Memory Hierarchy Optimizations and Performance Bounds for Sparse $A^T Ax$

Richard Vuduc, Attila Gyulassy, James Demmel, Katherine Yelick

Monday, June 2, 2003

Berkeley Benchmarking and OPtimization (BeBOP) Project

bebop.cs.berkeley.edu

Department of Electrical Engineering and Computer Science

U.C. Berkeley, California, USA

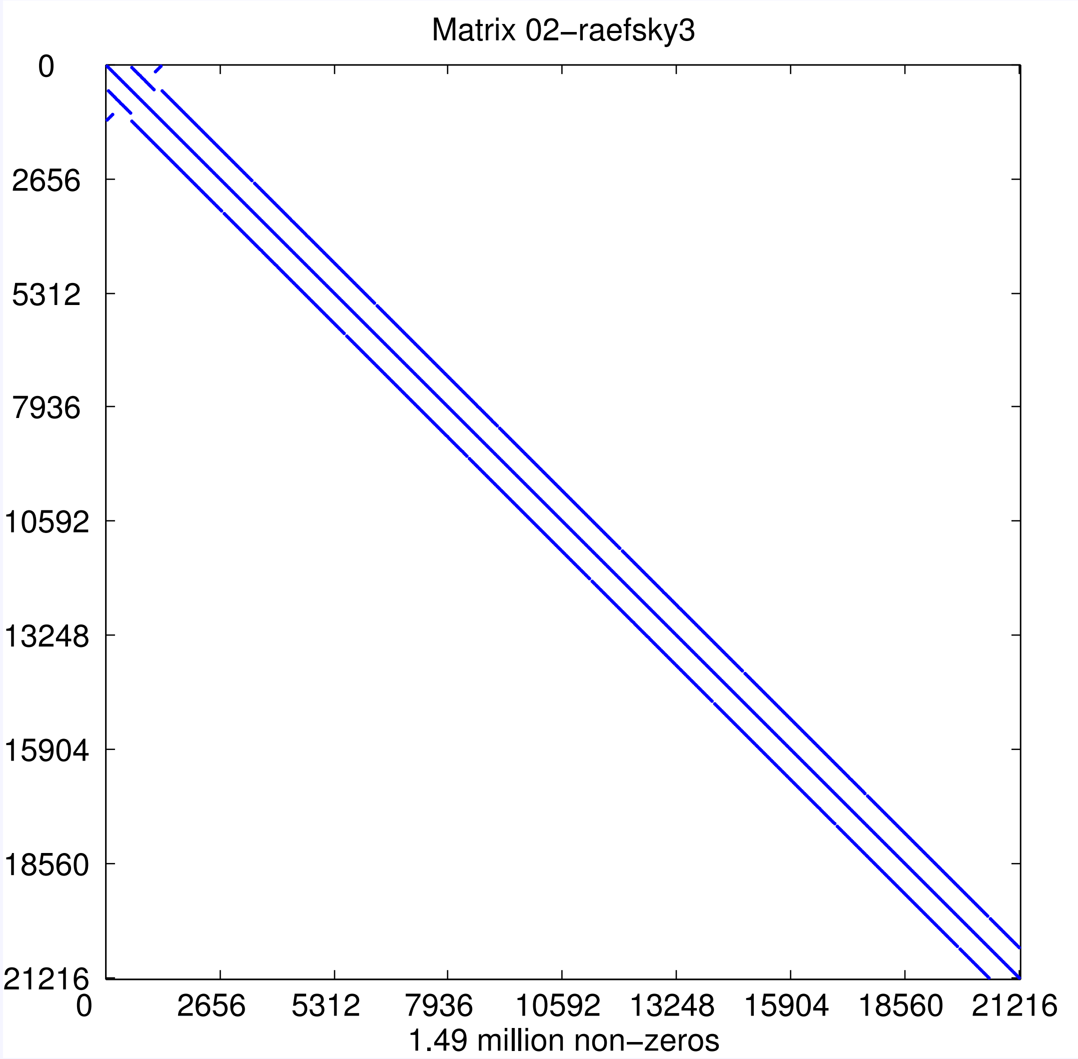
Workshop on Parallel Linear Algebra (at ICCS-2003)

[<--->]

Context: Automatic Tuning of Sparse Kernels

- Road-blocks to writing efficient sparse code
  - high bandwidth requirements (extra storage)
  - poor locality (indirect, irregular memory access)
  - poor instruction mix (data structure manipulation)
  - typical sparse matrix-vector multiply (SpM×V) performance: less than 10% of machine peak
  - **performance depends on kernel, architecture, and matrix**
- Goal: automatically choose “best” data structure and implementation (code), given matrix and machine
  - Inspiration: ATLAS/PHiPAC, FFTW/SPRIAL/UHFFT, SPARSITY . . .
- This talk: $y \leftarrow y + A^T Ax$, or Sp$A^T A$
  - Iterative interior point methods [WO95], SVD [Dem97], HITS algorithm [Kle99]
  - Other kernels: SpM×V [SC’02], trisolve [ICS/POHLL’02], $A^k x$, RAR^T, . . .

Motivating Example: Matrix `raefsky3`



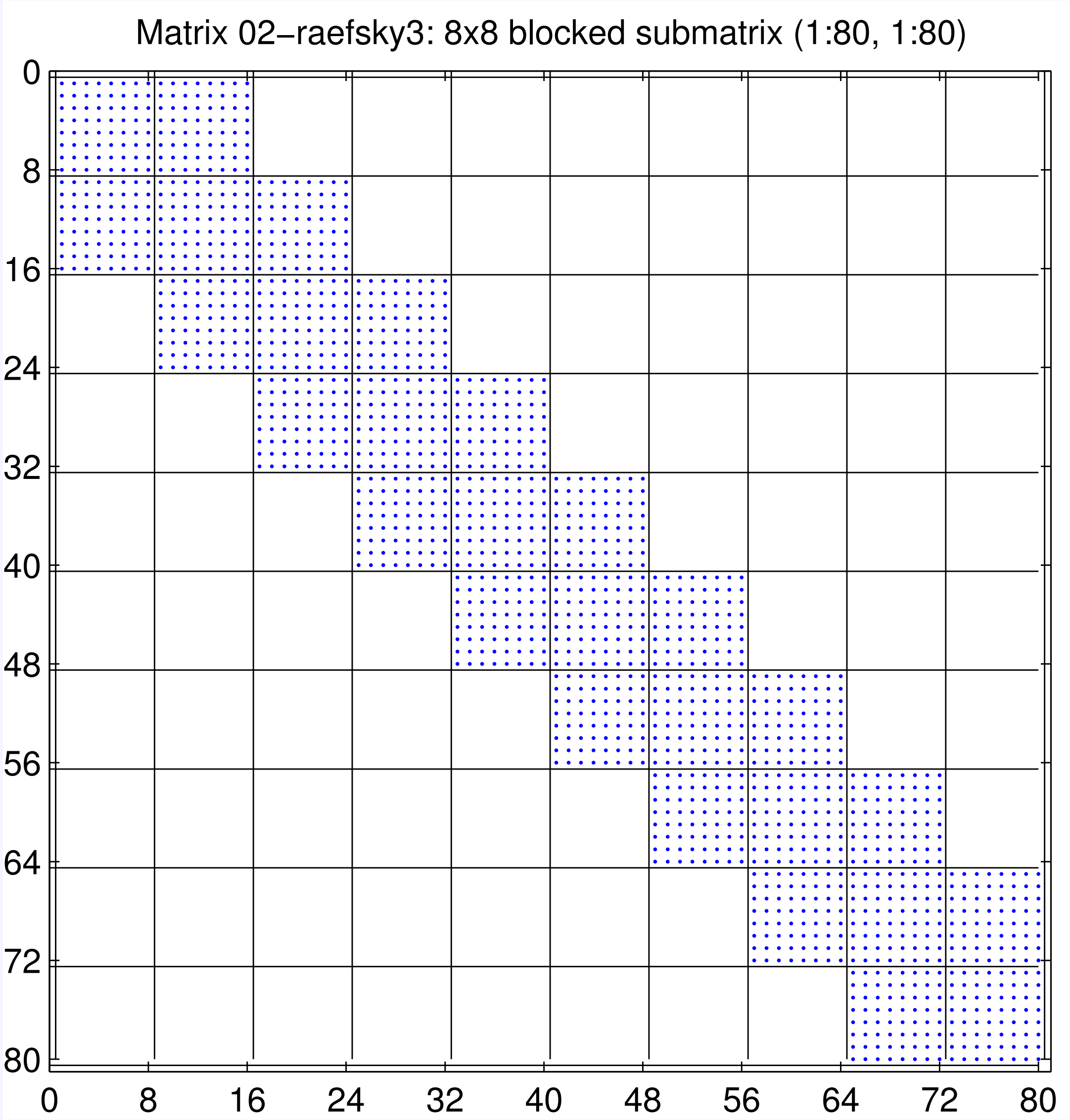
N = 21200

nnz = 1.5M

Kernel = Sp$A^T A$

[<––>]

Motivating Example: Matrix `raefsky3`



N = 21200

nnz = 1.5M

Kernel = Sp$A^T A$

A natural choice:

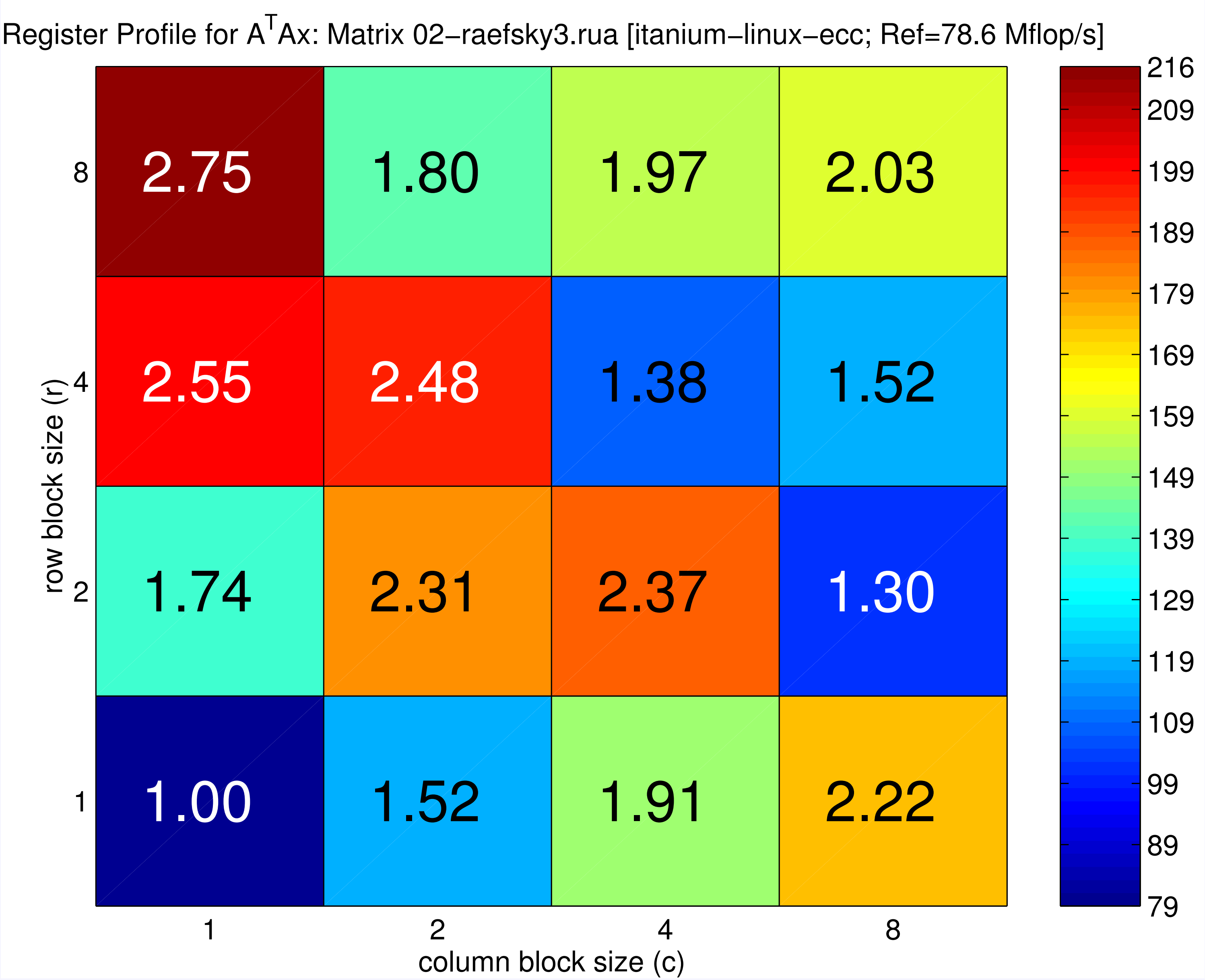
$8{\times}8$ blocked compressed sparse row (BCSR).

