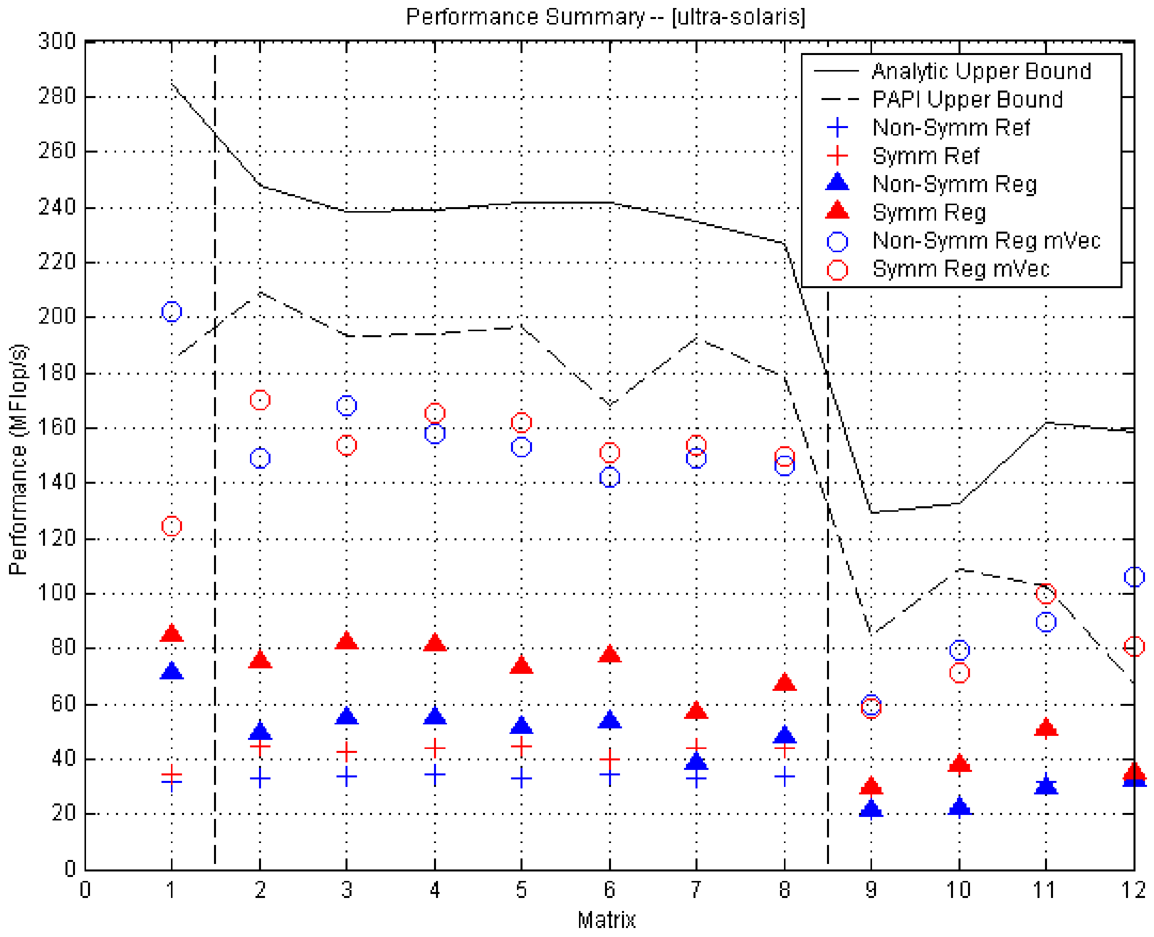
Performance Results: Sun Ultra 2i



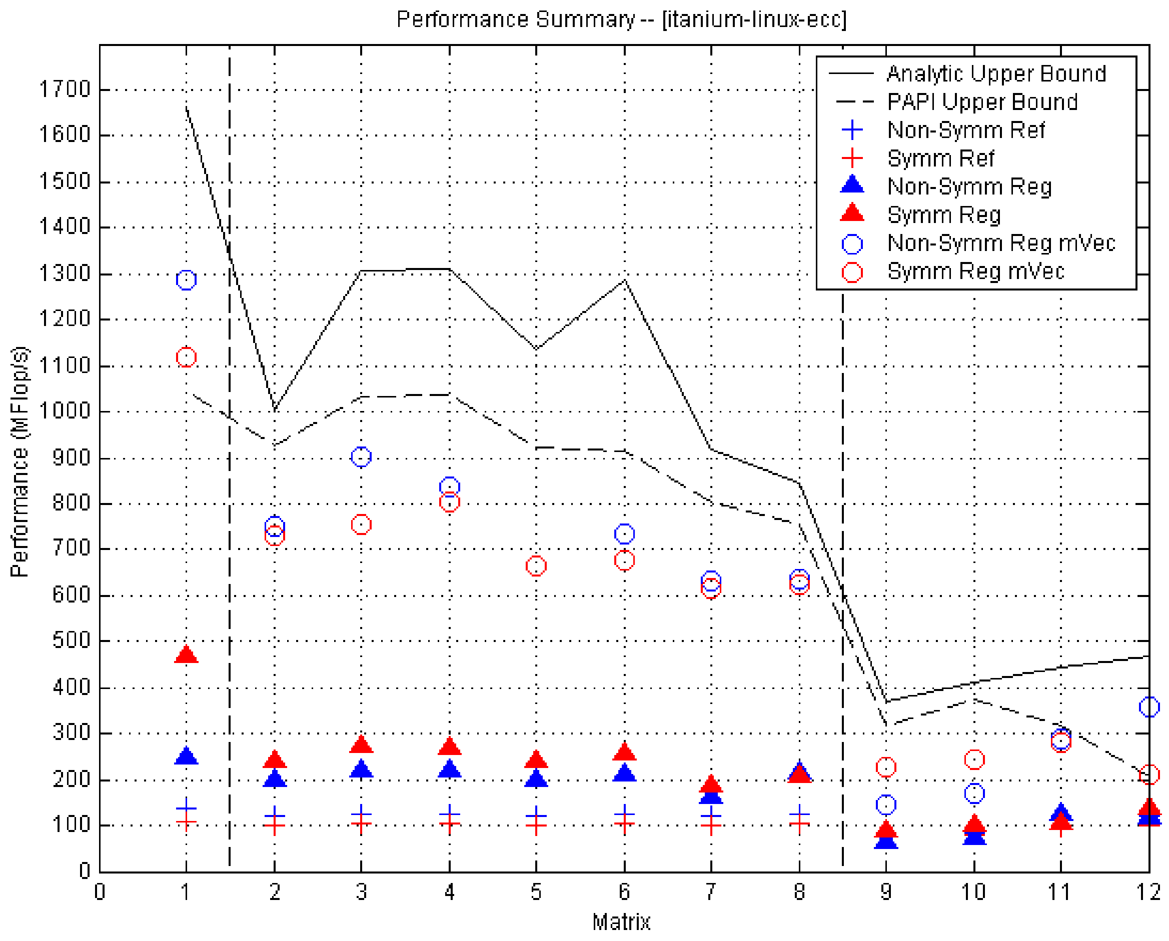
Performance Results: Sun Ultra 2i



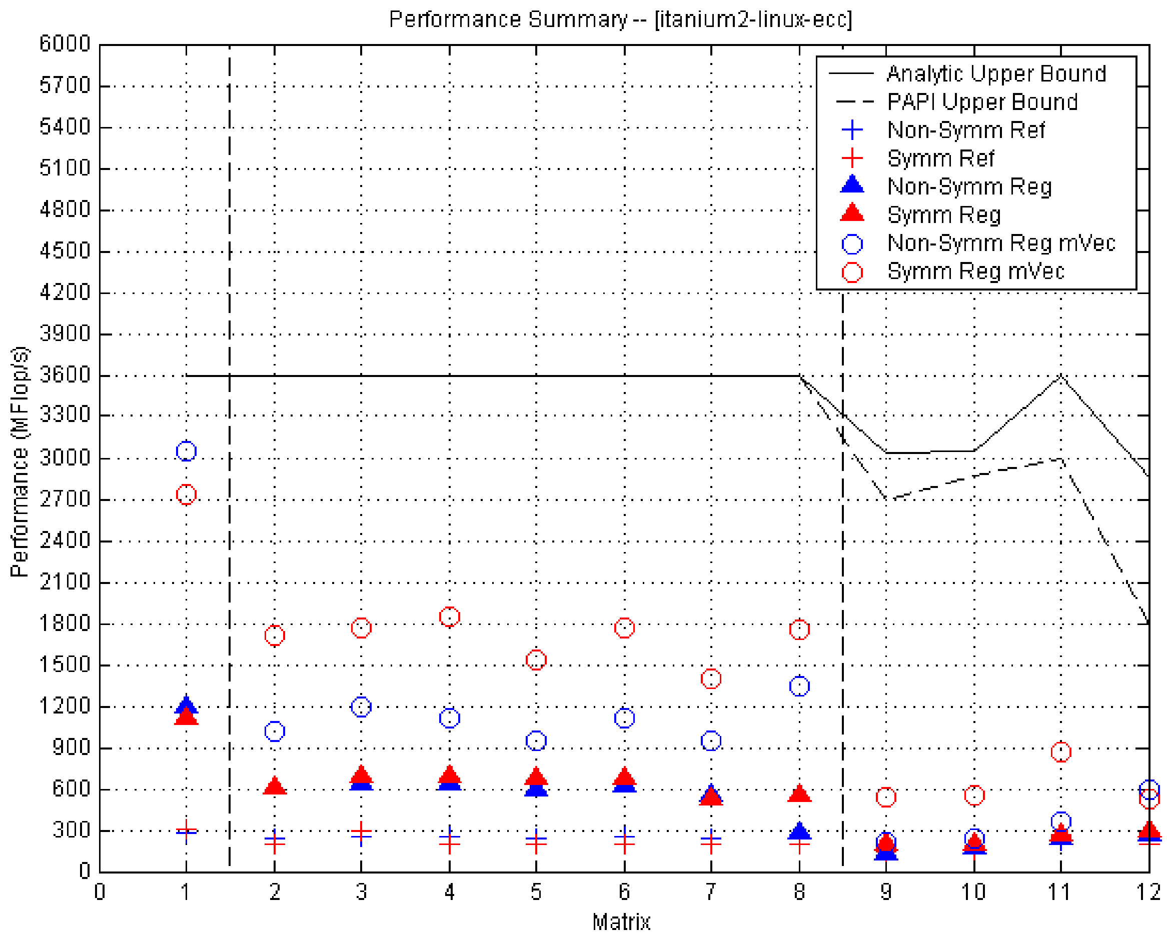
Performance Results: Sun Ultra 2i



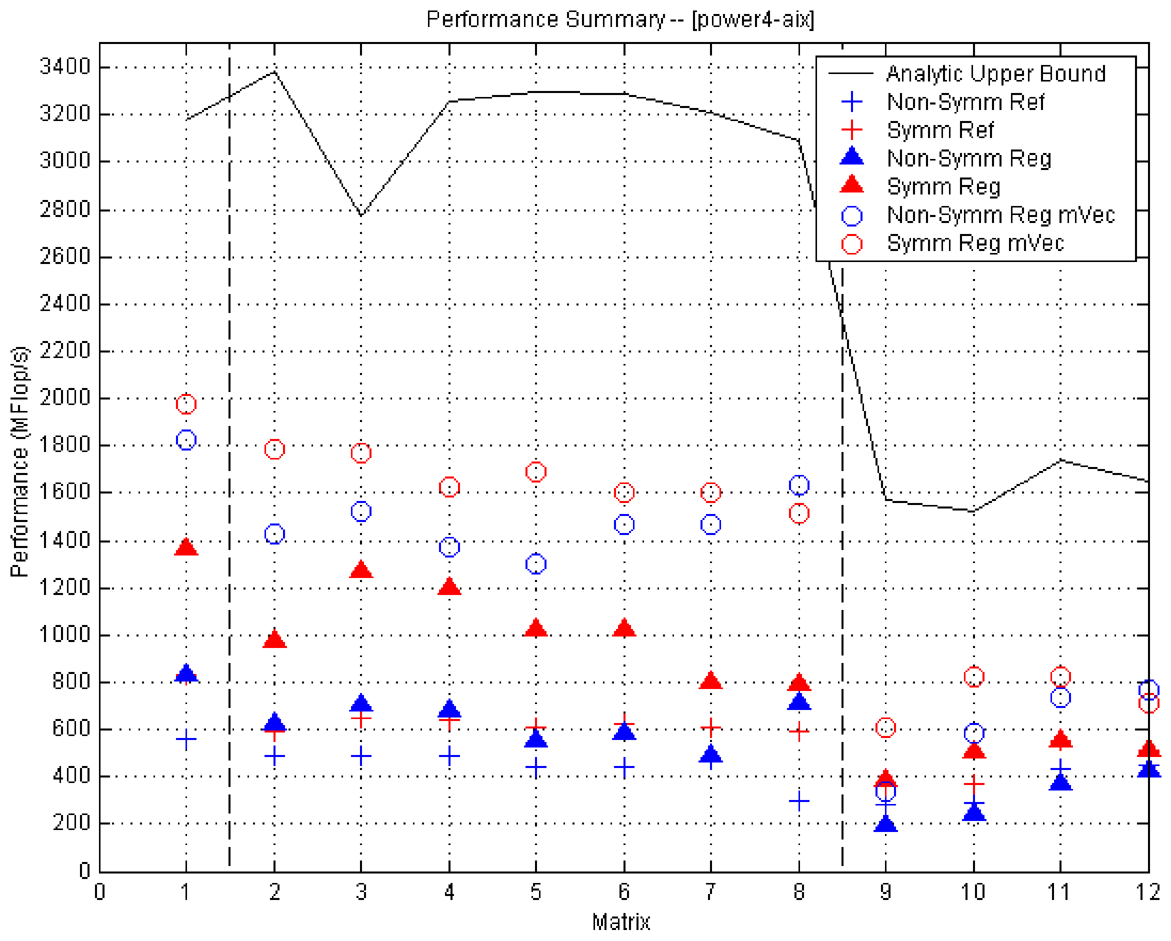
Performance Results: Intel Itanium 1



