Performance Optimizations and Bounds for Sparse Matrix-Vector Multiply

Richard Vuduc, James Demmel, Katherine Yelick
Shoaib Kamil, Rajesh Nishtala, Benjamin Lee

Wednesday, November 20, 2002

Berkeley Benchmarking and OPtimization (BeBOP) Project
www.cs.berkeley.edu/~richie/bebop
Computer Science Division, U.C. Berkeley
Berkeley, California, USA

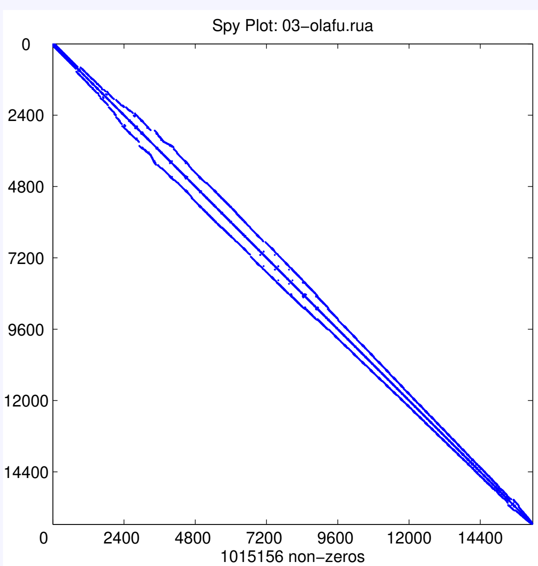
.

SC 2002: Session on Sparse Linear Algebra

Context: Performance Tuning in the Sparse Case

- Application performance dominated by a few *computational kernels*
- Performance tuning today
  - Vendor-tuned libraries (*e.g.*, BLAS) or user hand-tunes
  - Automatic tuning (*e.g.*, PHiPAC/ATLAS, FFTW/SPIRAL/UHFFT)
- Tuning sparse linear algebra kernels is hard
  - Sparse code has . . .
    - high bandwidth requirements (extra storage)
    - poor locality (indirect, irregular memory access)
    - poor instruction mix (data structure manipulation)
  - Sparse matrix-vector multiply (SpM$\times$V) performance: less than 10% of machine peak
  - **Performance depends on kernel, architecture, and matrix**

Example: Matrix `olafu`

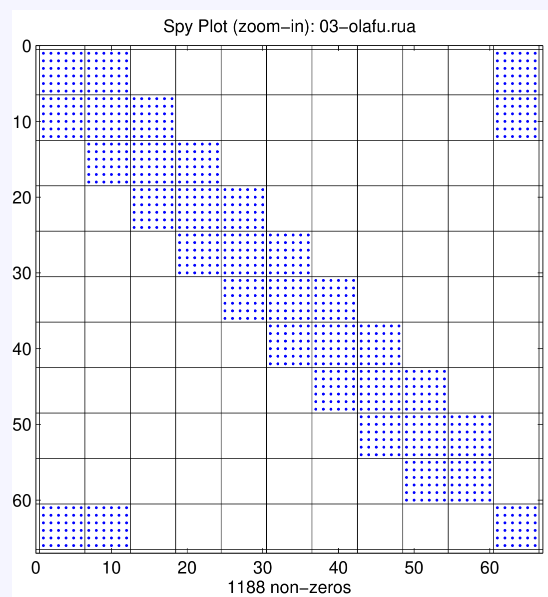


N = 16146

nnz = 1.0M

Kernel = SpM$\times$V

Example: Matrix `olafu`



N = 16146

nnz = 1.0M

Kernel = SpM$\times$V

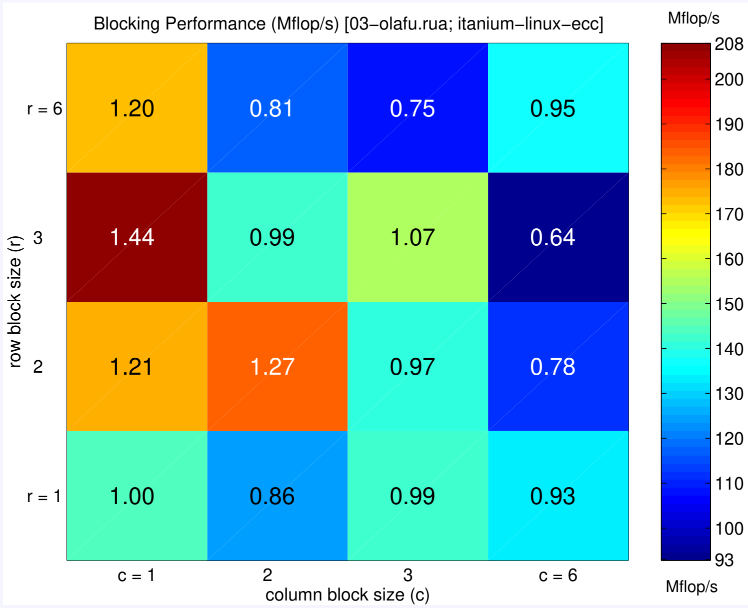
A natural choice:

6×6 blocked compressed sparse row (BCSR).

Experiment:

Measure performance of all $r\times c$ block sizes for $r, c \in \{1, 2, 3, 6\}$.

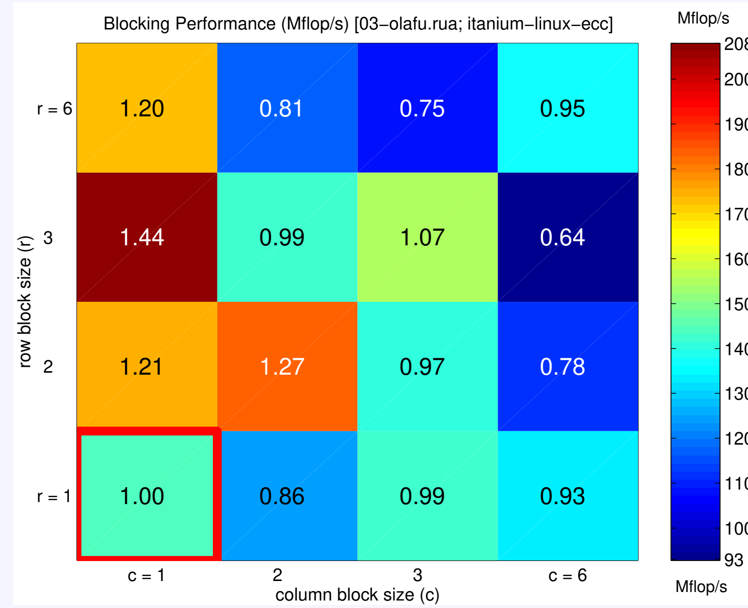
Speedups on Itanium: The Need for Search



