Performance Models for Evaluation and Automatic Tuning of Symmetric Sparse Matrix-Vector Multiply

University of California, Berkeley
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

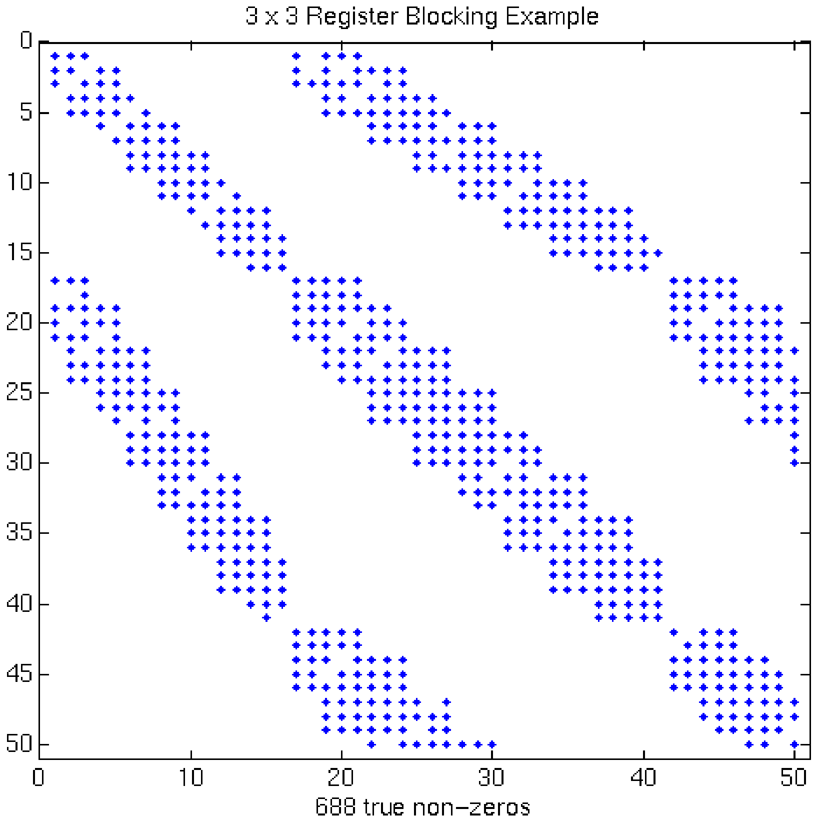
16 August 2004

Performance Tuning Challenges

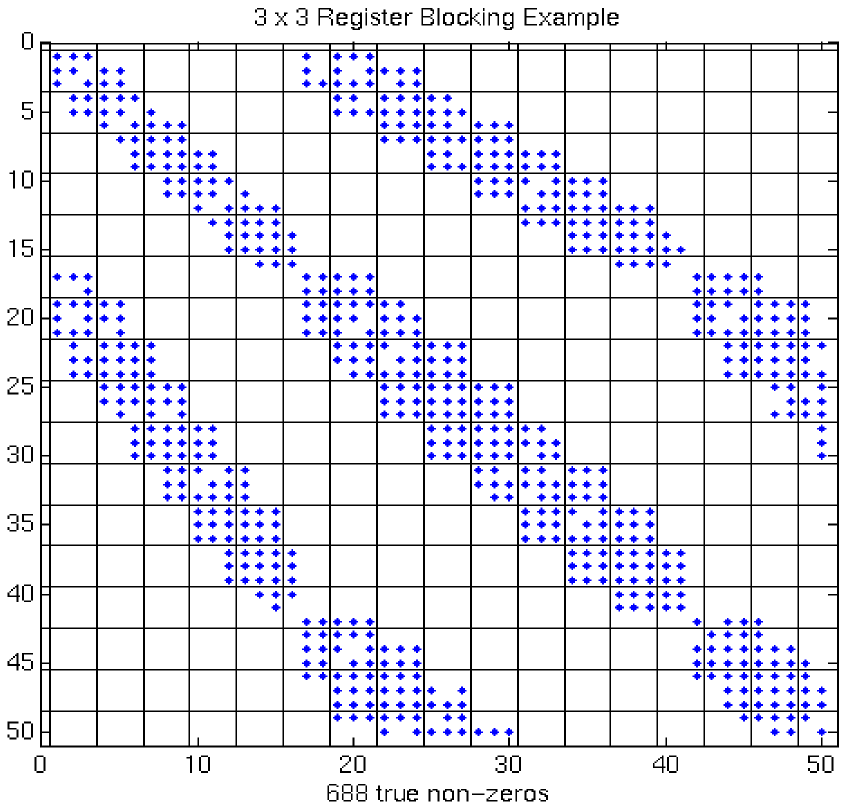
- Computational Kernels
  - Sparse Matrix-Vector Multiply (SpMV): $y = y + Ax$
    - A : Sparse matrix, symmetric ($i.e.,\ A = A^T$)
    - x, y : Dense vectors
  - Sparse Matrix-Multiple Vector Multiply (SpMM): $Y = Y + AX$
    - X, Y : Dense matrices

- 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: Register Blocking (1/3)

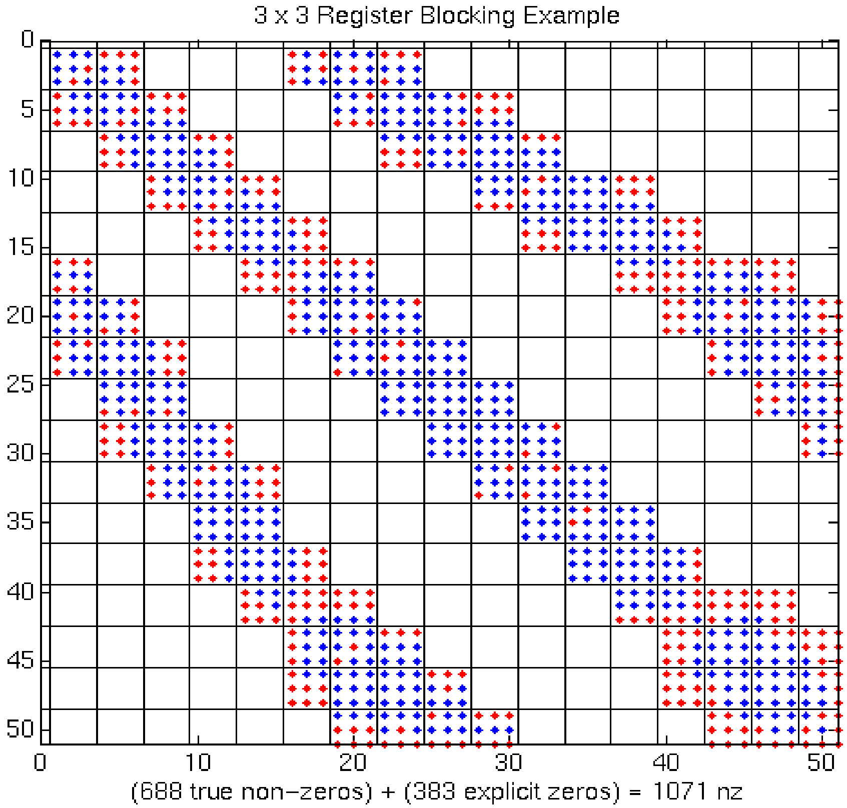


Optimizations: Register Blocking (2/3)