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

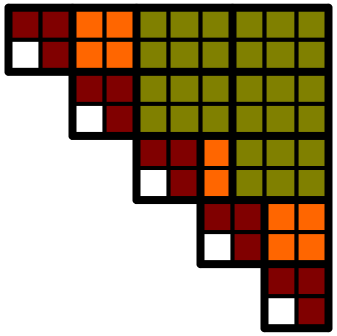
Appendices

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 $1 = i = v$
    - SR_i unrolls the multiplication of each matrix element by i
  - Dispatch algorithm, assuming vector width v
    - Invoke SR_v $floor(k/v)$ times
    - Invoke $SR_{k\%v}$ once, if $k\%v > 0$

References (1/3)

[BACD97] J. Bilmes, K. Asanovi´c, C.W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. In *Proceedings of the International Conference on Supercomputing*, Vienna, Austria, July 1997. ACM SIGARC. see http://www.icsi.berkeley.edu/.bilmes/phipac.

[BCD+01] S. Blackford, G. Corliss, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, C. Hu, W. Kahan, L. Kaufman, B. Kearfott, F. Krogh, X. Li, Z. Maany, A. Petitet, R. Pozo, K. Remington, W. Walster, C. Whaley, and J. Wolff von Gudenberg. Document for the Basic Linear Algebra Subprograms (BLAS) standard: BLAS Technical Forum, 2001. www.netlib.org/blast.