(Peak machine speed: 3.2 Gflop/s)

[<—>]

Speedups on Itanium: The Need for Search

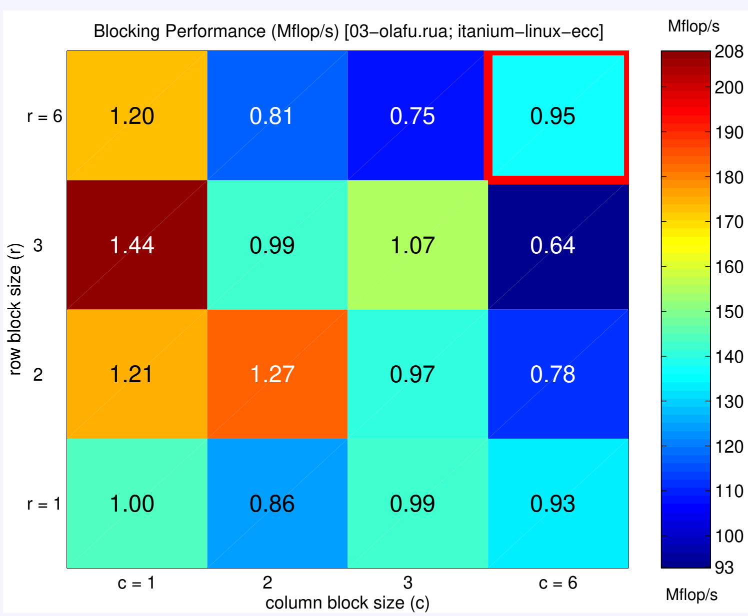


(Peak machine speed: 3.2 Gflop/s)

[<—>]

Performance Optimizations and Boundsfor Sparse Matrix-Vector Multiply – p.4/36

Speedups on Itanium: The Need for Search

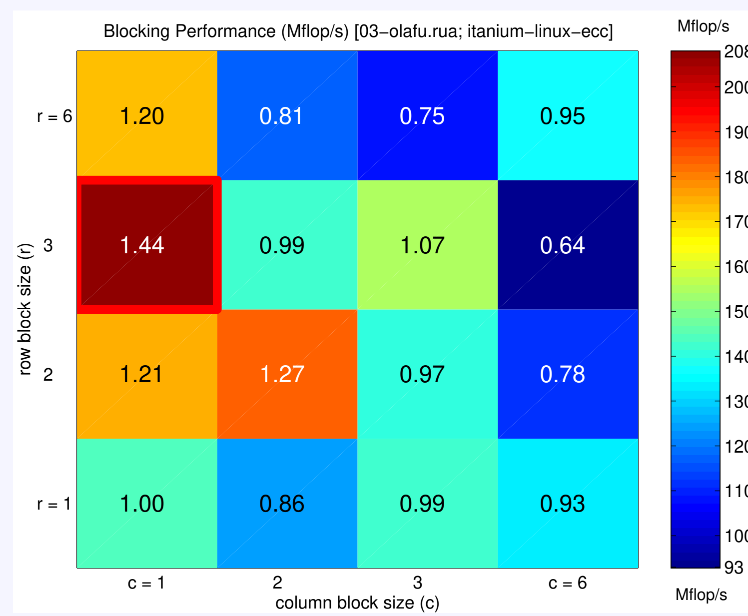


(Peak machine speed: 3.2 Gflop/s)

[<—>]

Performance Optimizations and Boundsfor Sparse Matrix-Vector Multiply – p.4/36

Speedups on Itanium: The Need for Search

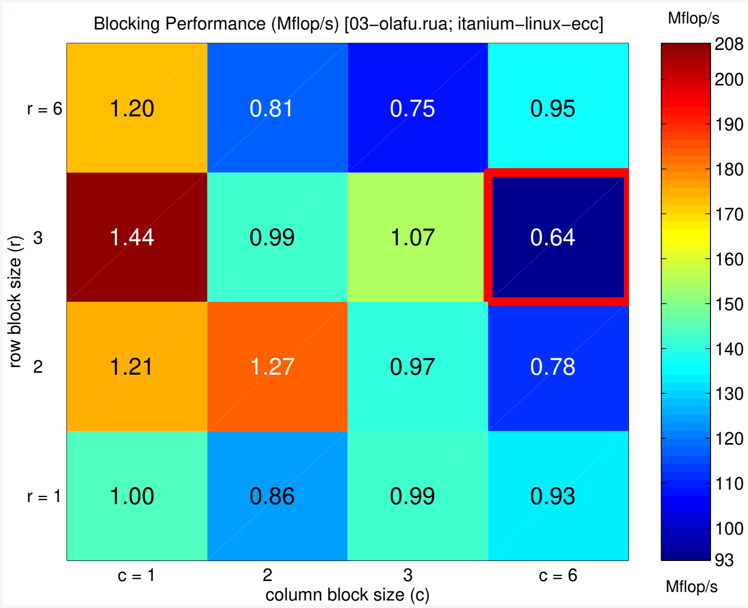


(Peak machine speed: 3.2 Gflop/s)

[<—>]

Performance Optimizations and Boundsfor Sparse Matrix-Vector Multiply – p.4/36

Speedups on Itanium: The Need for Search



(Peak machine speed: 3.2 Gflop/s)

Key Questions and Conclusions

- How do we choose the best data structure automatically?
  - New heuristic for choosing optimal (or near-optimal) block sizes
- What are the limits on performance of blocked SpM$\times$V?
  - Derive performance upper bounds for blocking
  - Often within 20% of upper bound, placing limits on improvement from more “low-level” tuning
  - Performance is memory-bound: reducing data structure size is critical
- Where are the new opportunities (kernels, techniques) for achieving higher performance?
  - Identify cases in which blocking does and does not work
  - Identify new kernels and opportunities for reducing memory traffic

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 [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, hypre, . . .