Experiment:

Measure performance of all $r{\times}c$ block sizes for $r, c \in \{1, 2, 4, 8\}$.

[<––>]

Speedups on Itanium: The Need for Search



(Peak machine speed: 3.2 Gflop/s)

[<−−>]

Key Questions and Conclusions

- How can we exploit the memory hierarchy for SpA^TA?
  - Interleave multiplication by A, A^T (up to 1.6x speedup)
  - Combine with prior SPARSITY register blocking optimization (4.2x)
- How do we choose the best data structure automatically?
  - Matrix may be unknown until run-time
  - Heuristic for choosing optimal (or near-optimal) block sizes
- What are the limits on performance of blocked SpA^TA?
  - Derive performance upper bounds for blocking
  - For SpM×V, within 20% of bound $\Rightarrow$ tuning limited [SC'02]
  - For SpA^TA, within 40% of bound $\Rightarrow$ tuning opportunities a la ATLAS

[<—>]

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])
  - Generic programming (Blitz++ [Vel98], MTL [SL98], GMCL [Neu98], . . .)
  - FLAME [GGHvdG01]
- Sparse performance modeling and tuning
  - Temam and Jalby [TJ92]
  - Toledo [Tol97], White and Sadayappan [JBWS97], 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, hypre, . . .

[<-->]

Memory Hierarchy Optimizations for SpA^TA

Cache-level: Interleave multiplication of x by A, A^T:

$$\begin{aligned} y = A^TAx &= (a_1 \ldots a_m) \begin{pmatrix} a_1^T \\ \ldots \\ a_m^T \end{pmatrix} x \\ &= \sum_{i=1}^{m} a_i (a_i^T x). \qquad (1) \end{aligned}$$

Dot-product, followed by “axpy”: reuse a_i^T.

[<—>]

Memory Hierarchy Optimizations for SpA^TA

Cache-level: Interleave multiplication of x by A, A^T:

$$\begin{aligned} y = A^TAx &= (a_1 \ldots a_m)\begin{pmatrix} a_1^T \\ \ldots \\ a_m^T \end{pmatrix} x \\ &= \sum_{i=1}^{m} a_i(a_i^T x). \end{aligned} \qquad (1)$$

Dot-product, followed by “axpy”: reuse a_i^T.

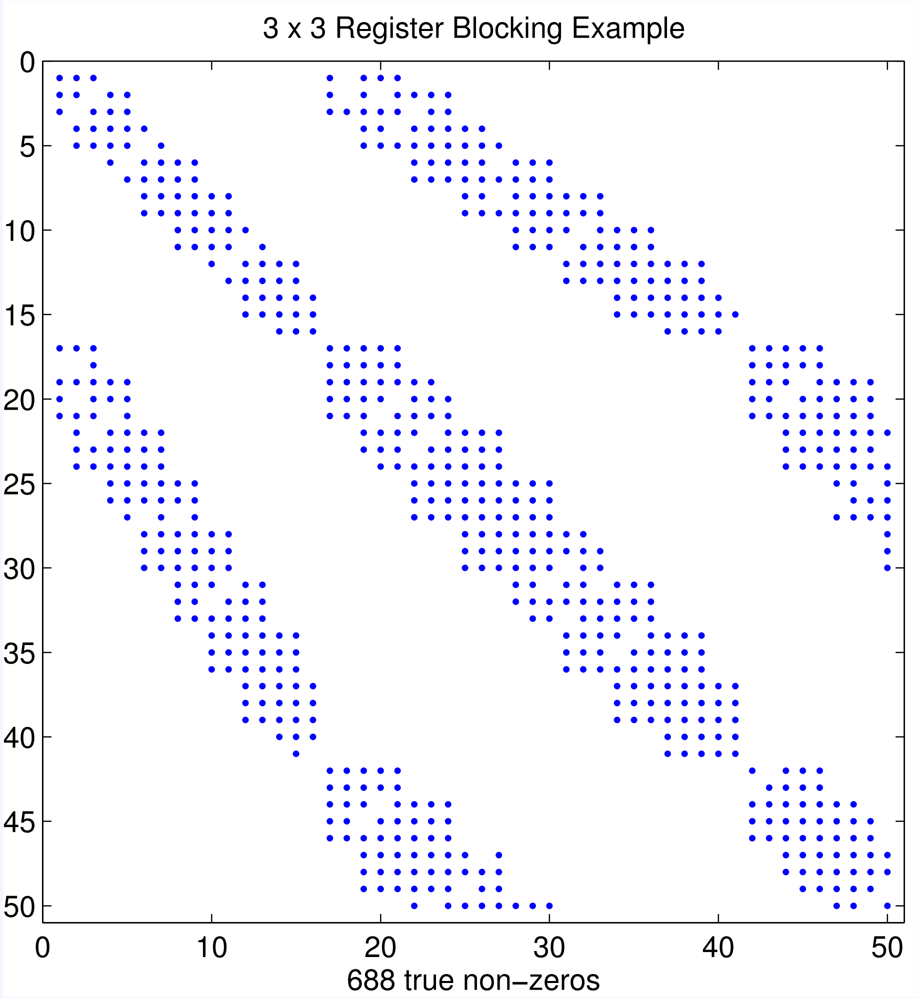
Register-level: Take a_i^T to be a block row composed of $r \times c$ blocks.

Question: How to choose r, c?

2x2 Code Example

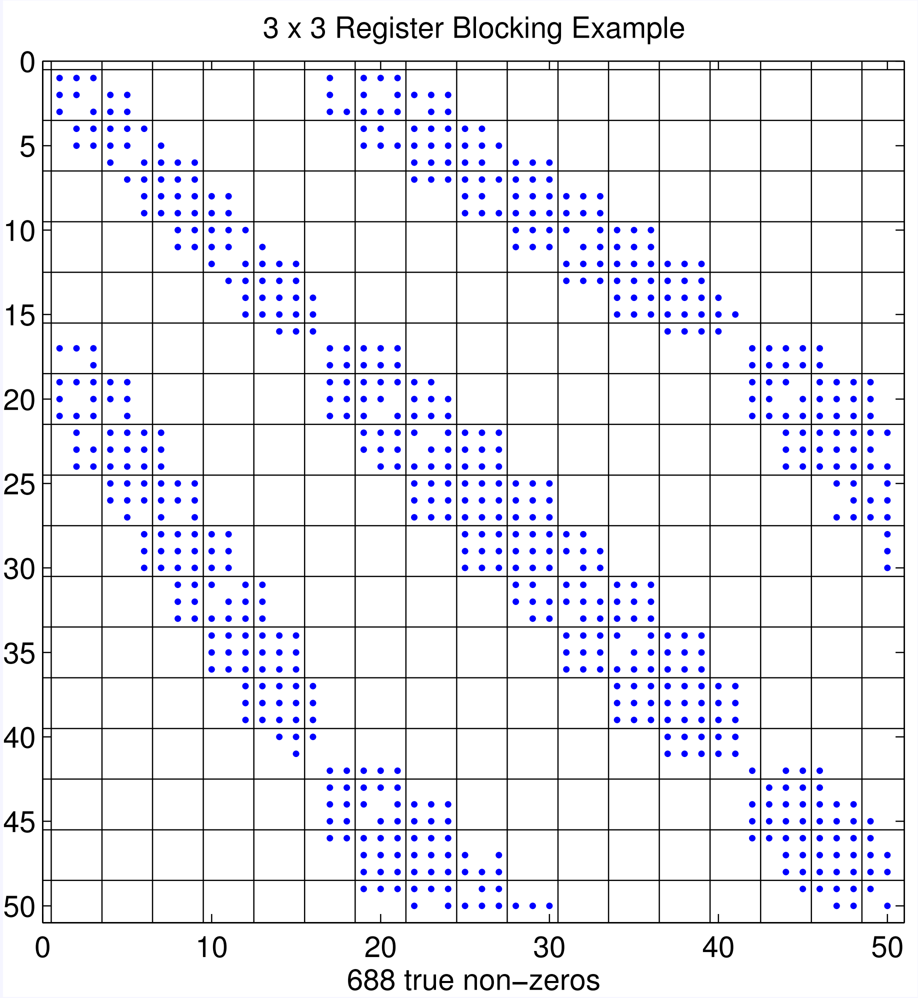
```
for( i = 0 ; i < mb ; i++, t += 2 ) {/* for each block row $A_i$ */
    register double t0 = 0, t1 = 0;
    for( j = ptr[i] ; j < ptr[i+1] ; val += 2*2 ) {/* $t \leftarrow A_i x$ */
        register double x0 = x[ind[0]+0] , x1 = x[ind[0]+1] , ;
        t0 += val[0*2+0] *x0 ;
        t1 += val[1*2+0] *x0 ;
        t0 += val[0*2+1] *x1 ;
        t1 += val[1*2+1] *x1 ;
    }
    RESET( ind, val );
    for( j = ptr[i] ; j < ptr[i+1] ; val += 2*2 ) {/* $t \leftarrow A_i^T x$ */
        register double y0 = 0, y1 = 0 ;
        y0 += val[0*2+0] *t0 ;
        y1 += val[0*2+1] *t0 ;
        y0 += val[1*2+0] *t1 ;
        y1 += val[1*2+1] *t1 ;
        y[ind[0]+0] += y0 ; y[ind[0]+1] += y1 ;
    }
}
```

Register-Level Blocking (SPARSITY): 3x3 Example



[<−−>]