[BW99] Aart J. C. Bik and Harry A. G. Wijshoff. Automatic nonzero structure analysis. *SIAM Journal on Computing*, 28(5):1576.1587, 1999.

[FC00] Salvatore Filippone and Michele Colajanni. PSBLAS: A library for parallel linear algebra computation on sparse matrices. *ACM Transactions on Mathematical Software*, 26(4):527.550, December 2000.

[FDZ99] Basilio B. Fraguela, Ram´on Doallo, and Emilio L. Zapata. Memory hierarchy performance prediction for sparse blocked algorithms. *Parallel Processing Letters*, 9(3), March 1999.

[FJ98] Matteo Frigo and Stephen Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Seattle, Washington, May 1998.

[GKKS99] William D. Gropp, D. K. Kasushik, David E. Keyes, and Barry F. Smith. Towards realistic bounds for implicit CFD codes. In *Proceedings of Parallel Computational Fluid Dynamics*, pages 241.248, 1999.

References (2/3)

[GR99] Roman Geus and S. R¨ ollin. Towards a fast parallel sparse matrix-vector multiplication. In E. H. D'Hollander, J. R. Joubert, F. J. Peters, and H. Sips, editors, *Proceedings of the International Conference on Parallel Computing (ParCo)*, pages 308.315. Imperial College Press, 1999.

[HPDR99] Dora Blanco Heras, Vicente Blanco Perez, Jose Carlos Cabaleiro Dominguez, and Francisco F. Rivera. Modeling and improving locality for irregular problems: sparse matrix-vector product on cache memories as a case study. In *HPCN Europe*, pages 201.210, 1999.

[Im00] Eun-Jin Im. *Optimizing the performance of sparse matrix-vector multiplication*. PhD thesis, University of California, Berkeley, May 2000.

[MMJ00] Dragan Mirkovic, Rishad Mahasoom, and Lennart Johnsson. An adaptive software library for fast fourier transforms. In *Proceedings of the International Conference on Supercomputing*, pages 215.224, Sante Fe, NM, May 2000.

[NGLPJ96] J. J. Navarro, E. Garc´ia, J. L. Larriba-Pey, and T. Juan. Algorithms for sparse matrix computations on high-performance workstations. In *Proceedings of the 10th ACM International Conference on Supercomputing*, pages 301.308, Philadelpha, PA, USA, May 1996.

[PH99] Ali Pinar and Michael Heath. Improving performance of sparse matrix vector multiplication. In *Proceedings of Supercomputing*, 1999.

[PSVM01] Markus Puschel, Bryan Singer, Manuela Veloso, and Jose M. F. Moura. Fast automatic generation of DSP algorithms. In *Proceedings of the International Conference on Computational Science*, volume 2073 of *LNCS*, pages 97.106, San Francisco, CA, May 2001. Springer.

[RP96] K. Remington and R. Pozo. NIST Sparse BLAS: User's Guide. Technical report, NIST, 1996. gams.nist.gov/spblas.

References (3/3)

[Saa94] Yousef Saad. SPARSKIT: A basic toolkit for sparse matrix computations, 1994. www.cs.umn.edu/Research/arpa/SPARSKIT/sparskit.html.

[Sto97] Paul Stodghill. *A Relational Approach to the Automatic Generation of Sequential Sparse Matrix Codes*. PhD thesis, Cornell University, August 1997.

[Tol97] Sivan Toledo. Improving memory-system performance of sparse matrix vector multiplication. In *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.

[VFD01] Sathish S. Vadhiyar, Graham E. Fagg, and Jack J. Dongarra. Towards an accurate model for collective communications. In *Proceedings of the International Conference on Computational Science*, volume 2073 of *LNCS*, pages 41.50, San Francisco, CA, May 2001. Springer.

[WPD01] R. Clint Whaley, Antoine Petitet, and Jack Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1):3.25, 2001.

[WS97] James B. White and P. Sadayappan. On improving the performance of sparse matrix-vector multiplication. In *Proceedings of the International Conference on High-Performance Computing*, 1997.