Approach to Automatic Tuning

- For each kernel,
  - *Identify* and *generate* a space of implementations
  - *Search* to find the fastest (using models, experiments)

Approach to Automatic Tuning

- For each kernel,
  - *Identify* and *generate* a space of implementations
  - *Search* to find the fastest (using models, experiments)
- The SPARSITY system for SpM×V [Im & Yelick '99]
  - Interface
    - Input: Your sparse matrix (CSR)
    - Output: Data structure + routine tuned to your matrix & machine
  - Implementation space
    - register level blocking ($r \times c$)
    - cache blocking, multiple vectors, ...
  - Search
    - Off-line: benchmarking (once per architecture)
    - Run-time: estimate matrix properties ("search") and predict best data structure parameters

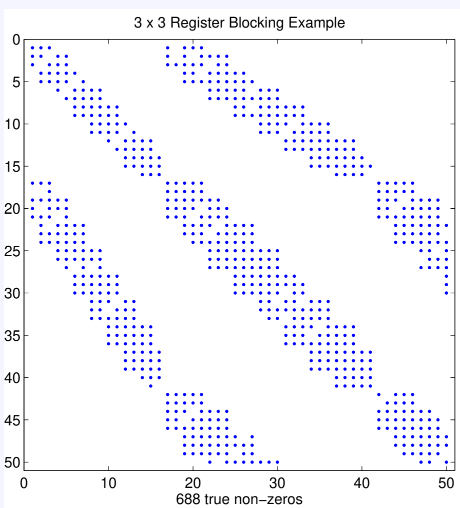
[<—>]

Approach to Automatic Tuning

- For each kernel,
  - *Identify* and *generate* a space of implementations
  - *Search* to find the fastest (using models, experiments)
- The SPARSITY system for SpM×V [Im & Yelick '99]
  - Interface
    - Input: Your sparse matrix (CSR)
    - Output: Data structure + routine tuned to your matrix & machine
  - Implementation space
    - **register level blocking ($r \times c$)**
    - cache blocking, multiple vectors, ...
  - Search
    - **Off-line: benchmarking** (once per architecture)
    - **Run-time: estimate matrix properties ("search") and predict best data structure parameters**

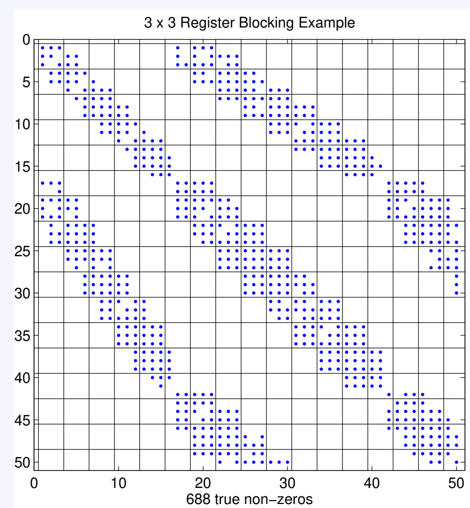
[<—>]

Register-Level Blocking (SPARSITY): 3x3 Example



[<—>]

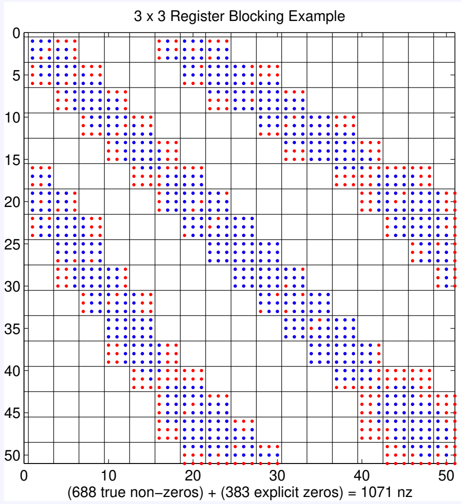
Register-Level Blocking (SPARSITY): 3x3 Example



- 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 Pentium III

[<—>]

Search: Choosing the Block Size

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

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

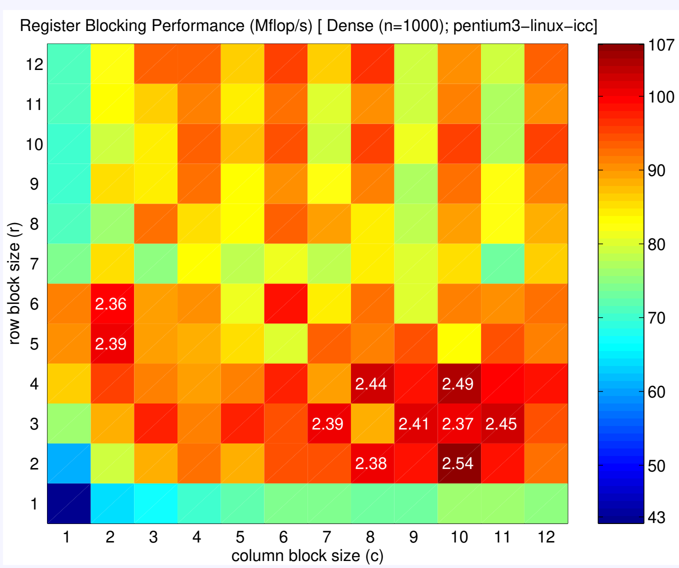
    Fill Ratio (r,c) = (number of stored values) / (number of true non-zeros)
  - Choose r, c that maximizes

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

- (Replaces previous SPARSITY heurstic)

[<—>]

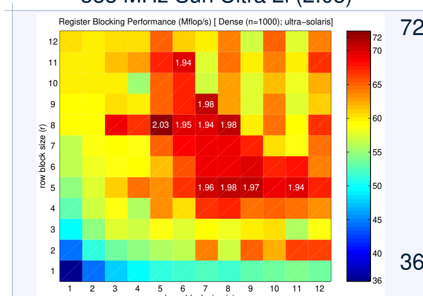
Off-line Benchmarking [Intel Pentium III]



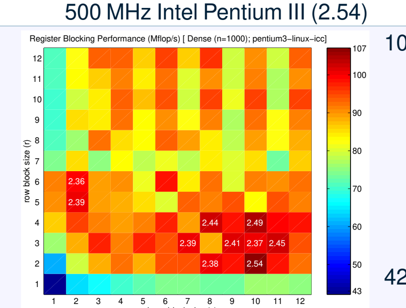
Top 10 codes labeled by speedup over unblocked code. Max speedup = 2.54 (2×10).

[<—>]

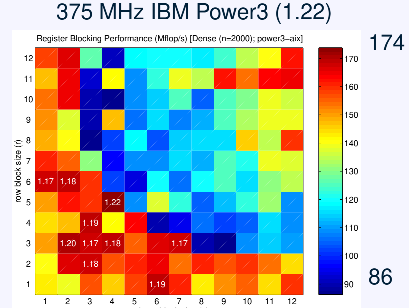
333 MHz Sun Ultra 2i (2.03)



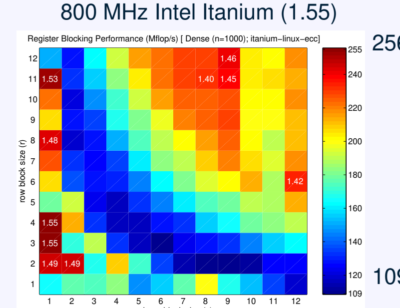
500 MHz Intel Pentium III (2.54)



375 MHz IBM Power3 (1.22)



800 MHz Intel Itanium (1.55)



[<—>]