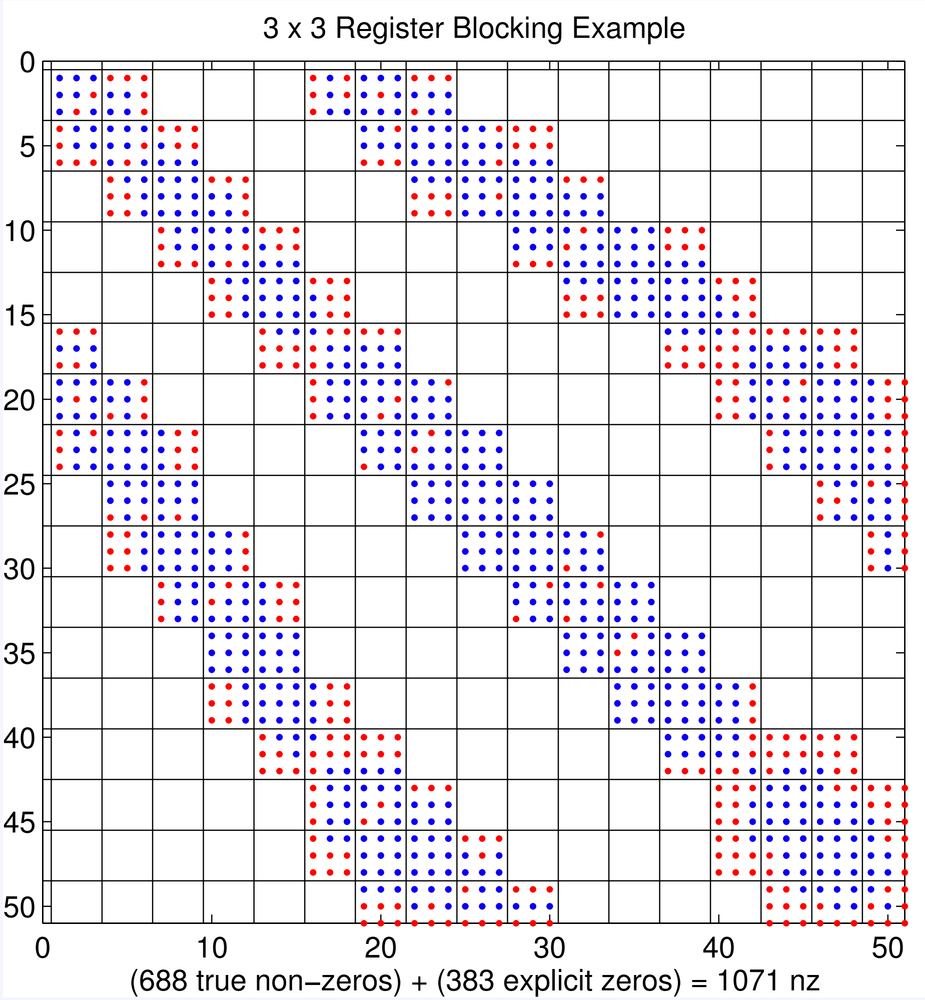
Register-Level Blocking (SPARSITY): 3x3 Example



- BCSR with uniform, aligned grid

[<−−>]

Register-Level Blocking (Sparsity): 3x3 Example

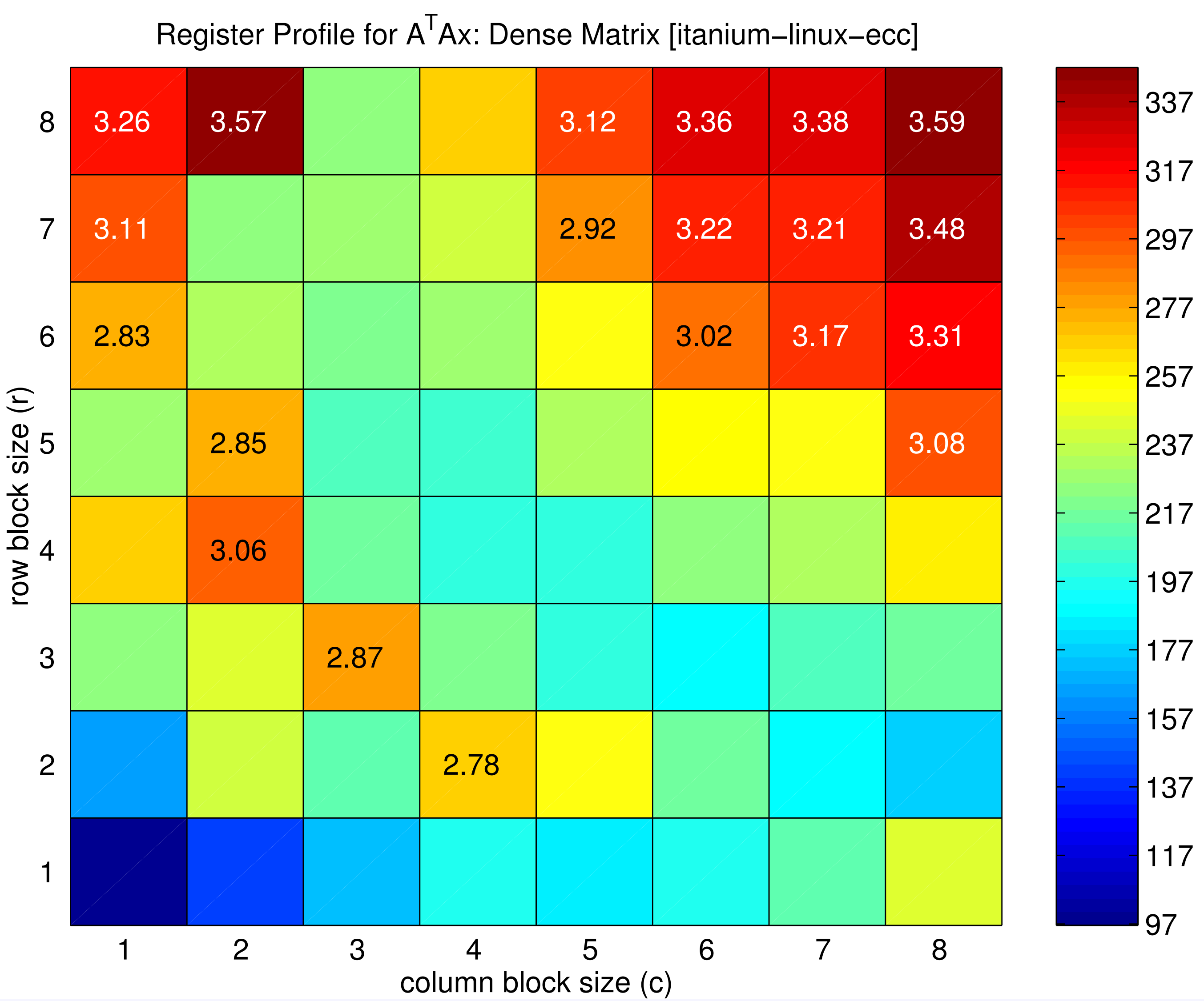


- Fill-in zeros: trade-off extra flops for better efficiency
  - This example: 50% fill led to 1.5x speedup on SpM×V on Pentium III

Approach to Automatic Tuning: Choosing r, c

- For each kernel (*e.g.*, Sp$A^T A$):
  - *Identify* and *generate* a space of implementations
    - $r \times c$ blocked, interleaved code up to 8×8
  - *Search* to find the fastest using models, experiments
    - Off-line benchmarking (once per architecture): Measure blocked interleaved Sp$A^T A$ Mflop/s on dense matrix in sparse format
    - Run-time: Estimate fill for all $r \times c$
- Inspiration: SPARSITY for Sp$M \times$V [Im & Yelick '99]
  - See also: SC'02 paper

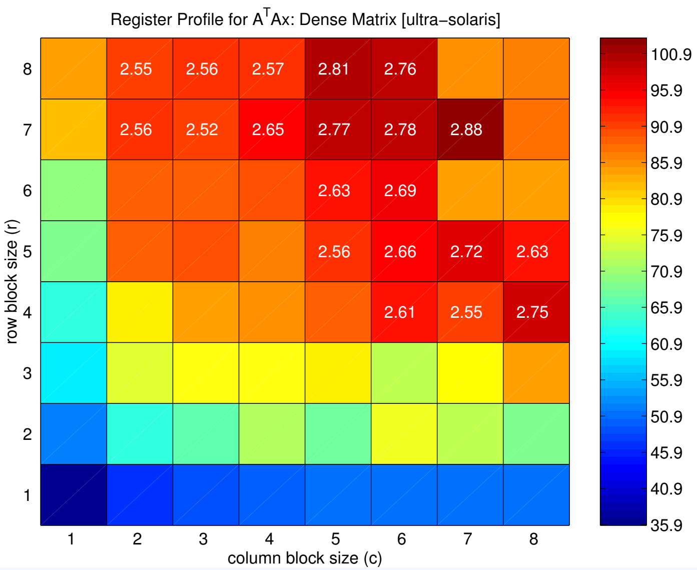
Off-line Benchmarking [Intel Itanium 1]



Top 10 codes labeled by speedup over unblocked, interleaved code. Max speedup = 3.59.

[<—>]

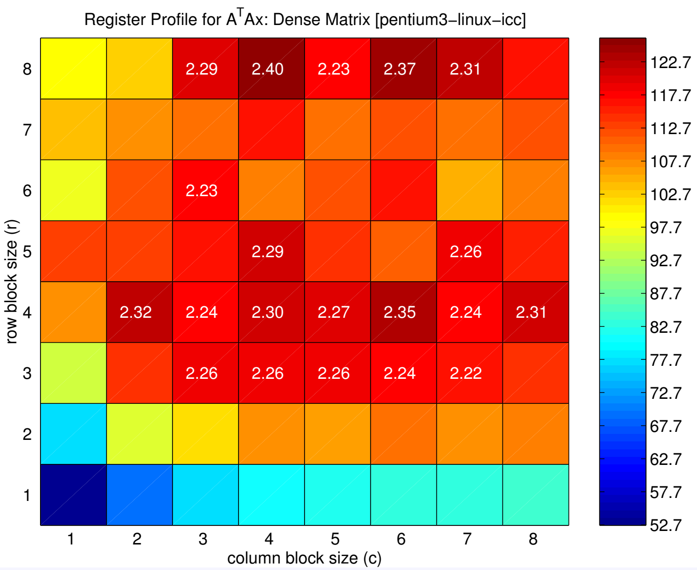
333 MHz Sun Ultra 2i (2.88)



100

35

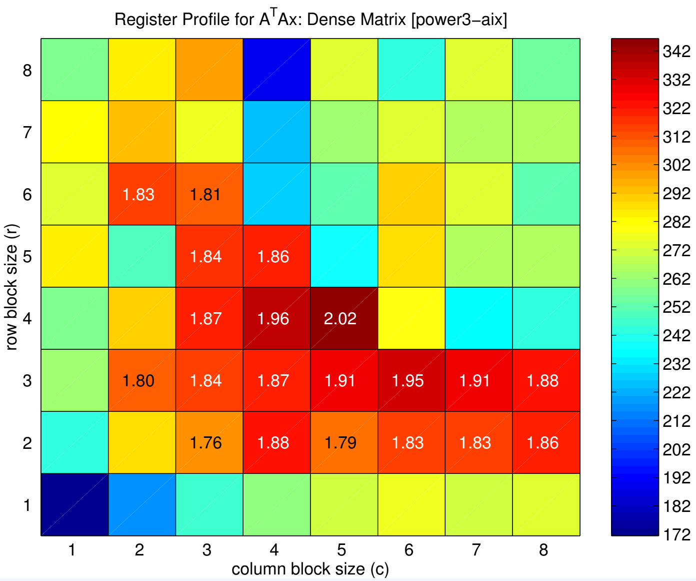
500 MHz Intel Pentium III (2.40)



123

52

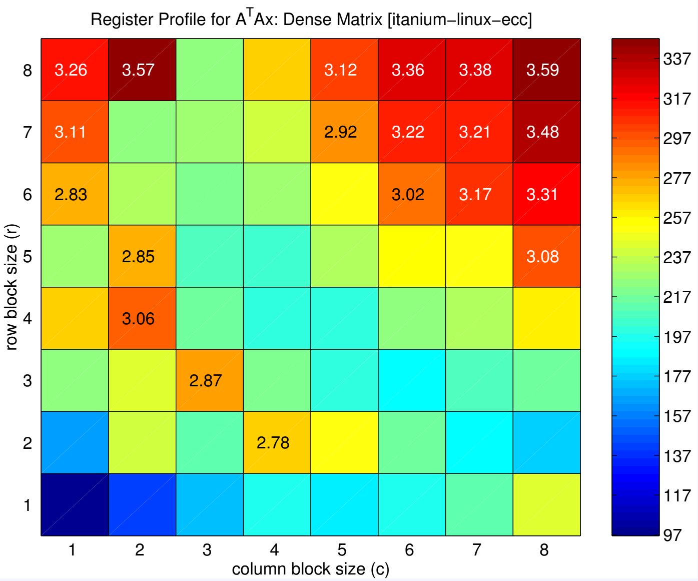
375 MHz IBM Power3 (2.02)