- BCSR with uniform, aligned grid

Optimizations: Register Blocking (3/3)



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

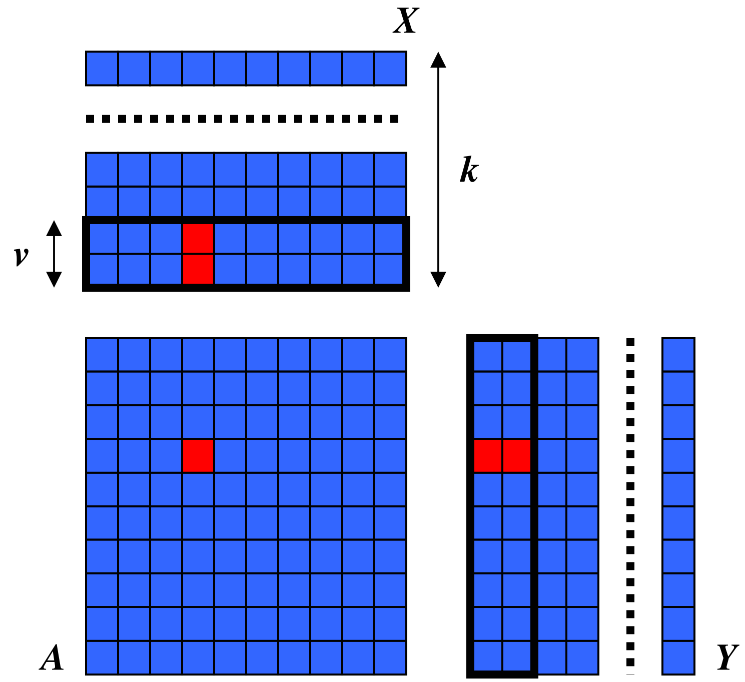
Optimizations: Matrix Symmetry

- 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

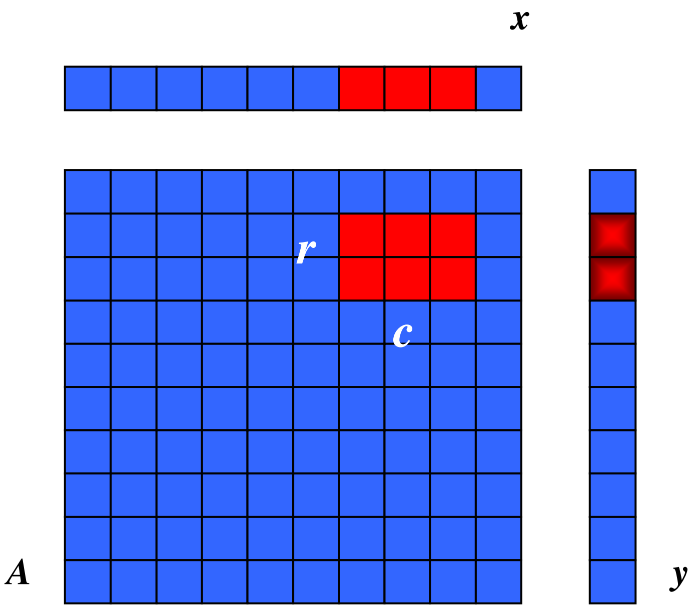
Optimizations: Multiple Vectors

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



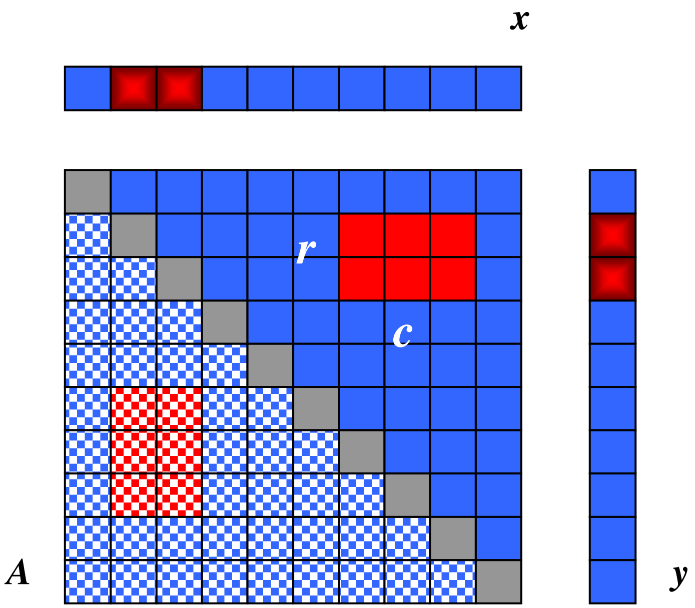
Optimizations: Register Usage (1/3)