Performance Bounds for Register Blocking

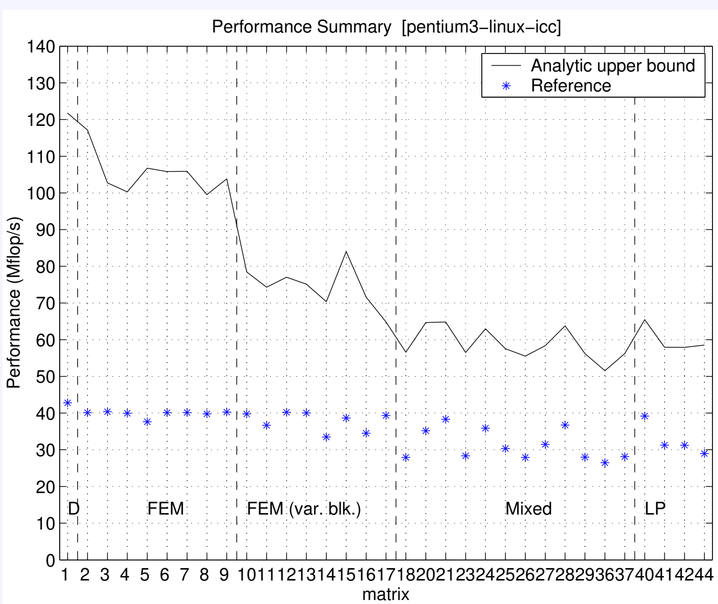
- How close are we to the speed limit of blocking?
- *Upper-bounds* on performance: (flops) / (time)
  - Flops $\approx$ 2 * (number of true non-zeros)
  - Lower-bound on time: Two key assumptions
    - Consider only *memory* operations
    - Count only compulsory misses (*i.e.*, ignore conflicts)
  - Other features of model
    - Account for cache line size
    - Bound is function of matrix and r, c
- Model of execution time (see paper)
  - Charge full latency (α_i) for hits at each cache level i, *e.g.*,

$$
\begin{aligned}
T &= (\text{L1 hits})\alpha_1 + (\text{L2 hits})\alpha_2 + (\text{mem hits})\alpha_{\text{mem}} \\
&= (\text{Loads})\alpha_1 + (\text{L1 misses})(\alpha_2 - \alpha_1) + (\text{L2 misses})(\alpha_{\text{mem}} - \alpha_2)
\end{aligned}
$$

Overview of Performance Results

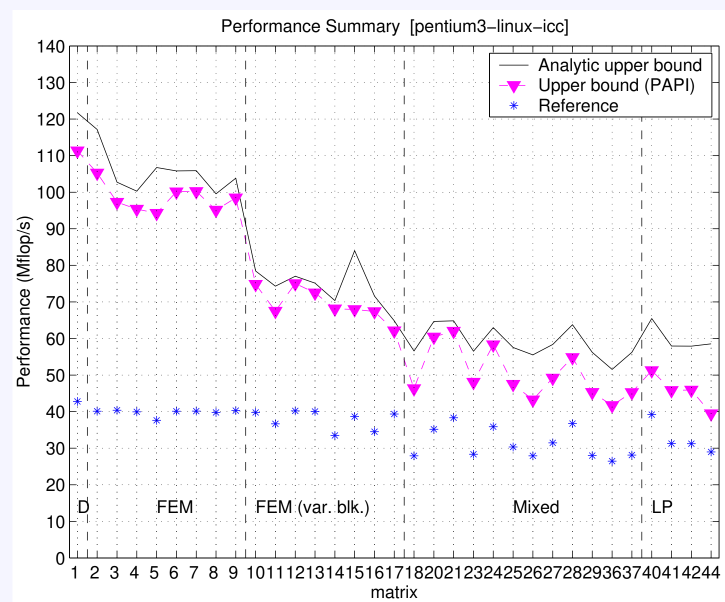
- Experimental setup
  - Four machines: Pentium III, Ultra 2i, Power3, Itanium
  - 44 matrices: dense, finite element, mixed, linear programming
  - Measured misses and cycles using PAPI [BDG+00]
  - Reference: CSR (*i.e.*, 1×1 or "unblocked")
- Main observations
  - SPARSITY vs. reference: up to 2.5x faster, especially on FEM
  - Block size selection: chooses within 10% of best
  - SPARSITY performance typically within 20% of upper-bound
  - SPARSITY least effective on Power3

Performance Results: Intel Pentium III



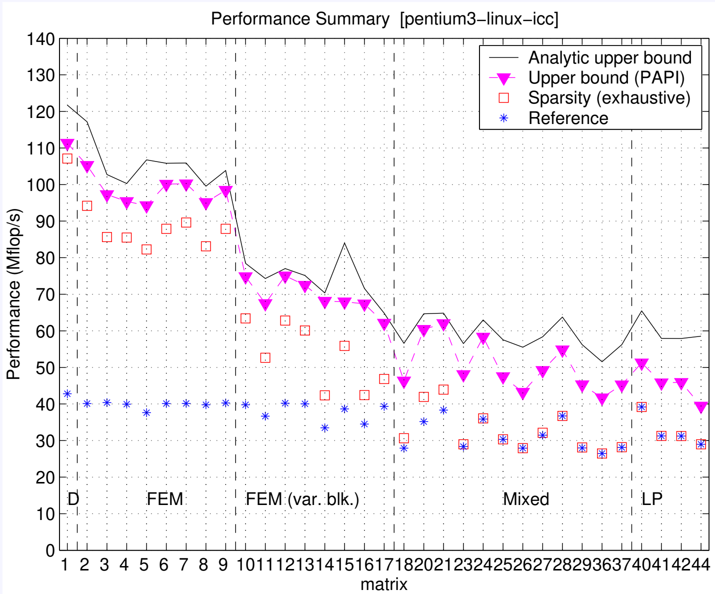
- DGEMV (n=1000): 96 Mflop/s

Performance Results: Intel Pentium III



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

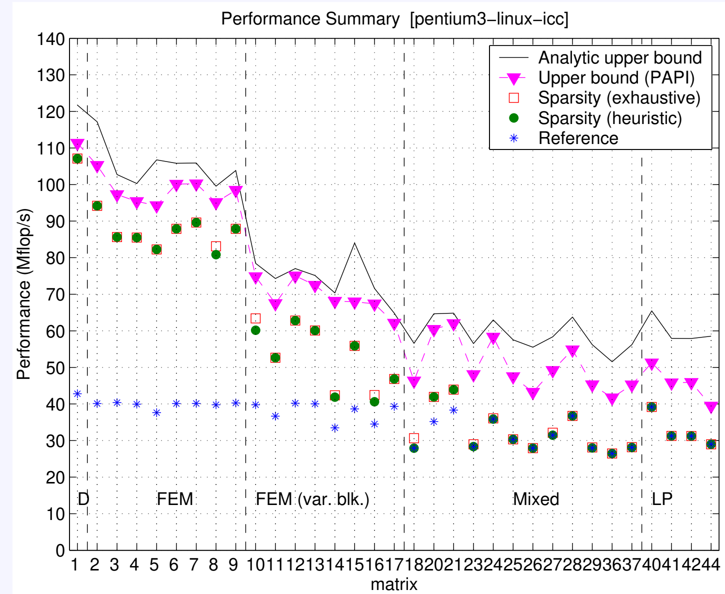
Performance Results: Intel Pentium III



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

[<—>]