342

172

800 MHz Intel Itanium (3.59)



337

97

[<––>]

Performance Bounds for Block Interleaved Sp$A^T A$

- How close are we to the speed limit of blocking?
- *Upper-bounds* on performance: (flops) / (time)
  - Flops = 4 * (number of true non-zeros)
  - Lower-bound on time: two key assumptions
    - Consider only *memory* operations
    - Count only compulsory, capacity misses (*i.e.*, ignore conflicts)
  - Bound is a function of
    - Cache capacity
    - Cache line size
    - Access latencies at each level
    - Matrix and r, c (through fill ratio)
- For details and justification, see paper and [SC'02]

[<—>]

Cache Miss Model: Parameters

- Model parameters:
  - A is $m \times n$ with k non-zeros, stored in $r \times c$ BCSR format
  - L_i cache, where $1 \leq i \leq \kappa$
    - size C_i
    - line size l_i
    - access latency α_i
  - Memory access latency α_{mem}
  - One double == γ integers
- Working set: volume of data needed for each block row
  - $\hat{W}$ = matrix data working set per block row
  - $\hat{V}$ = vector data working set per block row

Cache Miss Model: Lower Bound

- Number of compulsory misses: $\frac{1}{l_i}\left(\frac{m}{r}\hat{W} + 2n\right)$
- Lower bound on L_i misses: two cases
  - 1. Entire working set fits in cache: $\hat{W} + \hat{V} \leq C_i$

$$M_i \geq \frac{1}{l_i}\left[\frac{m}{r}\hat{W} + 2n\right]$$

  - 2. Working set exceeds capacity: $\hat{W} + \hat{V} > C_i$

$$M_i \geq \frac{1}{l_i}\left[\frac{m}{r}\hat{W} + 2n + \frac{m}{r}\left(\hat{W} + \hat{V} - C_i\right)\right]$$

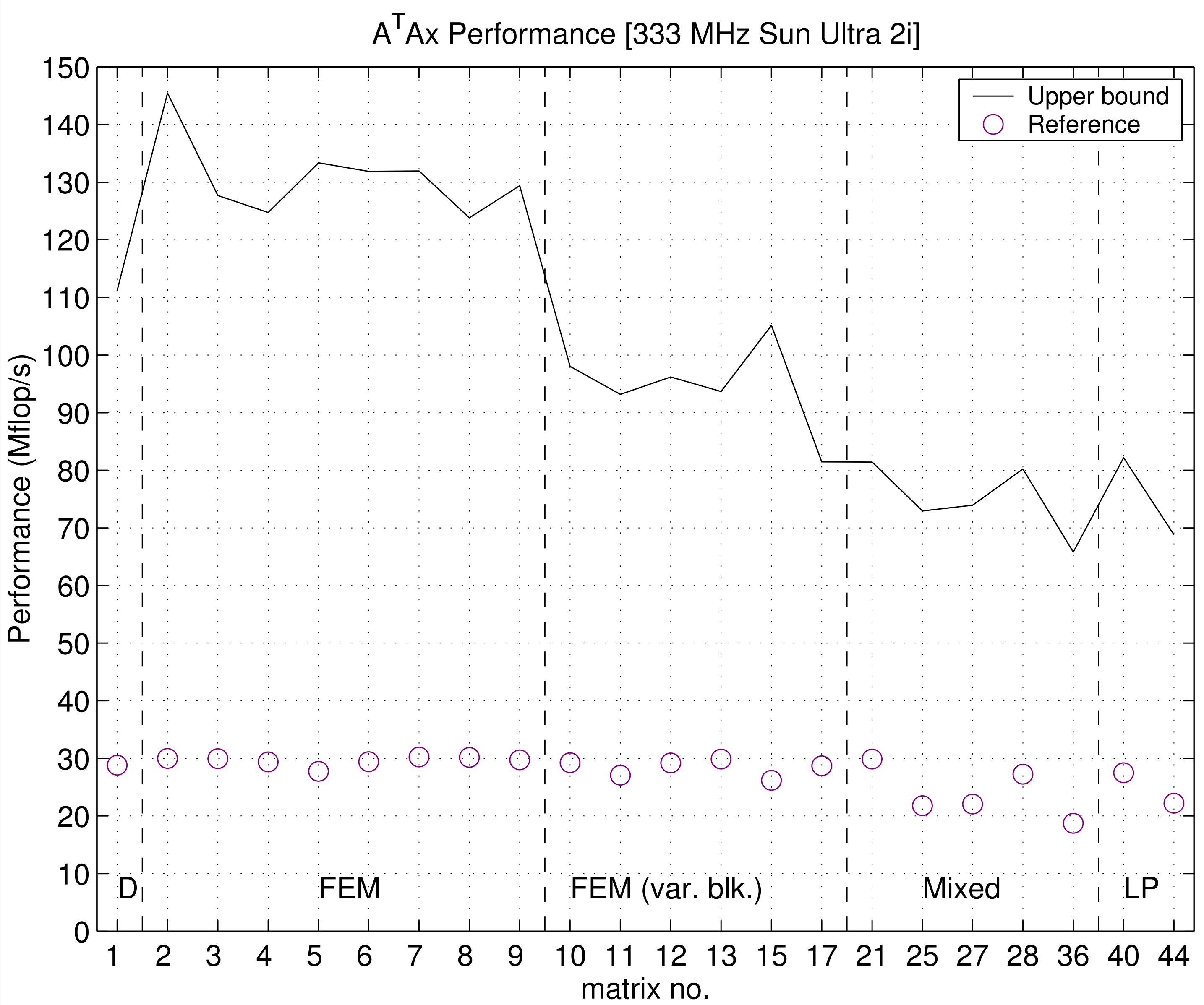
- Assumes full associativity and complete “user” control of data placement in cache—see paper

[<−−>]

Overview of Performance Results

- Experimental setup
  - Four machines: Pentium III, Ultra 2i, Power3, Itanium
  - 44 matrices: dense, finite element, mixed, linear programming
  - Reference: CSR (*i.e.*, $1{\times}1$ or “unblocked”), no interleaving
  - Measured misses and cycles using PAPI [BDG$^+$00]
- Main observations
  - Interleaving (cache optimization): up to 1.6x speedup
  - Reg + cache: up to 4.2x speedup, 1.8x over blocked non-interleaved
  - Heuristic choice within 10% of best performance
  - Performance is 20–40% of bound: more tuning possible

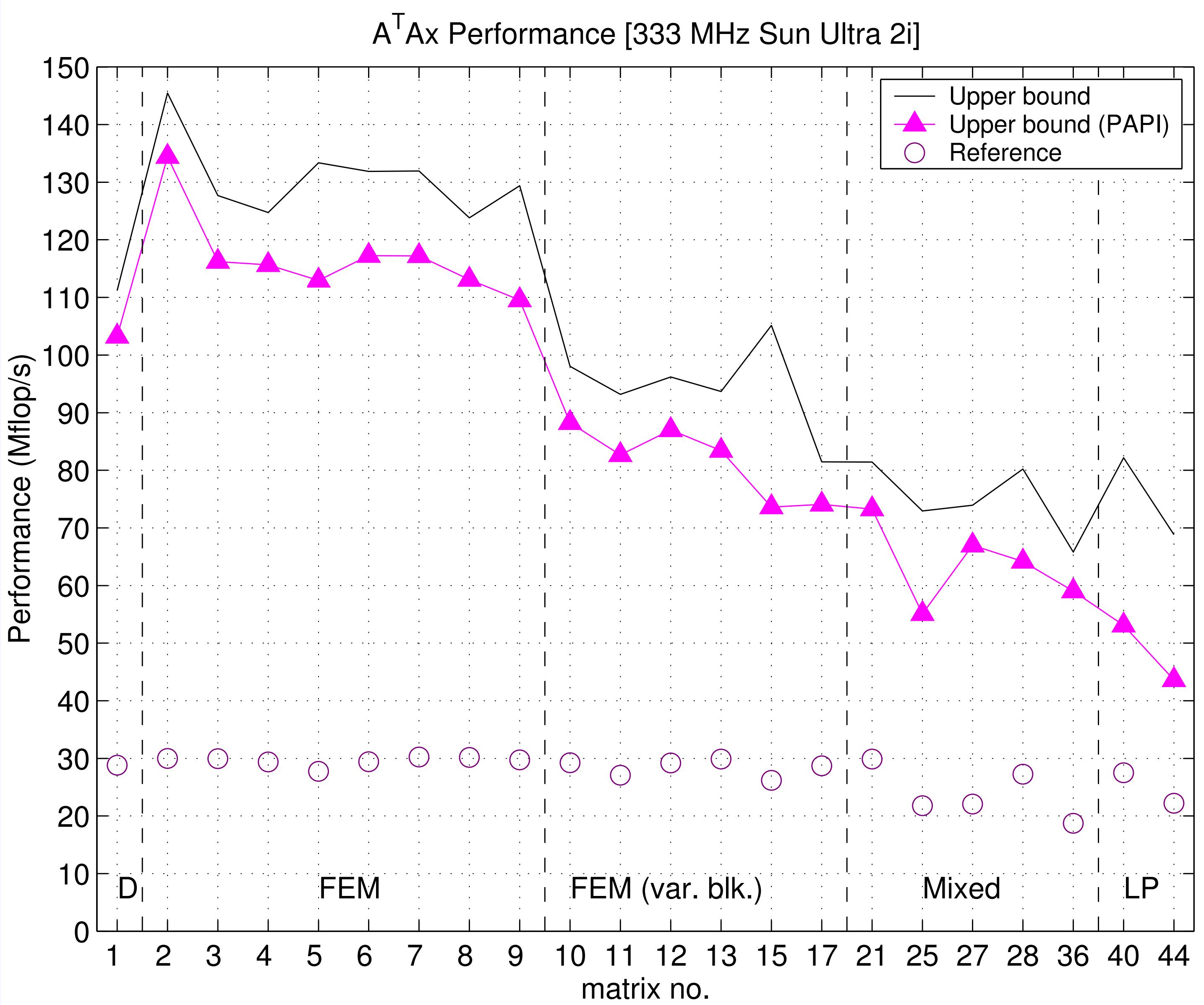
Performance Results: Sun Ultra 2i



- DGEMV (n=1000): 58 Mflop/s

[<-->]