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



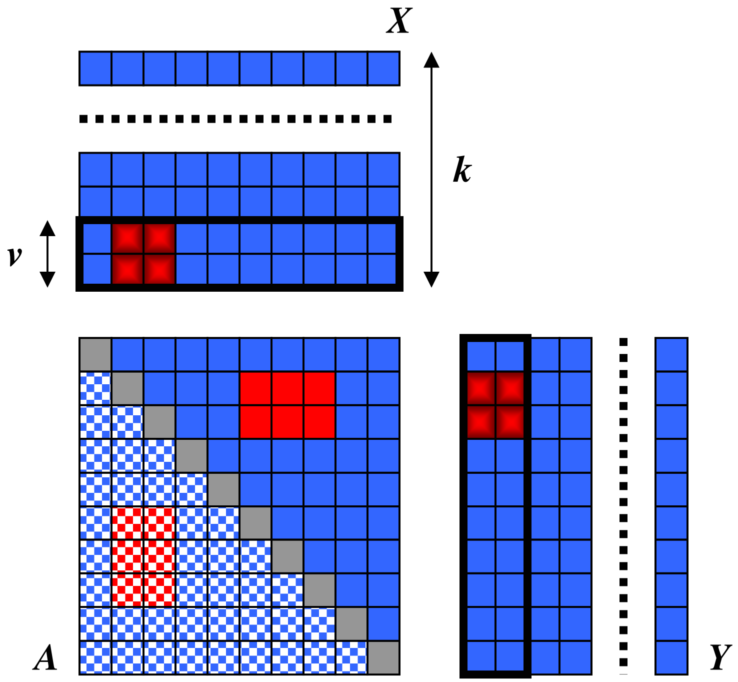
Optimizations: Register Usage (2/3)

- 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***)



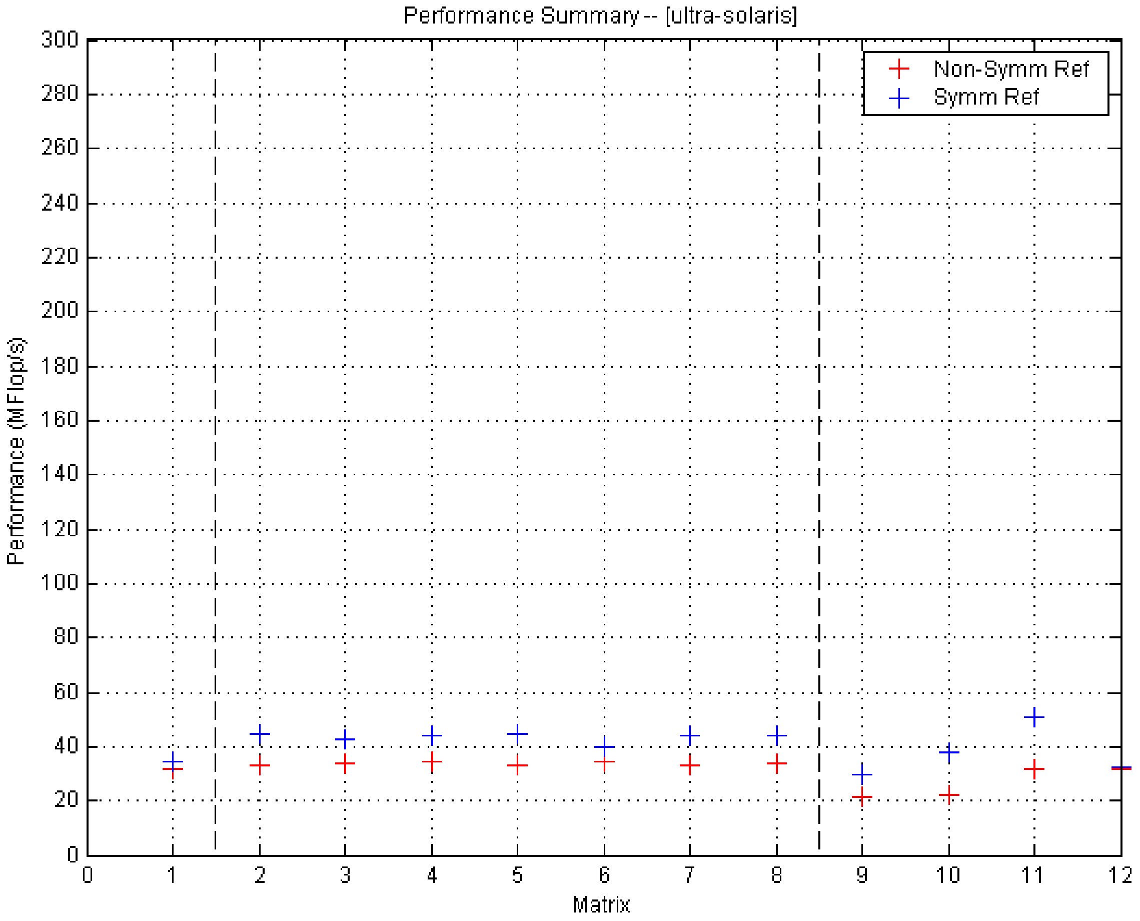
Evaluation: Methodology

- Three Platforms
  - Sun Ultra 2i, Intel Itanium 2, IBM Power 4
- Matrix Test Suite
  - Twelve matrices
  - Dense, Finite Element, Linear Programming, Assorted
- Reference Implementation
  - No symmetry, no register blocking, single vector multiplication
- Tuning Parameters
  - SpMM code characterized by parameters (r , c , v)
    - Register block size : r x c
    - Vector width : v