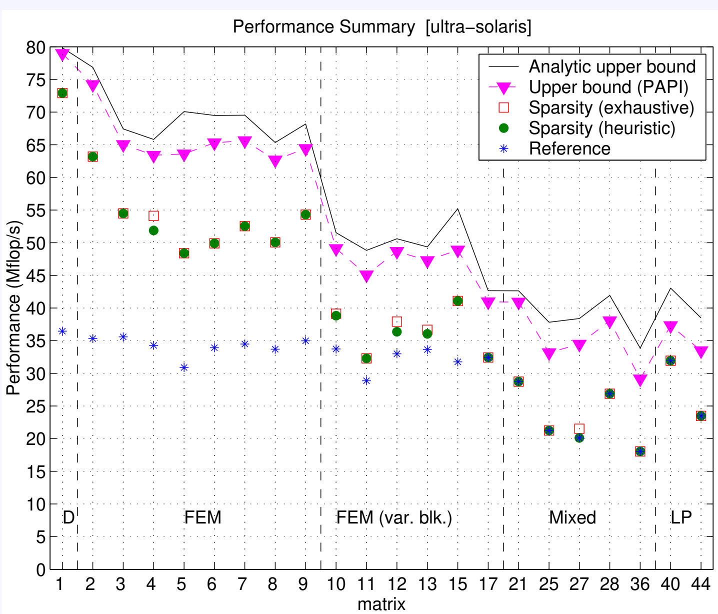
Performance Results: Intel Pentium III



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

[<—>]

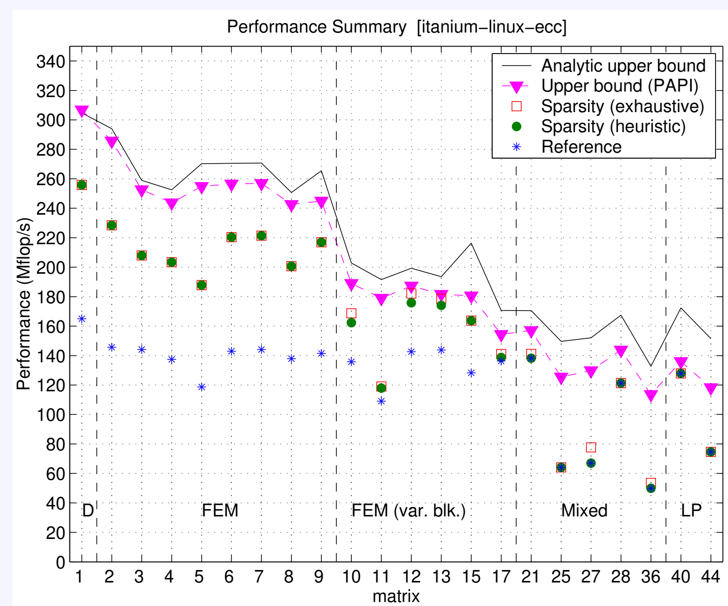
Performance Results: Sun Ultra 2i



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

[<—>]

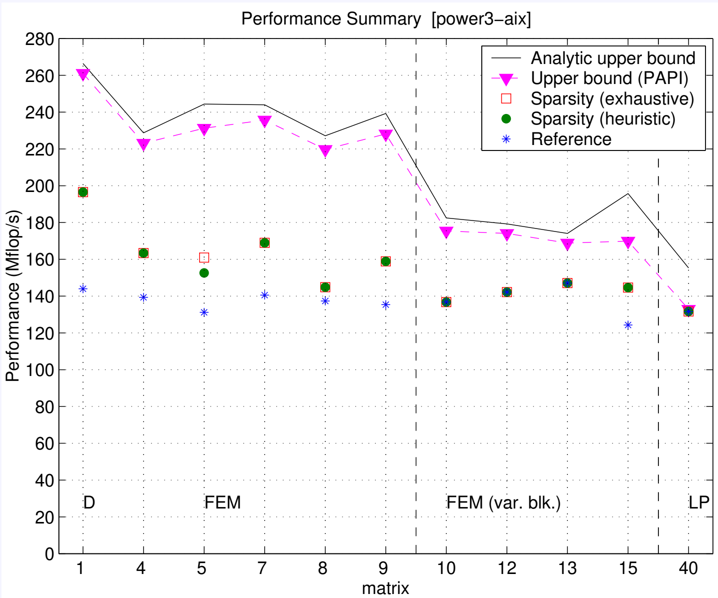
Performance Results: Intel Itanium



- DGEMV (n=1000): 310 Mflop/s

[<—>]

Performance Results: IBM Power3



- DGEMV (n=2000): 260 Mflop/s

Conclusions

- Tuning can be difficult, even when matrix structure is known
  - Performance is a complicated function of architecture and matrix
- New heuristic for choosing block size selects optimal implementation, or near-optimal (performance within 5–10%)
- Limits of low-level tuning for blocking are near
  - Performance is often within 20% of upper-bound, particularly with FEM matrices
  - Unresolved: closing the gap on the Power3

BeBOP: Current and Future Work (1/2)

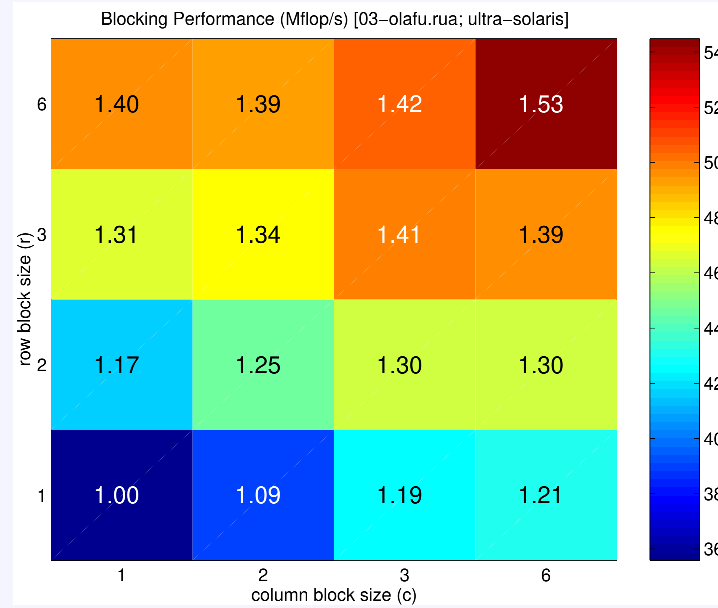
- Further performance improvements
  - symmetry (1.5–2x speedups)
  - diagonals, block diagonals, and bands (1.2–2x),
  - splitting for variable block structure (1.3–1.7x),
  - reordering to create dense structure (1.7x),
  - cache blocking (1.5–4x)
  - multiple vectors (2–7x)
  - and combinations . . .
  - How to choose optimizations & tuning parameters?
- Sparse triangular solve (ICS'02: POHLL Workshop paper)
  - hybrid sparse/dense structure (1.2–1.8x)
- Higher-level kernels that permit reuse
  - $AA^T x$, $A^T A x$ (1.5–3x)
  - Ax and $A^T y$ simultaneously, $A^k x$, RAR^T, . . . (future work)