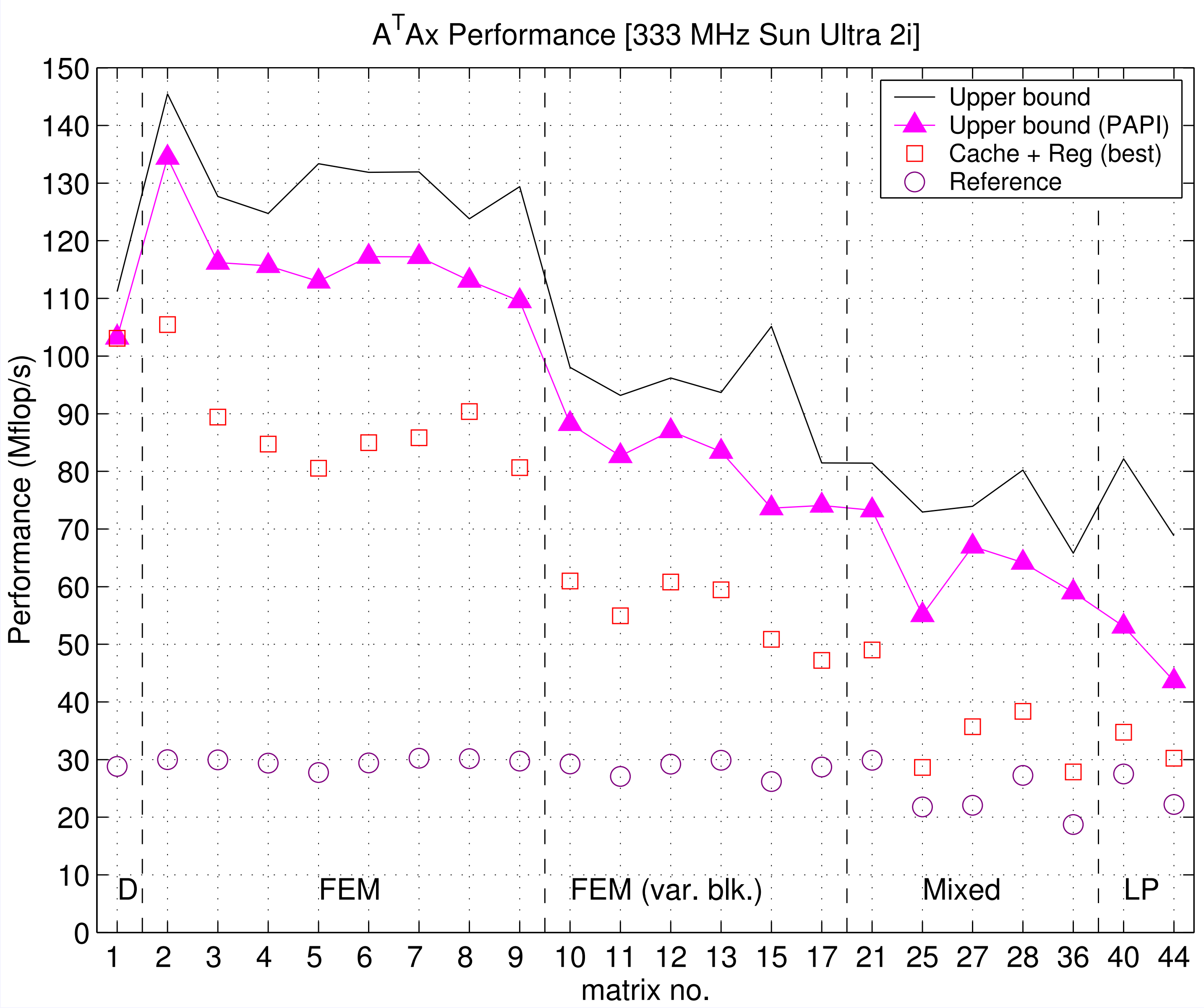
Performance Results: Sun Ultra 2i



- DGEMV (n=1000): 58 Mflop/s

[<—>]

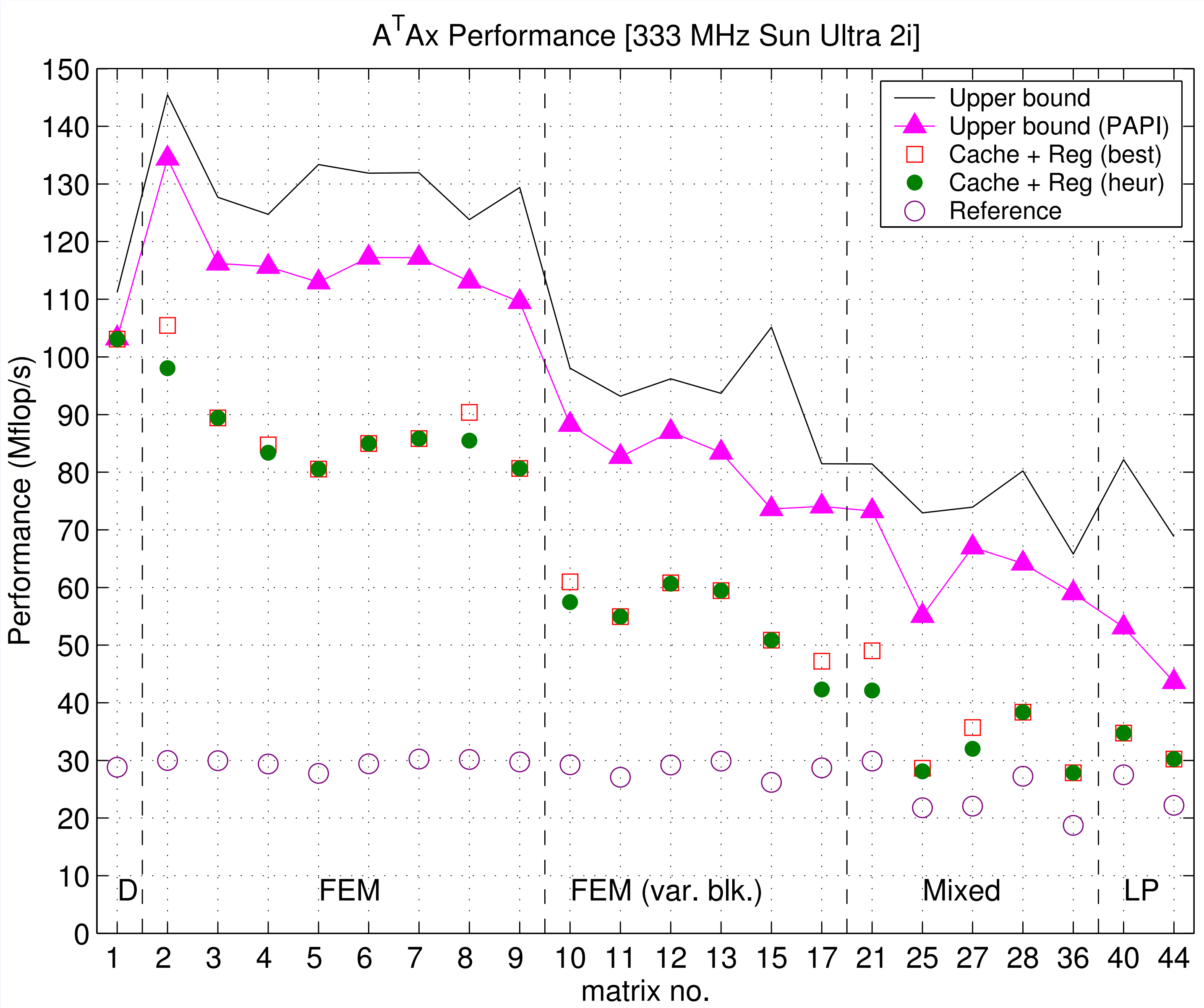
Performance Results: Sun Ultra 2i



- DGEMV (n=1000): 58 Mflop/s

[<--->]

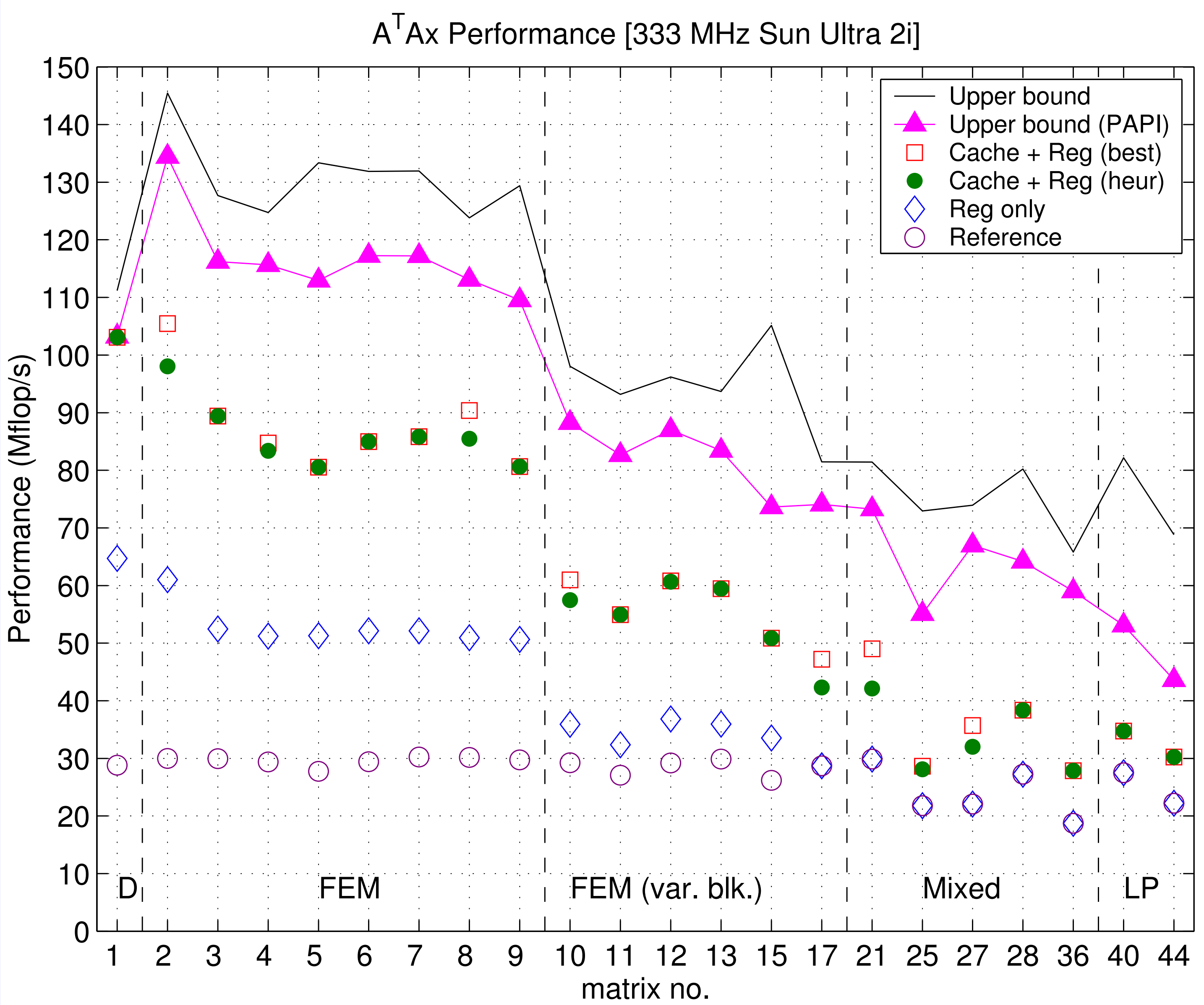
Performance Results: Sun Ultra 2i



- DGEMV (n=1000): 58 Mflop/s

[<—>]