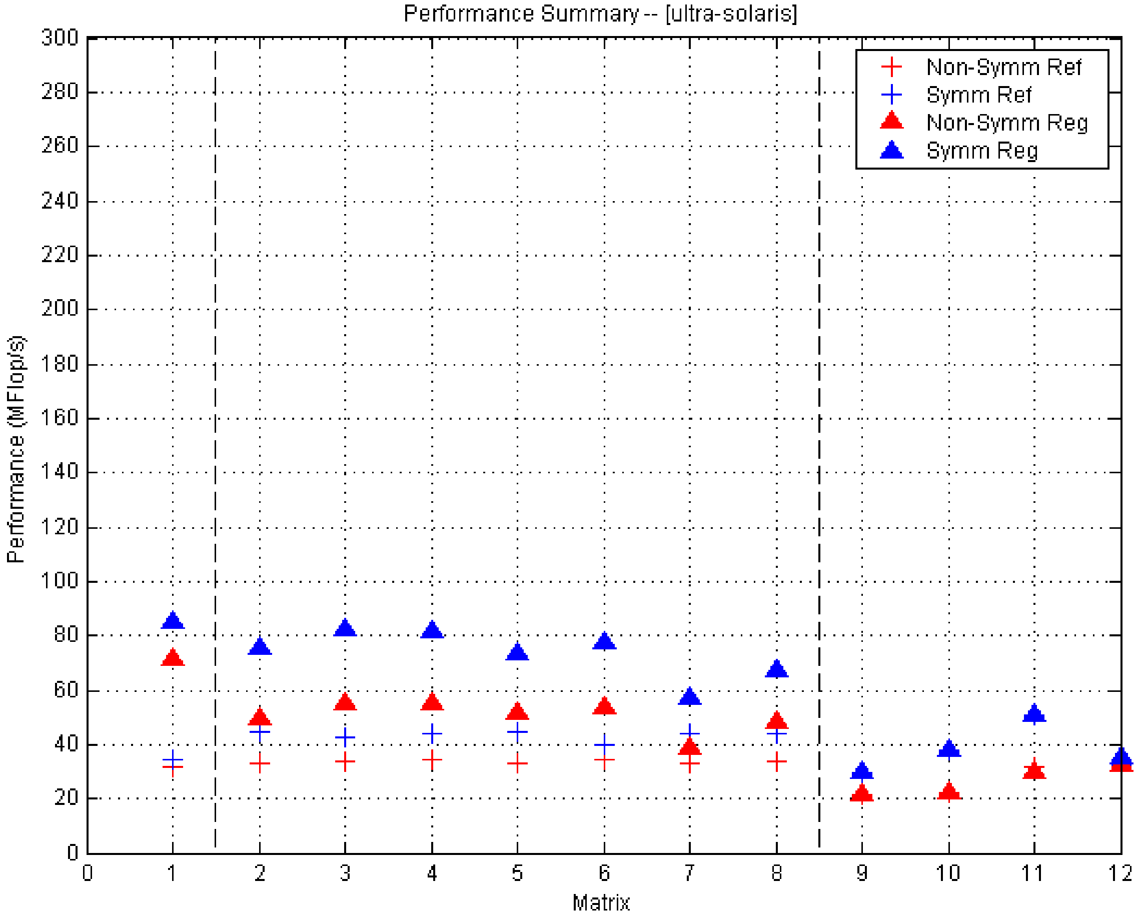
Evaluation: Exhaustive Search

- Performance
  - 2.1x max speedup (1.4x median) from symmetry (SpMV)
    - {Symm BCSR Single Vector} vs {Non-Symm BCSR Single Vector}
  - 2.6x max speedup (1.1x median) from symmetry (SpMM)
    - {Symm BCSR Multiple Vector} vs {Non-Symm BCSR Multiple Vector}
  - 7.3x max speedup (4.2x median) from combined optimizations
    - {Symm BCSR Multiple Vector} vs {Non-Symm CSR Single Vector}
- Storage
  - 64.7% max savings (56.5% median) 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