BeBOP: Current and Future Work (2/2)

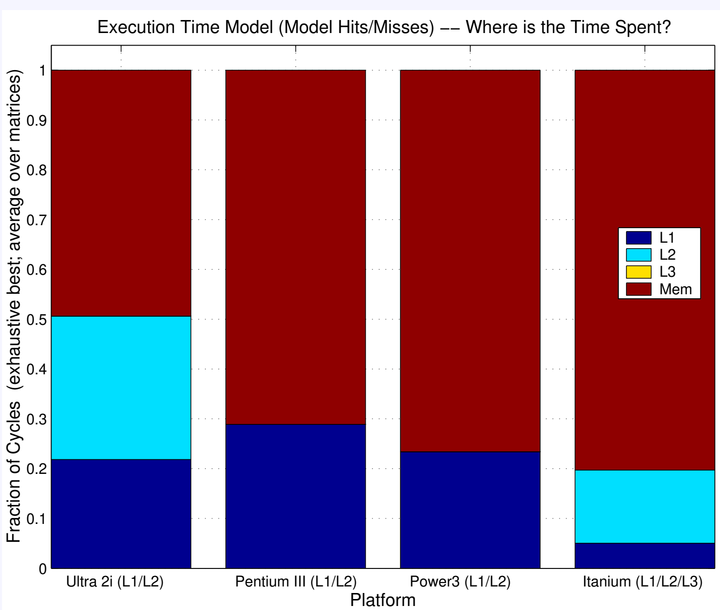
- An automatically tuned sparse matrix library
  - Code generation via sparse compilers (Bernoulli; Bik)
  - Plan to extend new Sparse BLAS standard by one routine to support tuning
- Architectural issues
  - Improvements for Power3?
  - Latency vs. bandwidth (see paper)
  - Using models to explore architectural design space

EXTRA SLIDES

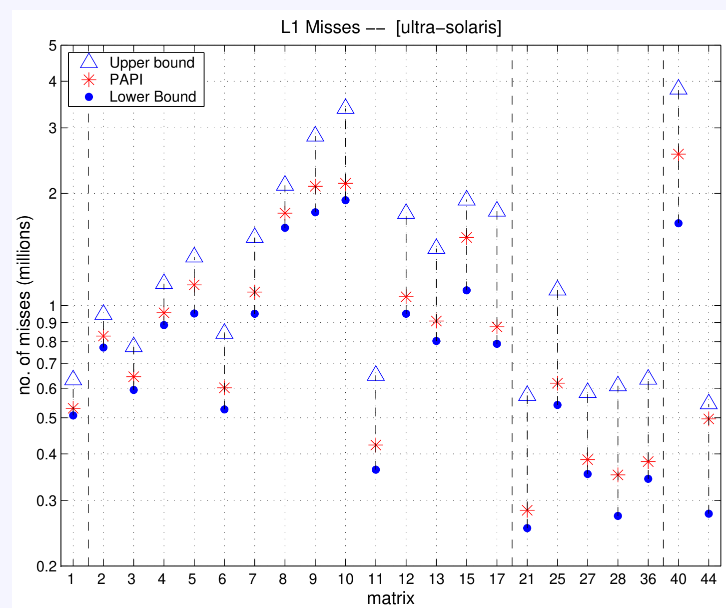
Example: No Big Surprises on Sun Ultra 2i



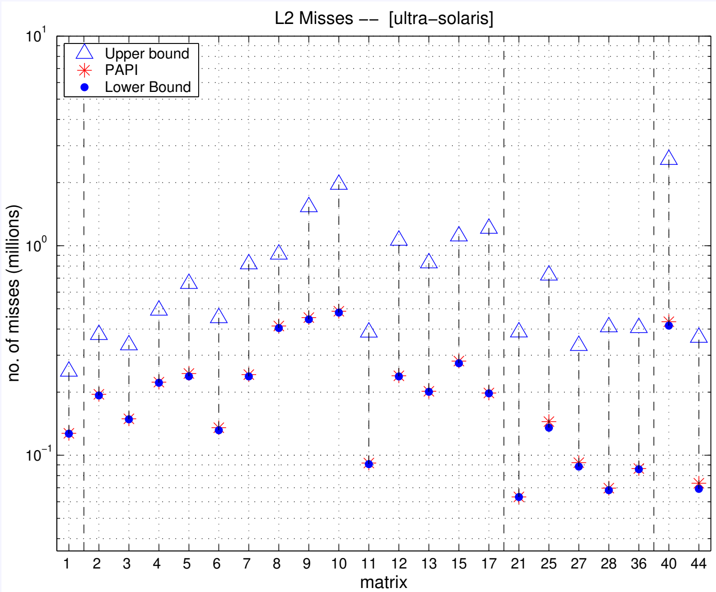
Where in Memory is the Time Spent?



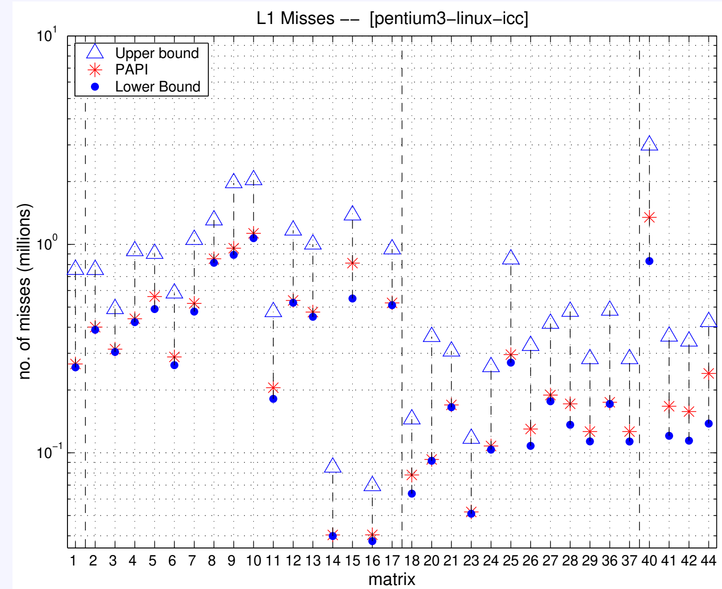
Cache Miss Bound Verification: Sun Ultra 2i (L1)