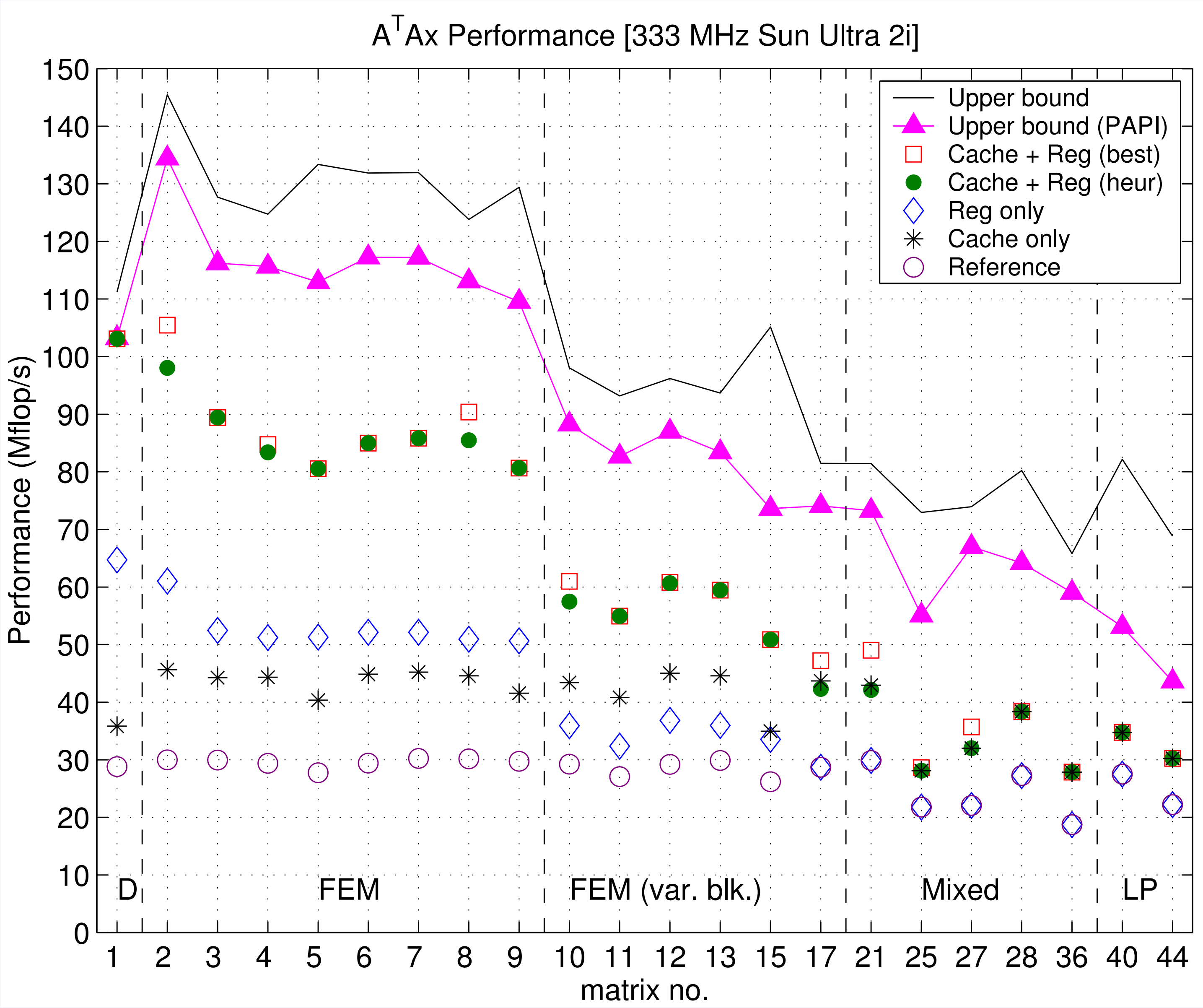
Performance Results: Sun Ultra 2i



- DGEMV (n=1000): 58 Mflop/s

[<--->]

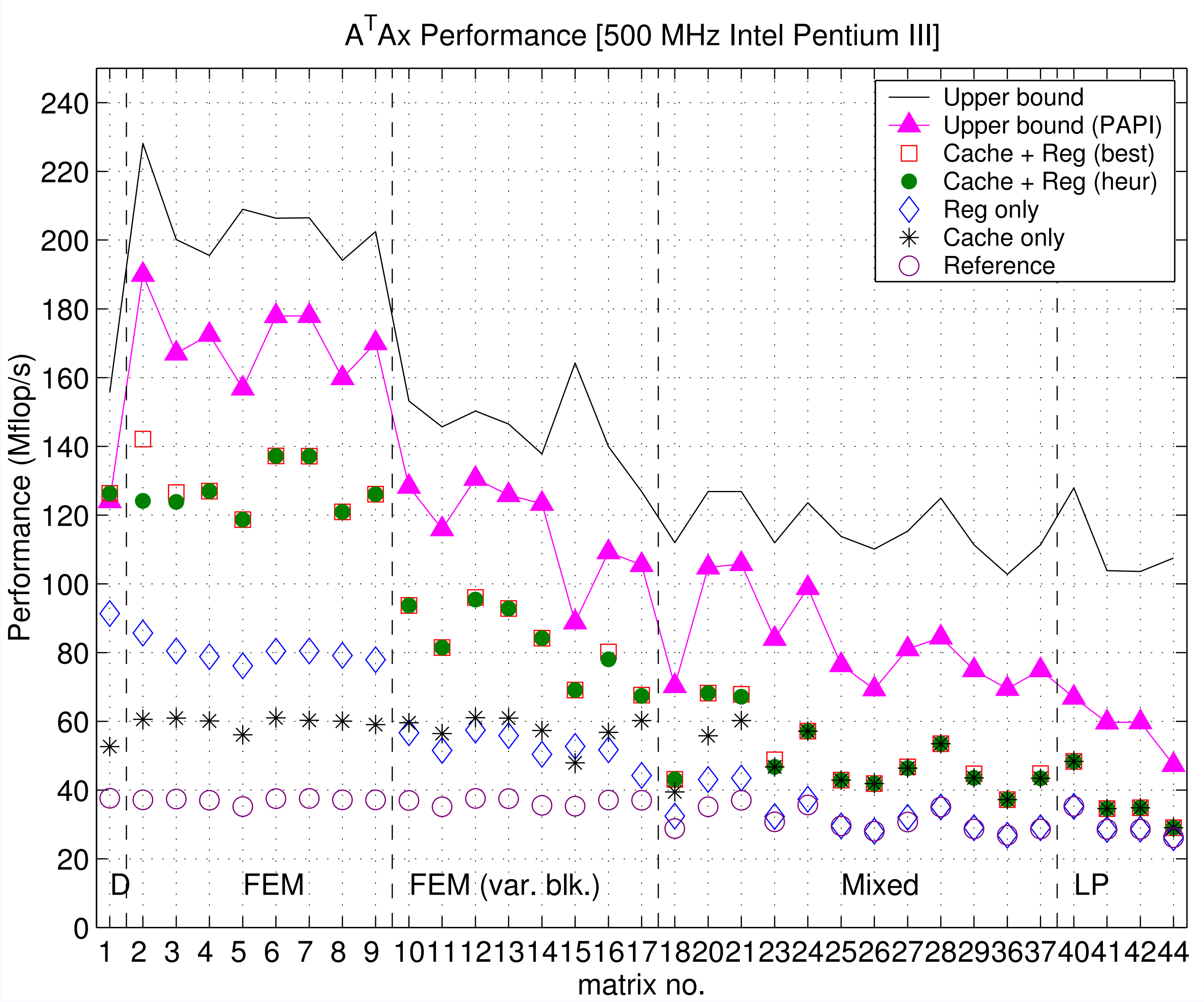
Performance Results: Sun Ultra 2i



- DGEMV (n=1000): 58 Mflop/s

[<-->]

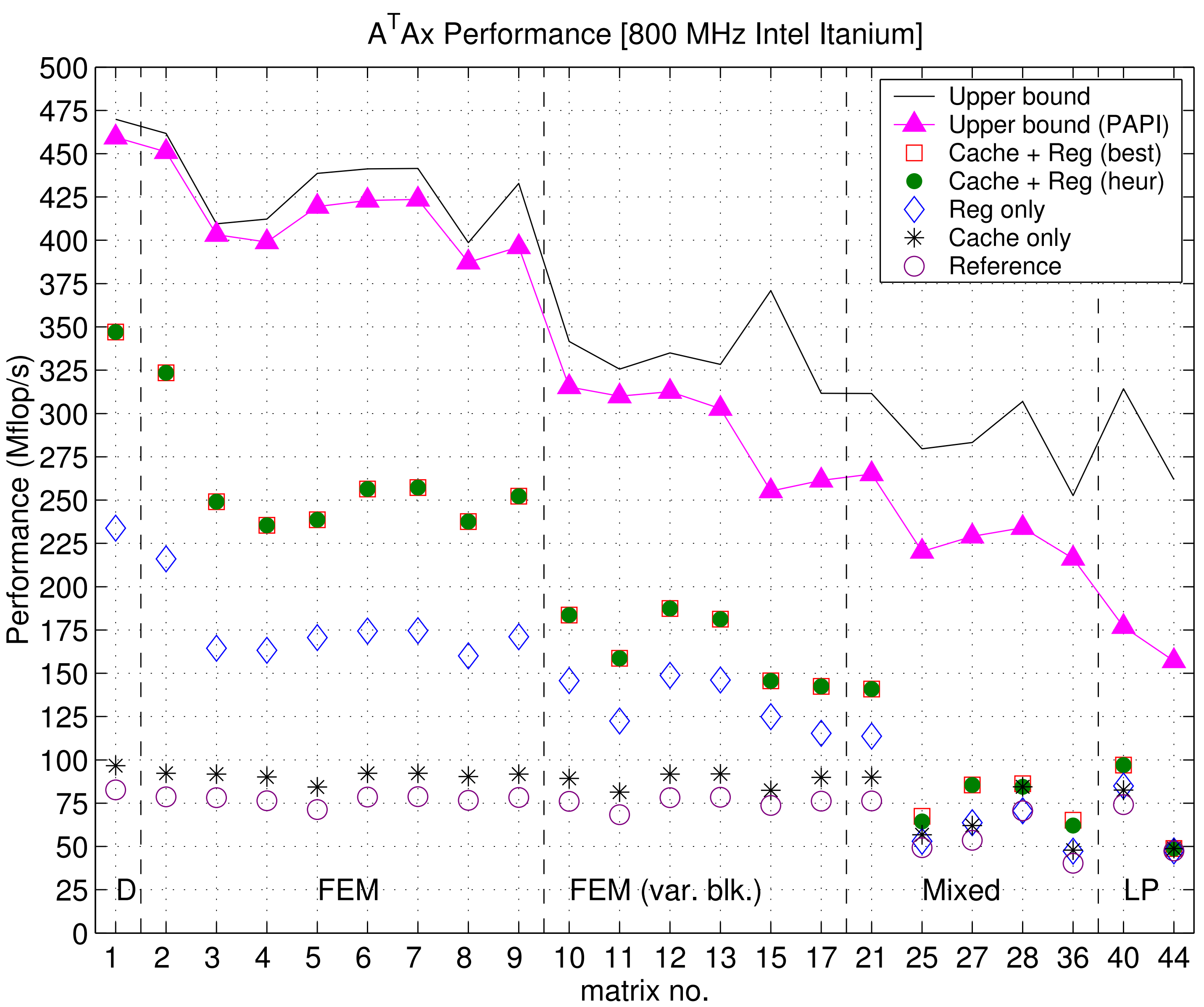
Performance Results: Intel Pentium III



- DGEMV (n=1000): 96 Mflop/s

[<--->]