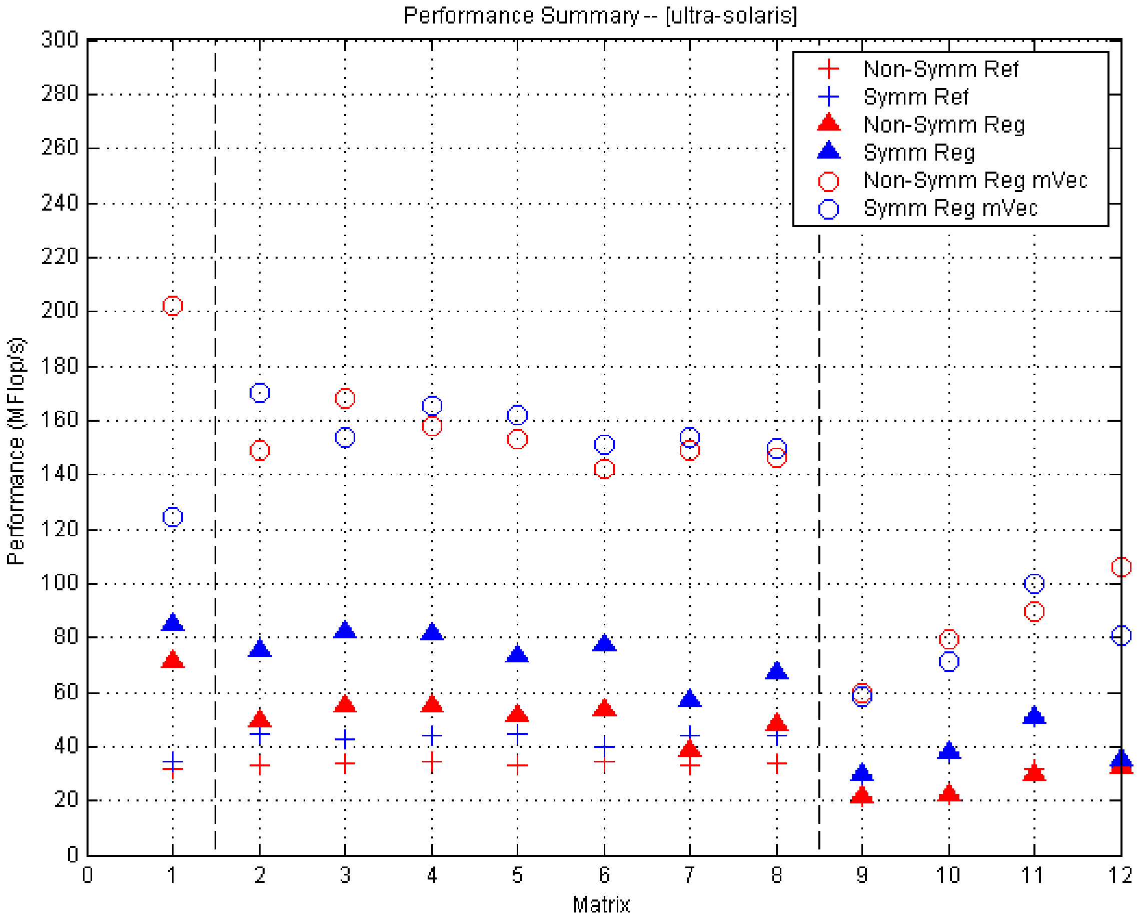
Performance Results: Sun Ultra 2i



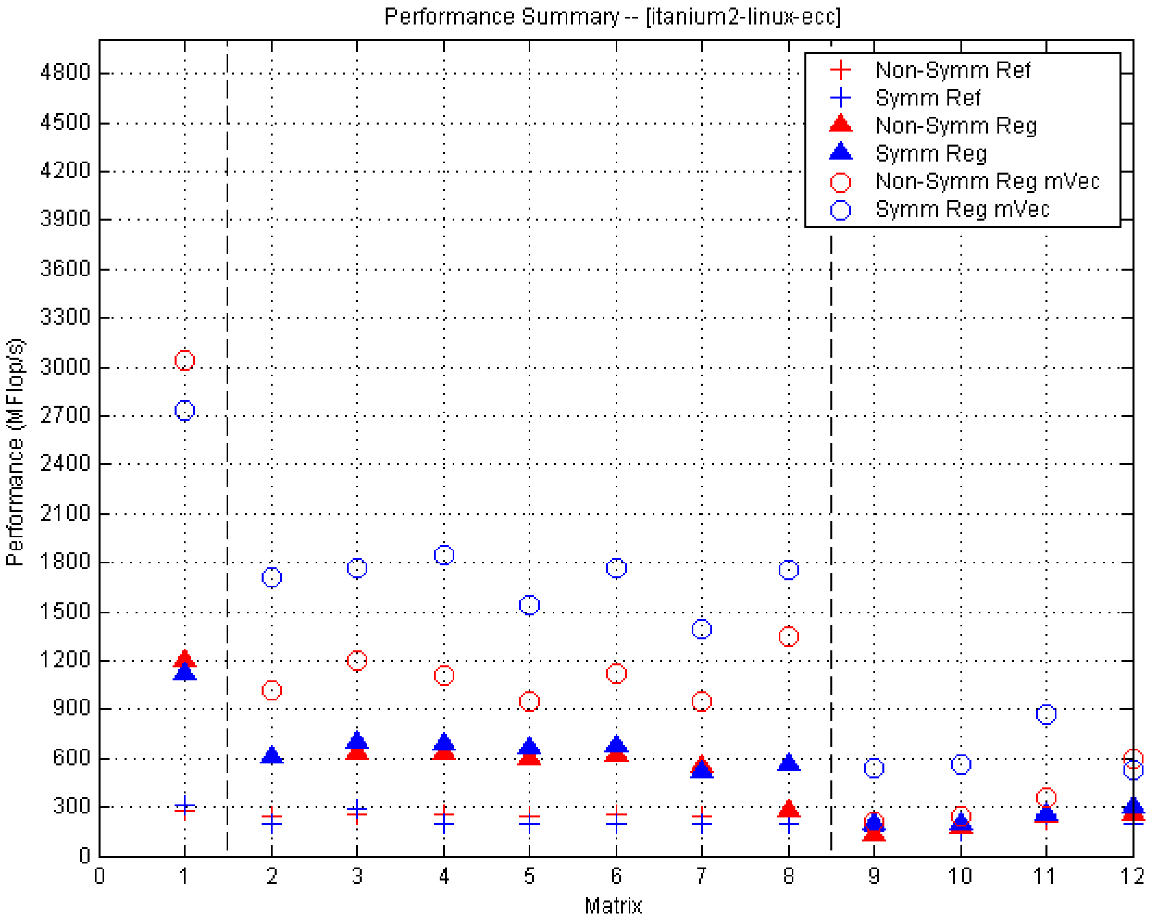
Performance Results: Sun Ultra 2i



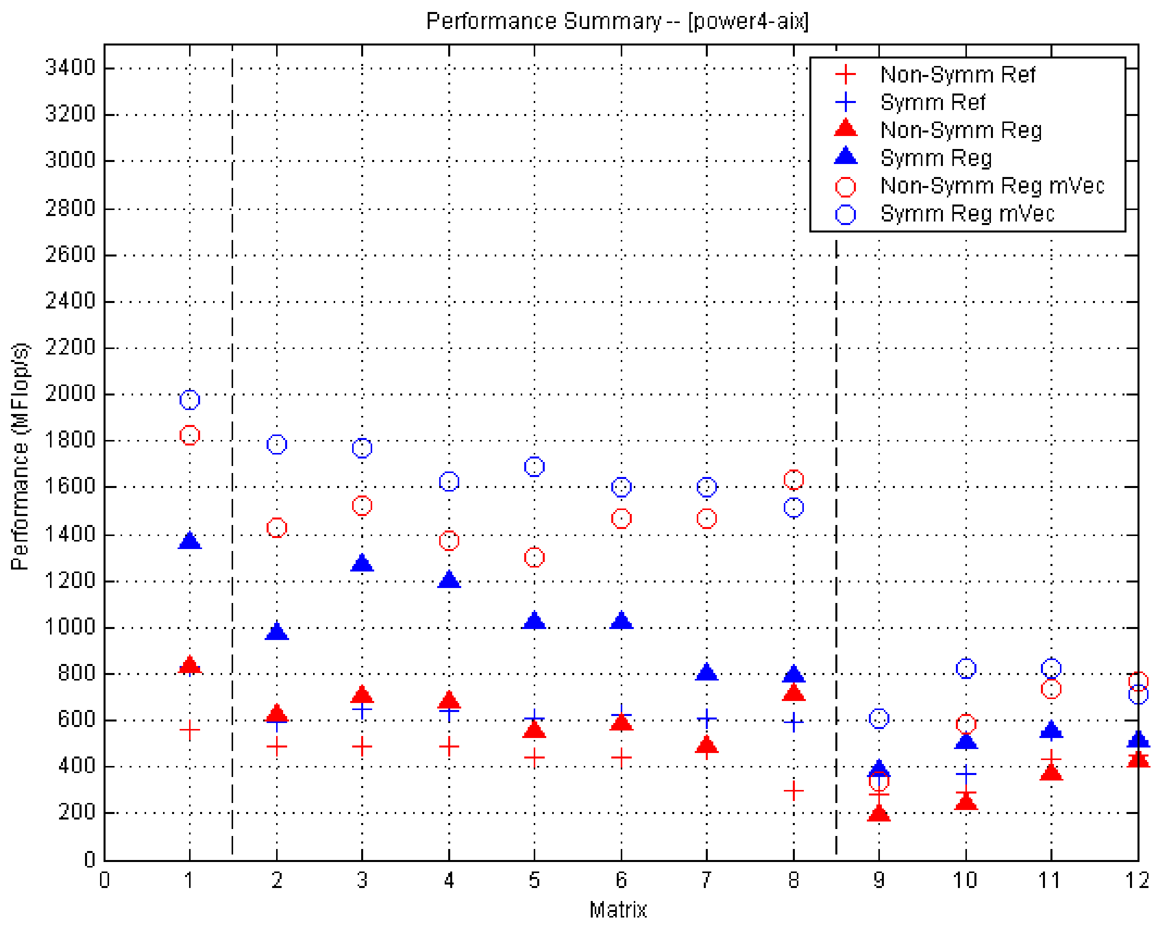
Performance Results: Sun Ultra 2i



Performance Results: Intel Itanium 2



Performance Results: IBM Power 4



Automated Empirical Tuning

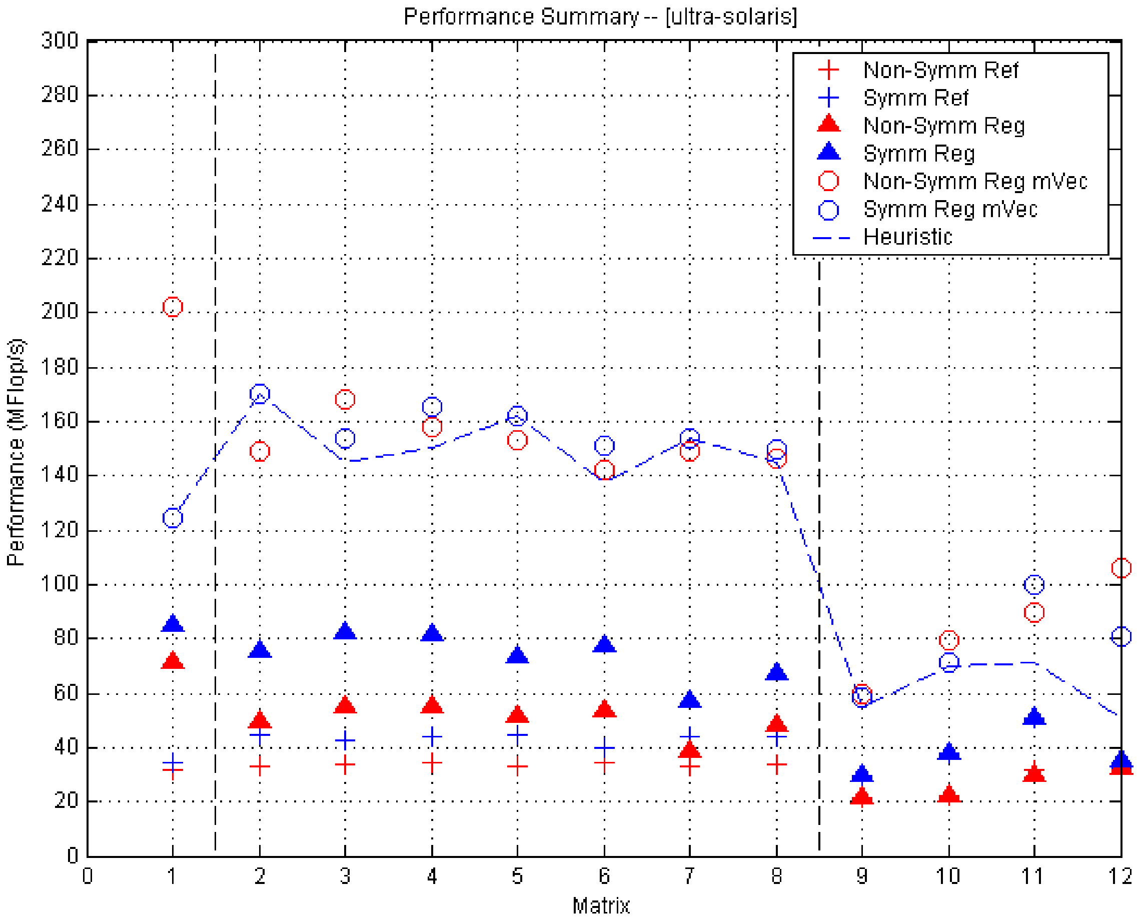
- Exhaustive search infeasible
  - Cost of matrix conversion to blocked format

- Parameter Selection Procedure
  - Off-line benchmark
    - Symmetric SpMM performance for dense matrix D in sparse format
    - $\{ P_{rcv}(D) \mid 1 \le r,c \le b_{max} \text{ and } 1 \le v \le v_{max} \}$, Mflop/s
  - Run-time estimate of fill
    - Fill is number of stored values divided by number of original non-zeros
    - $\{ f_{rc}(A) \mid 1 \le r,c \le b_{max} \}$, always at least 1.0
  - Heuristic performance model
    - Choose (r , c , v) to maximize estimate of optimized performance
    - $max_{rcv} \{ P_{rcv}(A) = P_{rcv}(D) / f_{rc}(A) \mid 1 \le r,c \le b_{max} \text{ and } 1 \le v \le \min(v_{max} , k) \}$