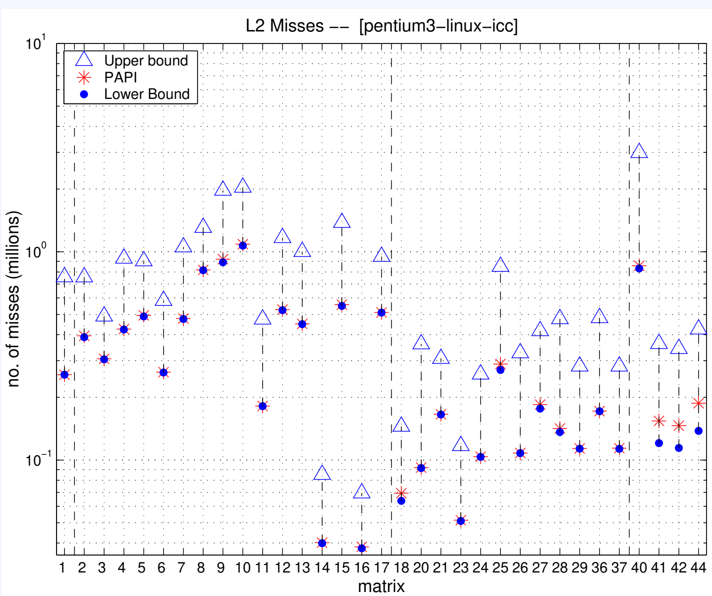
Cache Miss Bound Verification: Sun Ultra 2i (L2)



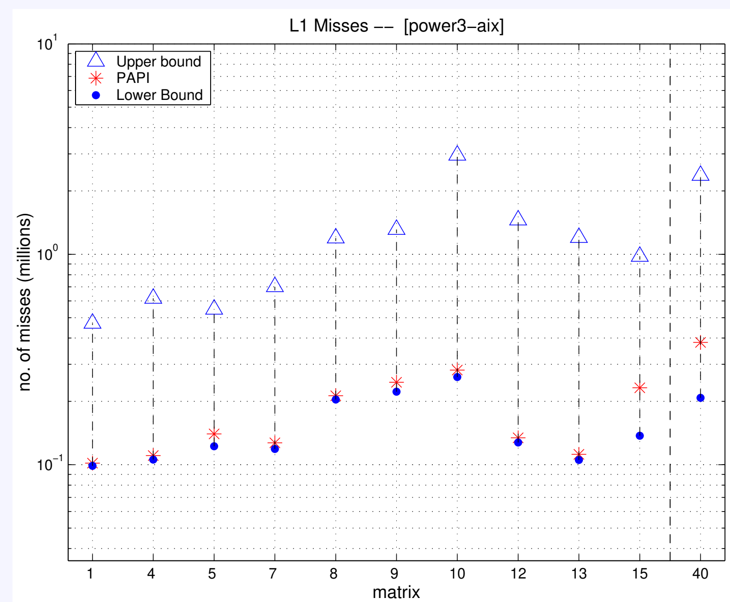
Cache Miss Bound Verification: Intel Pentium III (L1)



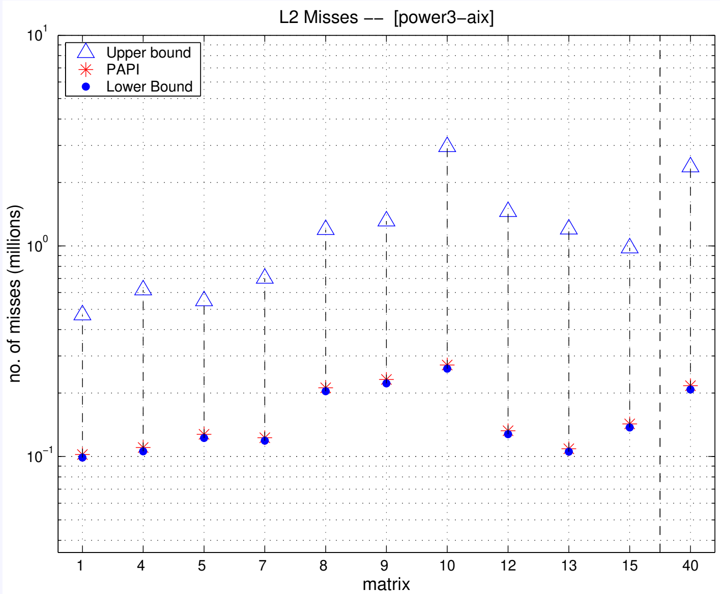
Cache Miss Bound Verification: Intel Pentium III (L2)



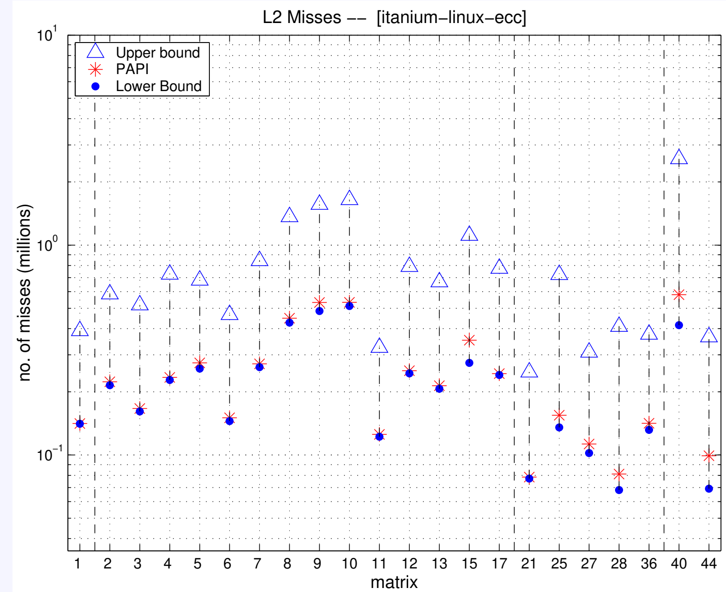
Cache Miss Bound Verification: IBM Power3 (L1)