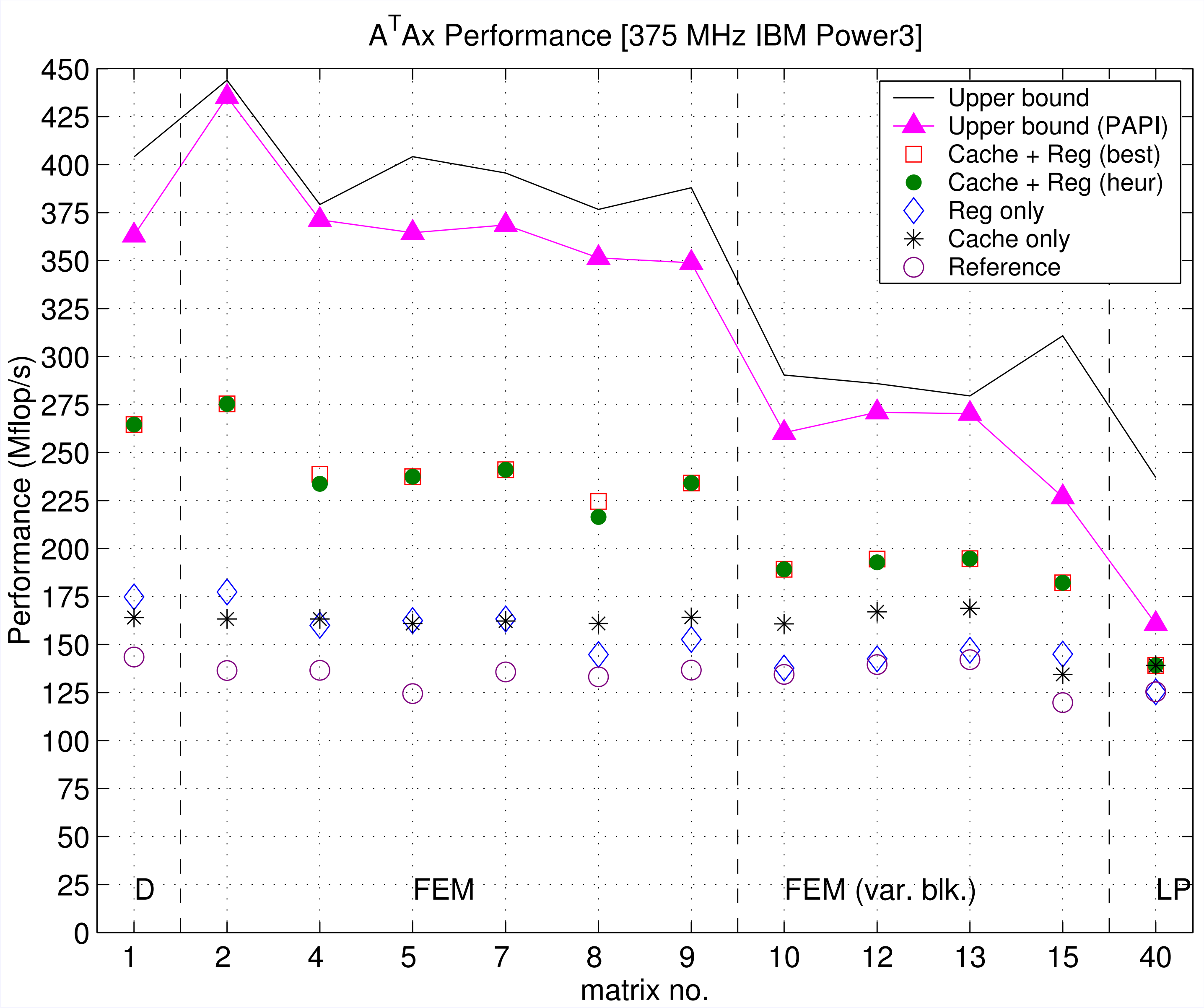
Performance Results: Intel Itanium 1



- DGEMV (n=1000): 315 Mflop/s

[<--->]

Performance Results: IBM Power3



- DGEMV (n=1000): 260 Mflop/s

[<--->]

Conclusions

- Tuning can be difficult, even when matrix structure is known
  - Performance is a complicated function of architecture and matrix
  - With memory hierarchy optimizations, 4.2x speedup possible
- Heuristic for choosing block size selects optimal implementation, or near-optimal (performance within 5–10%)
- Opportunities to apply ATLAS/PHiPAC/FFTW/. . . style tuning
  - Performance is often within 20–40% of upper-bound, particularly with FEM matrices

BeBOP: Current and Future Work (1/2)

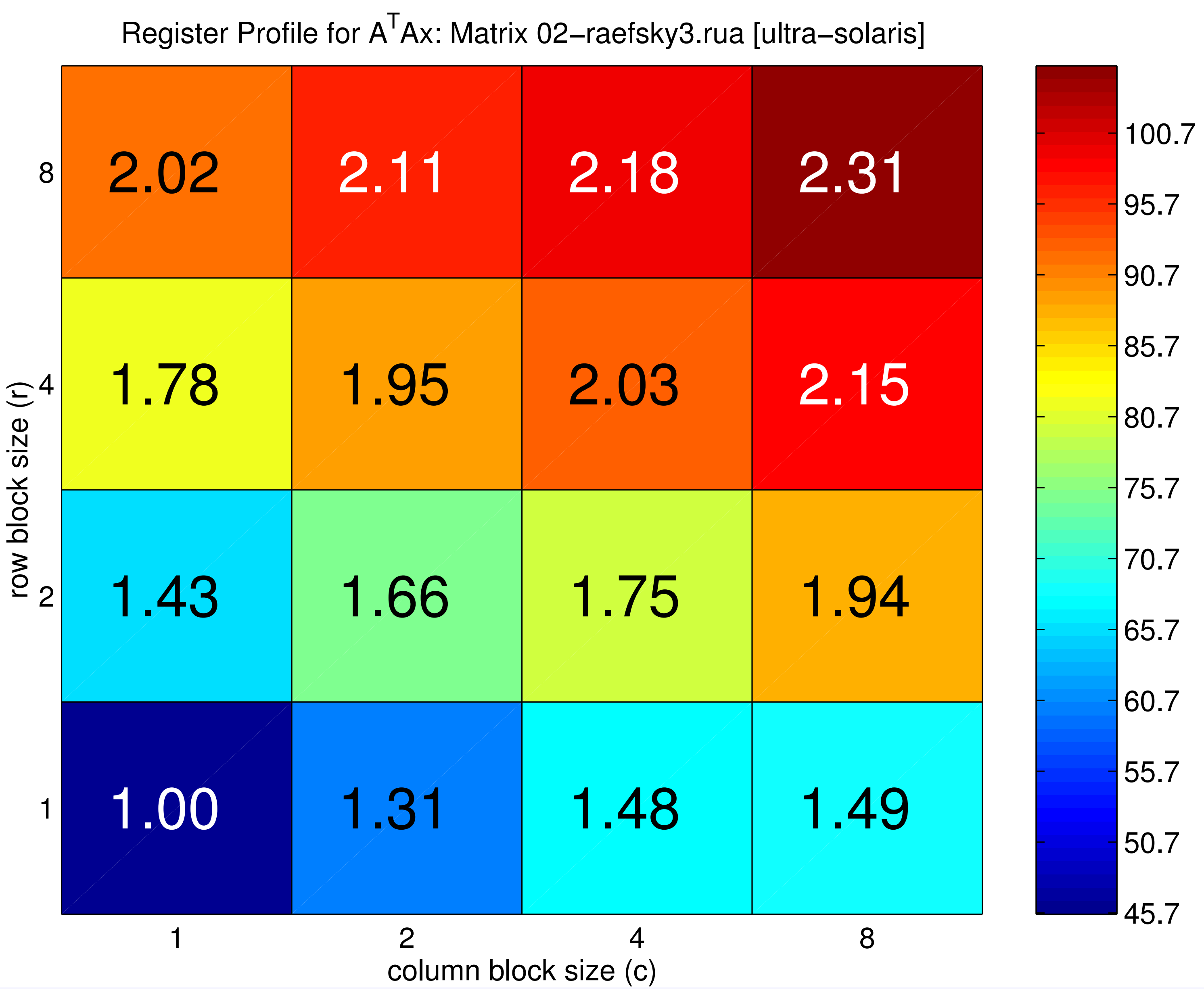
- Further performance improvements for SpM×V
  - symmetry (up to 2x speedups)
  - diagonals, block diagonals, and bands (2x),
  - splitting for variable block structure (1.5x),
  - reordering to create dense structure (1.7x),
  - cache blocking (4x)
  - multiple vectors (7x)
  - and combinations . . .
  - How to choose optimizations & tuning parameters?
- Sparse triangular solve (ICS'02: POHLL Workshop paper)
  - hybrid sparse/dense structure (1.8x)
- Higher-level kernels that permit reuse
  - AA^Tx, A^TAx (4.2x)
  - Ax and A^Ty simultaneously, A^kx, RAR^T, . . . (future work)

BeBOP: Current and Future Work (2/2)

- An automatically tuned sparse matrix library
  - Code generation via sparse compilers (Bernoulli; Bik)
  - Plan to extend new Sparse BLAS standard by one routine to support tuning
- Architectural issues
  - Latency vs. bandwidth (see paper)
  - Using models to explore architectural design space

Extra Slides

Speedups on Ultra 2i: Block Interleaved

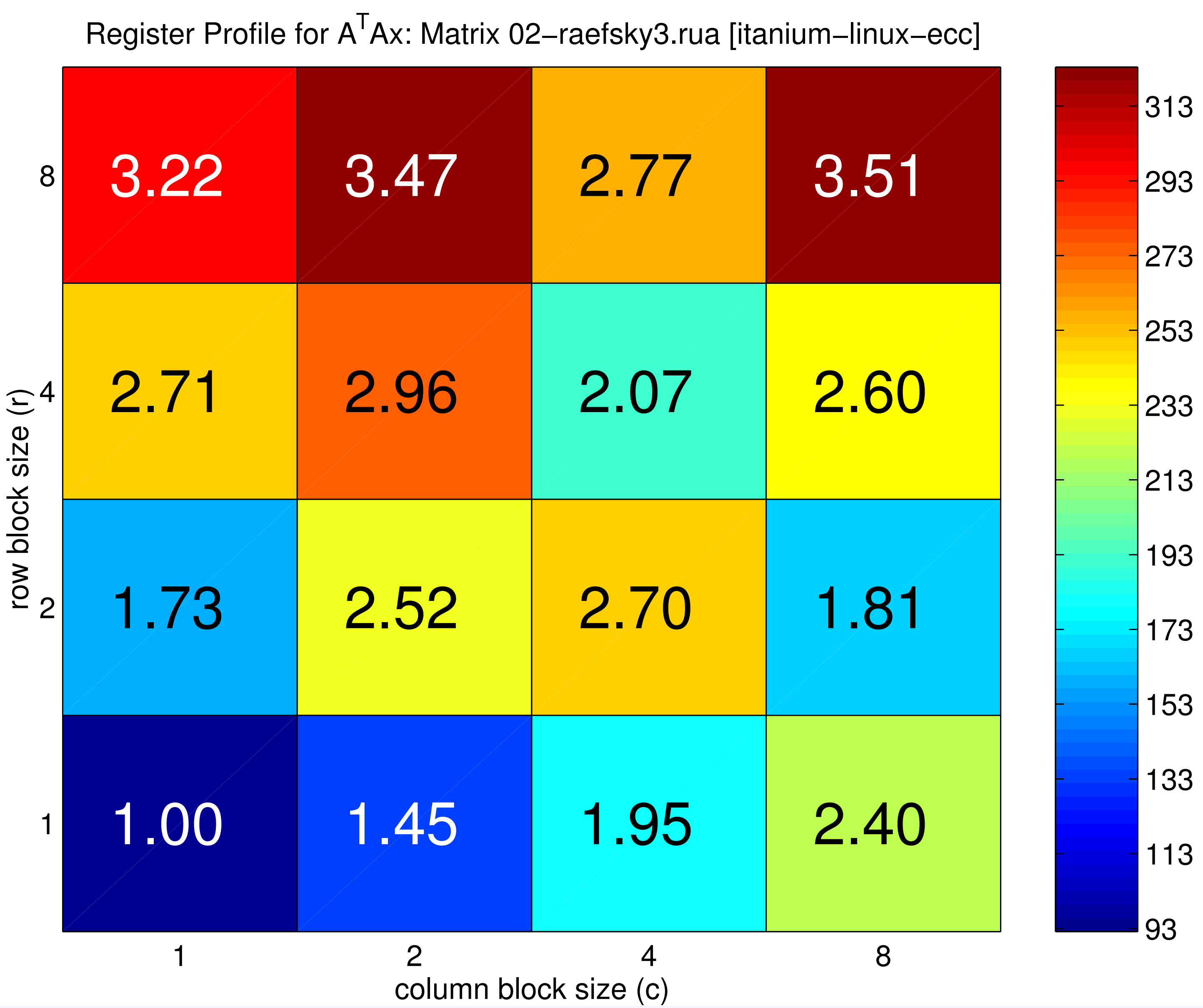


row block size (r) \ column block size (c)	1	2	4	8
8	2.02	2.11	2.18	2.31
4	1.78	1.95	2.03	2.15
2	1.43	1.66	1.75	1.94
1	1.00	1.31	1.48	1.49