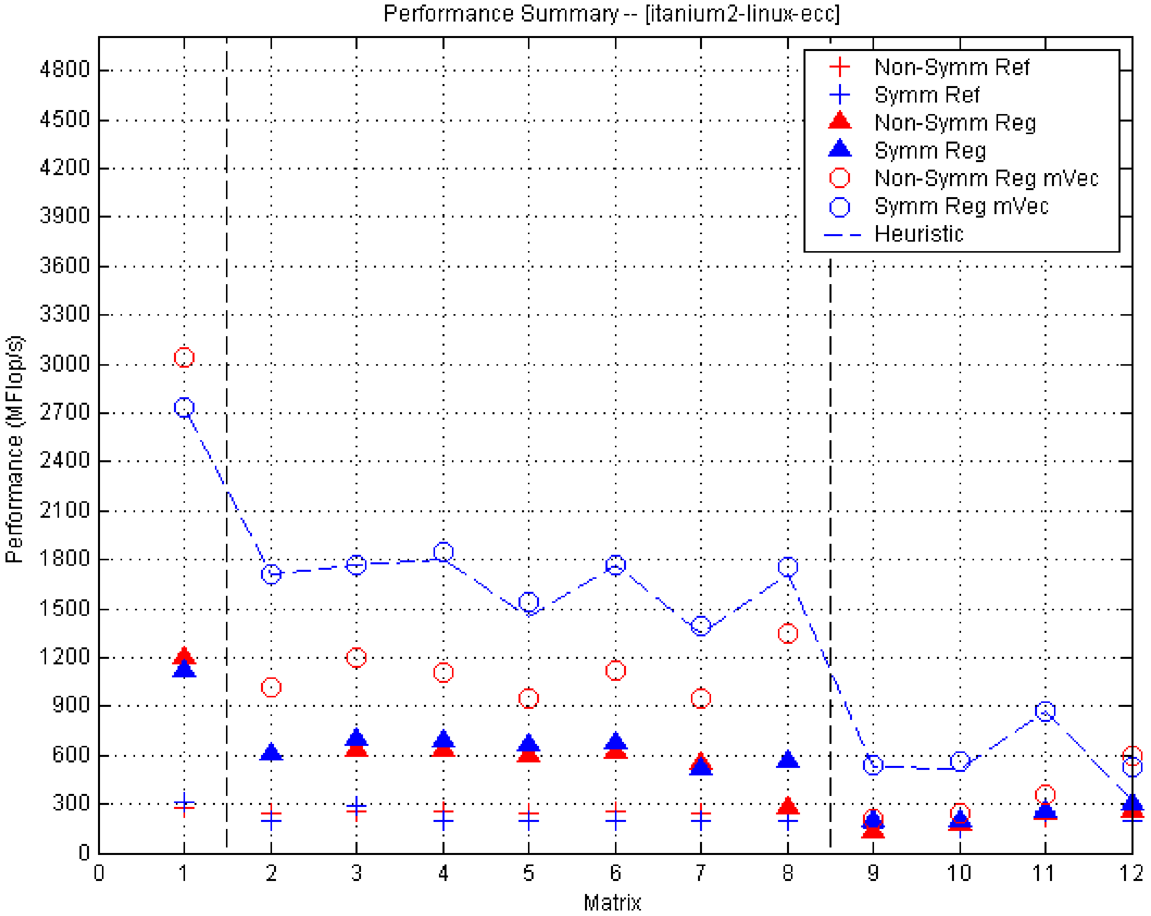
Evaluation: Heuristic Search

- Heuristic Performance
  - Always achieves at least 93% of best performance from exhaustive search
    - Ultra 2i, Itanium 2
  - Always achieves at least 85% of best performance from exhaustive search
    - Power 4