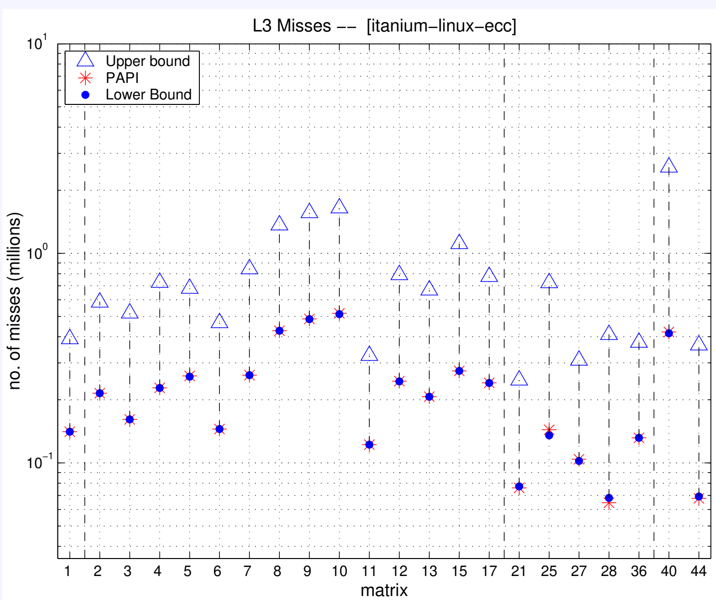
Cache Miss Bound Verification: IBM Power3 (L2)



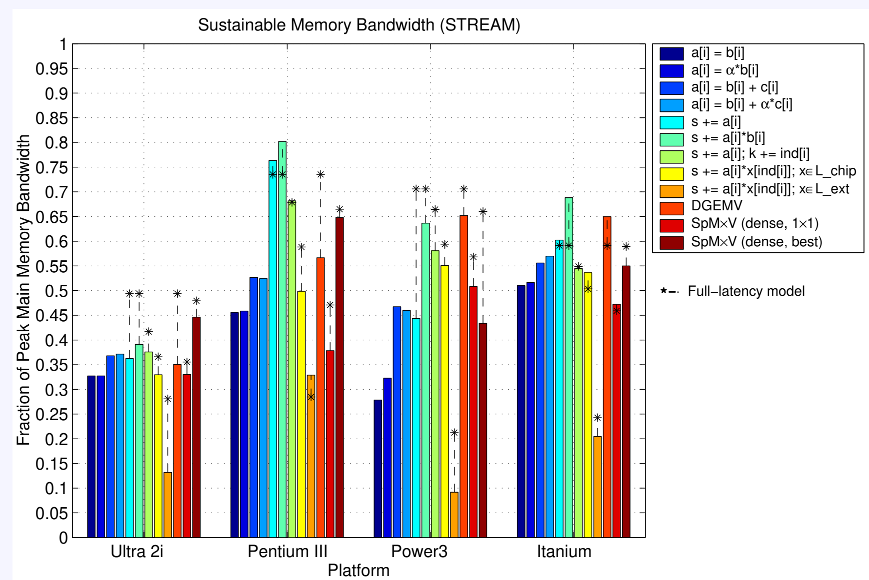
Cache Miss Bound Verification: Intel Itanium (L2)



Cache Miss Bound Verification: Intel Itanium (L3)



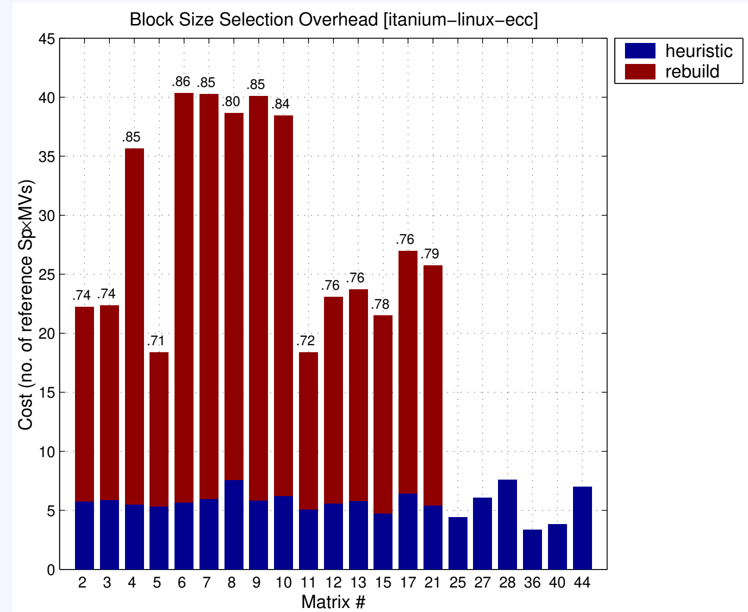
Latency vs. Bandwidth



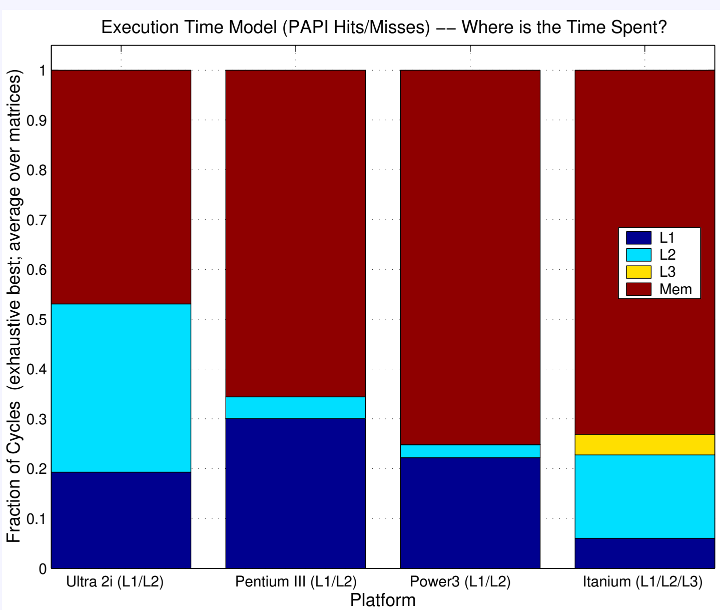
Related Work (2/2)

- 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)

What is the Cost of Search?



Where in Memory is the Time Spent? (PAPI Data)



Fill: Some Surprises!

- Sometimes faster to fill in many zeros

Matrix	Speedup	Fill ratio	$\frac{\text{(Size BCSR)}}{\text{(Size CSR)}}$	Dense $\frac{\text{(Perf BCSR)}}{\text{(Perf CSR)}}$	Platform
11	1.09	1.70	1.26	1.55	Itanium
13	**1.50**	**1.52**	**1.07**	**2.30**	Pentium 3
17	1.04	1.59	1.23	1.54	Itanium
27	1.16	1.94	1.47	1.54	Itanium
27	1.10	1.53	1.25	1.23	Ultra 2i
29	1.00	1.98	1.44	1.89	Pentium 3

[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.

[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.

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. `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.

[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.

[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.htm`

[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.

[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.