(Peak machine speed: 3.2 Gflop/s)

[<—>]

Speedups on Itanium: Block Interleaved



Register Profile for A^TAx: Matrix 02–raefsky3.rua [itanium–linux–ecc]

row block size (r) \ column block size (c)	1	2	4	8
8	3.22	3.47	2.77	3.51
4	2.71	2.96	2.07	2.60
2	1.73	2.52	2.70	1.81
1	1.00	1.45	1.95	2.40

(Peak machine speed: 3.2 Gflop/s)

[<-->]

Search: Choosing the Block Size

- Off-line benchmarking (once per architecture)
  - Measure **Dense Performance (r,c)**

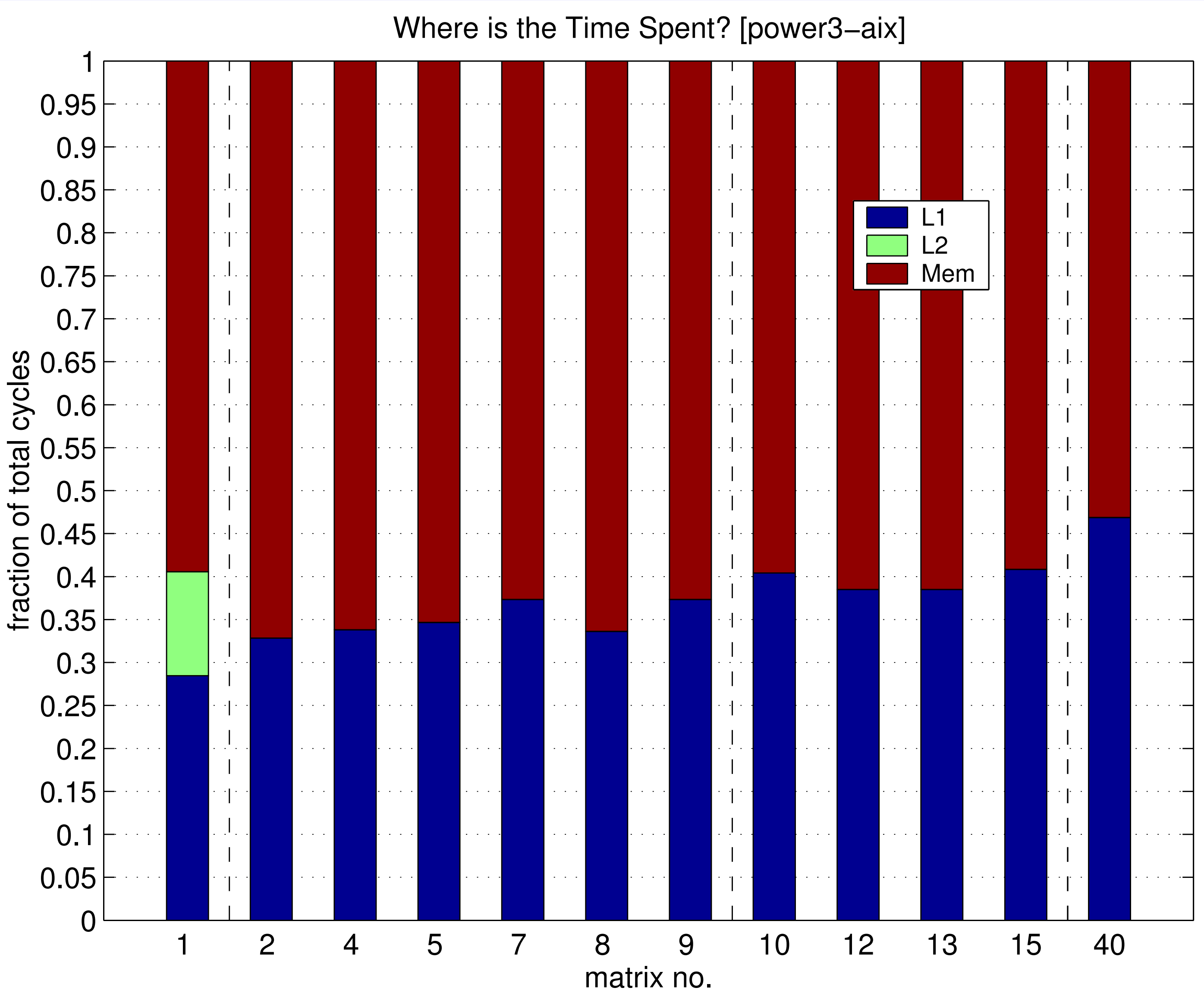
    Performance (Mflop/s) of optimized Sp$A^T A$ for a dense matrix in sparse $r \times c$ blocked format
- At run-time, when matrix is known:
  - Estimate **Fill Ratio (r,c)**, $\forall r, c$

    Fill Ratio (r,c) = (number of stored values) / (number o f true non-zeros)
  - Choose r, c that maximizes

$$\text{Est. Performance}\,(\text{r}, \text{c}) \quad = \quad \frac{\text{Dense Performance}\,(\text{r}, \text{c})}{\text{Fill Ratio}\,(\text{r}, \text{c})}$$

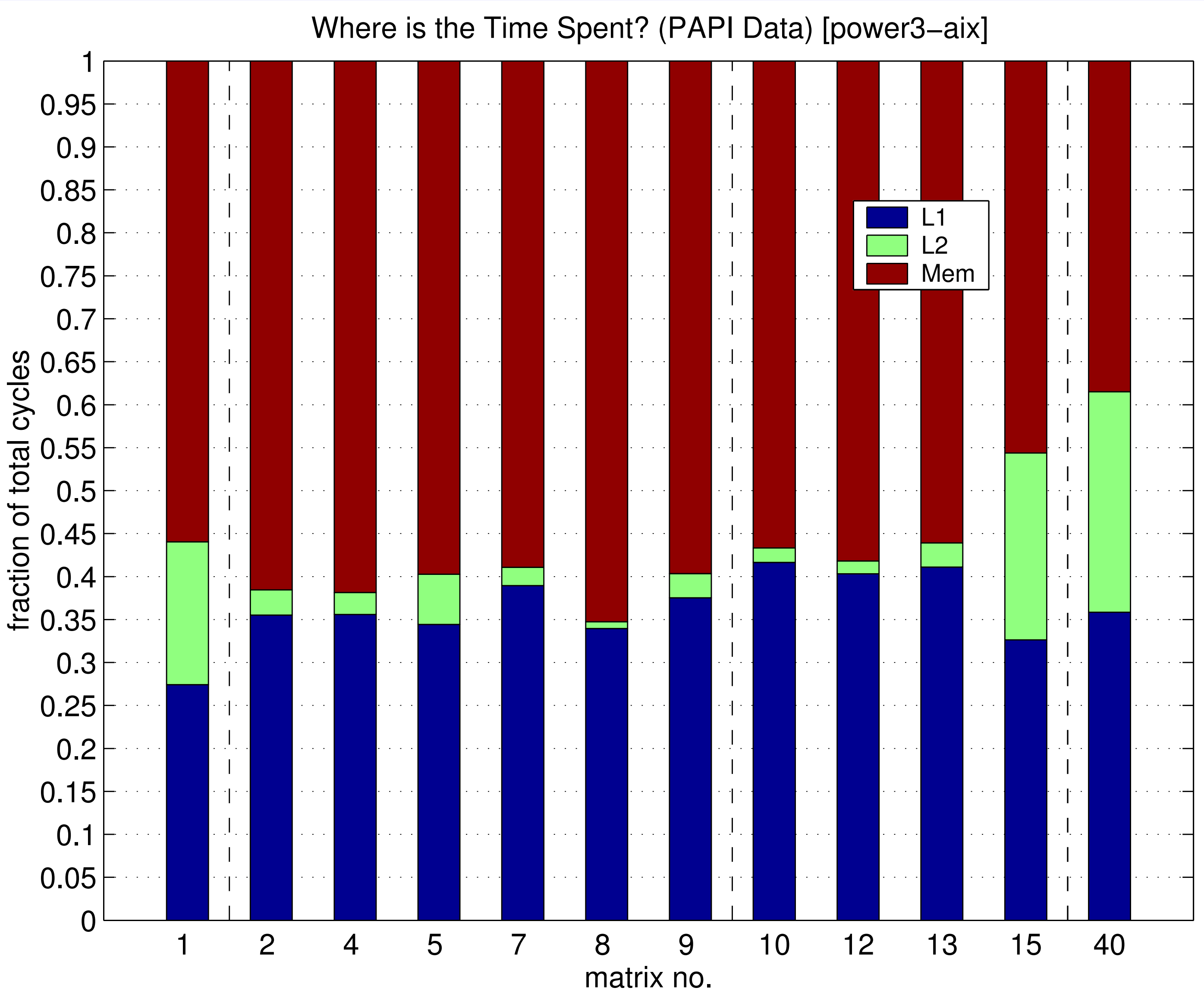
- (See SC'02)

Where Does the Time Go? Model Prediction (Power3)



[<—>]

Where Does the Time Go? PAPI (Power3)



[<−−>]

Platforms

	Sun	Intel	IBM	Intel
	Ultra 2i	Pentium III	Power3	Itanium
Clock (MHz)	333	500	375	800
Bandwidth (MB/s)	500	680	1600	2100
Peak (Mflop/s)	667	500	1500	3200
DGEMM (Mflop/s)	425	331	1300	2200
DGEMV (Mflop/s)	58	96	260	315
STREAM TRIAD (MB/s)	250	350	715	1100
No. double FP regs	16	8	32	128
L1 (KB) : line (B) : lat (cy)	16:16:1	16:32:1	64:128:.5	16:32:1
L2	2048:64:7	512:32:18	8192:128:9	96:64:6–9
L3	n/a	n/a	n/a	2048:64:21–24
Memory lat ($\approx$ cy)	36–66	26–60	35–139	36–85
Compiler	cc 6	icc 7	xlc 5.1	icc 7

[<—>]

References

[BACD97] J. Bilmes, K. Asanović, 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. Chapter 3: `www.netlib.org/blast`.

[BDG$^+$00] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of Supercomputing*, November 2000.

[BW99] Aart J. C. Bik and Harry A. G. Wijshoff. Automatic nonzero structure analysis. *SIAM Journal on Computing*, 28(5):1576–1587, 1999.

[Dem97] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[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, Ramón 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.

[GGHvdG01] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Transactions on Mathematical Software*, 27(4), December 2001.

[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.

[GR99] Roman Geus and S. Röllin. 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.

[JBWS97] III James B. White and P. Sadayappan. On improving the performance of sparse matrix-vector multiplication. In *Proceedings of the International Conference on High-Performance C omputing*, 1997.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[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.

[Neu98] T. Neubert. Anwendung von generativen Programmiertechniken am Beispiel der Matrixalgebra. Master's thesis, Technische Universität Chemnitz, 1998.

[NGLPJ96] J. J. Navarro, E. García, 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 Püschel, Bryan Singer, Manuela Veloso, and José 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`.

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

[SL98] Jeremy G. Siek and Andrew Lumsdaine. A rational approach to portable high performance: the Basic Linear Algebra Instruction Set (BLAIS) and the Fixed Algorithm Size Template (fast) library. In *Proceedings of ECOOP*, 1998.

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

[TJ92] O. Temam and W. Jalby. Characterizing the behavior of sparse algorithms on caches. In *Proceedings of Supercomputing '92*, 1992.

[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.

[Vel98] Todd Veldhuizen. Arrays in blitz++. In *Proceedings of ISCOPE*, volume 1505 of *LNCS*. Springer-Verlag, 1998.

[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.

[WO95] Weichung Wang and Dianne P. O'Leary. Adaptive use of iterative methods in interior point methods for linear programming. Technical Report UMIACS-95-111, University of Maryland at College Park, College Park, MD, USA, 1995.

[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.