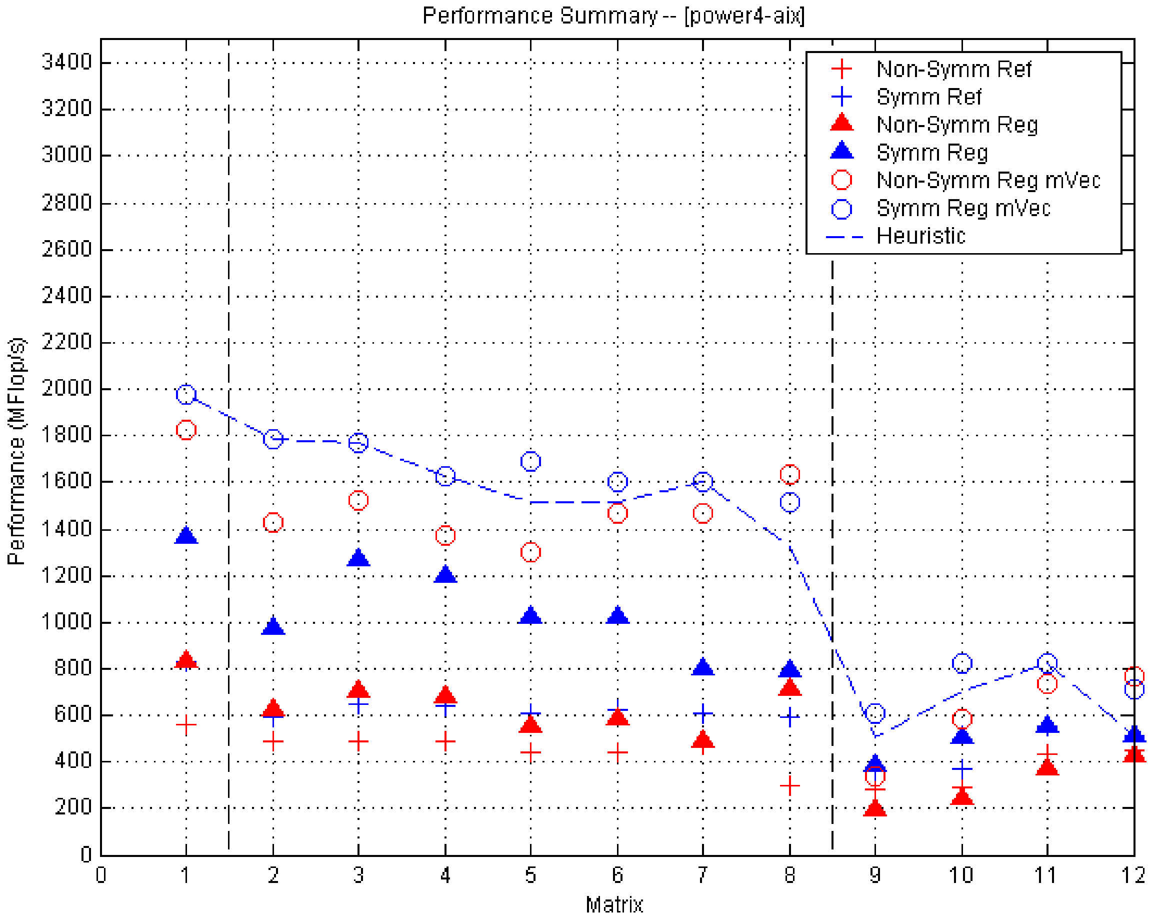
Performance Results: Sun Ultra 2i



Performance Results: Intel Itanium 2



Performance Results: IBM Power 4



Performance Models

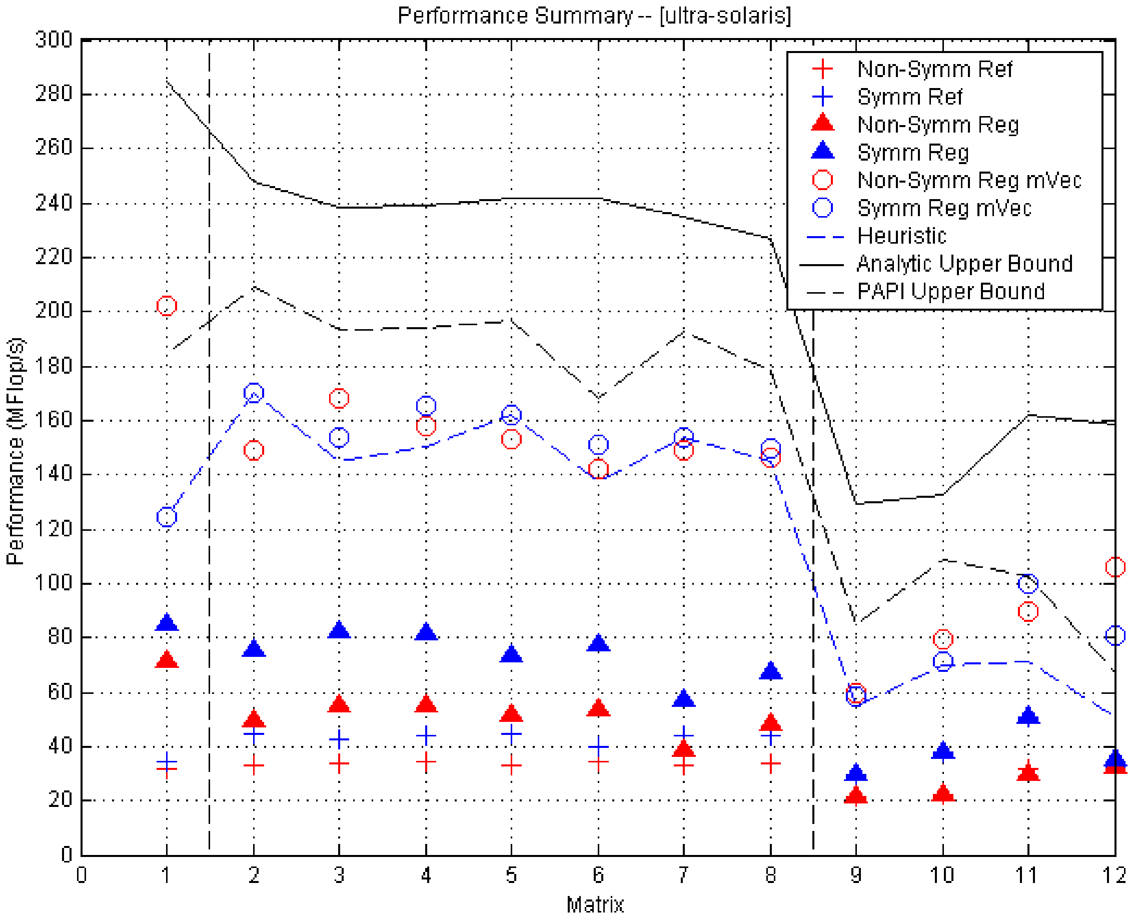
- Model Characteristics and Assumptions
  - Considers only the cost of memory operations
  - Accounts for minimum effective cache and memory latencies
  - Considers only compulsory misses (***i.e.***, ignore conflict misses)
  - Ignores TLB misses

- Execution Time Model
  - Loads and cache misses
    - Analytic model (based on data access patterns)
    - Hardware counters (via PAPI)
  - Charge a_i for hits at each cache level
    - $T = (L1\ hits)\ a_1 + (L2\ hits)\ a_2 + (Mem\ hits)\ a_{mem}$
    - $T = (Loads)\ a_1 + (L1\ misses)\ (a_2 - a_1) + (L2\ misses)\ (a_{mem} - a_2)$

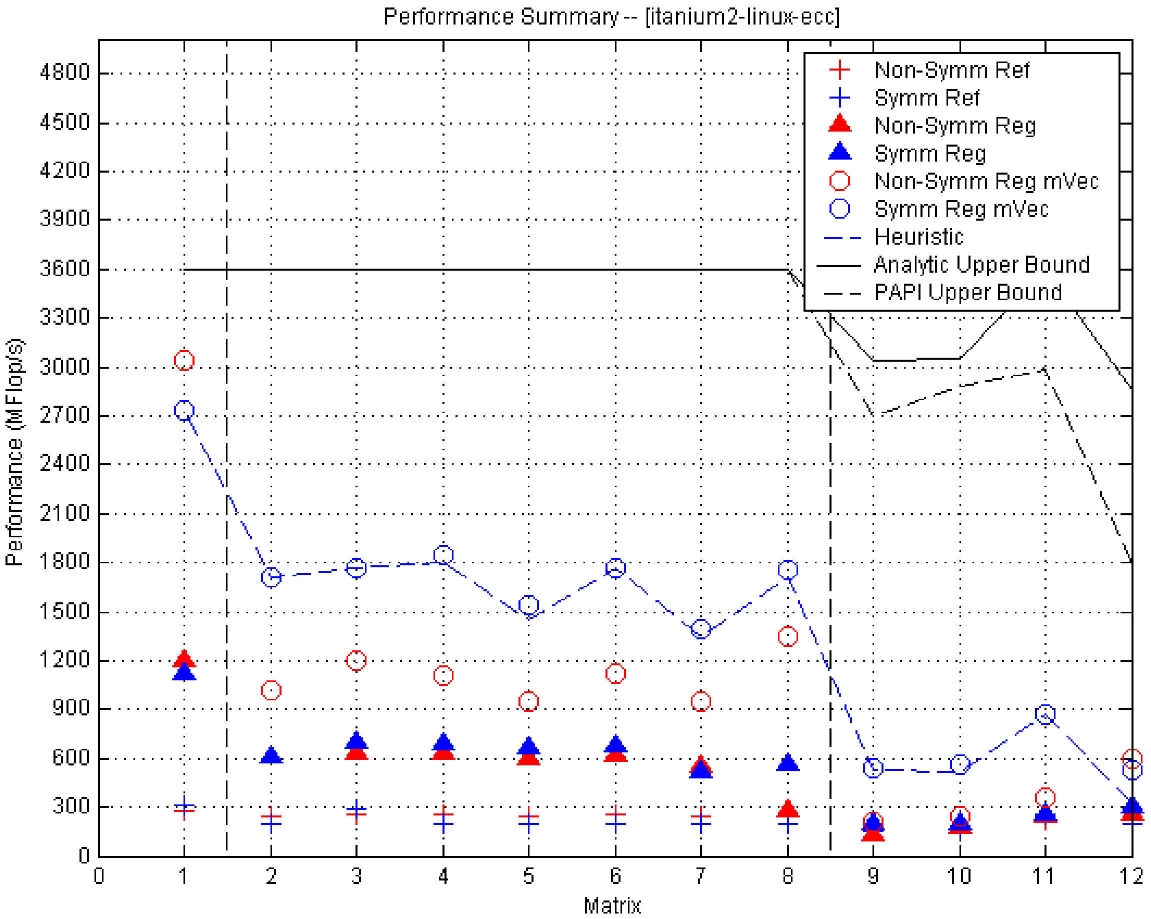
Evaluation: Performance Bounds

- Measured Performance vs. PAPI Bound
  - Measured performance is 68% of PAPI bound, on average
  - FEM applications are closer to bound than non-FEM matrices

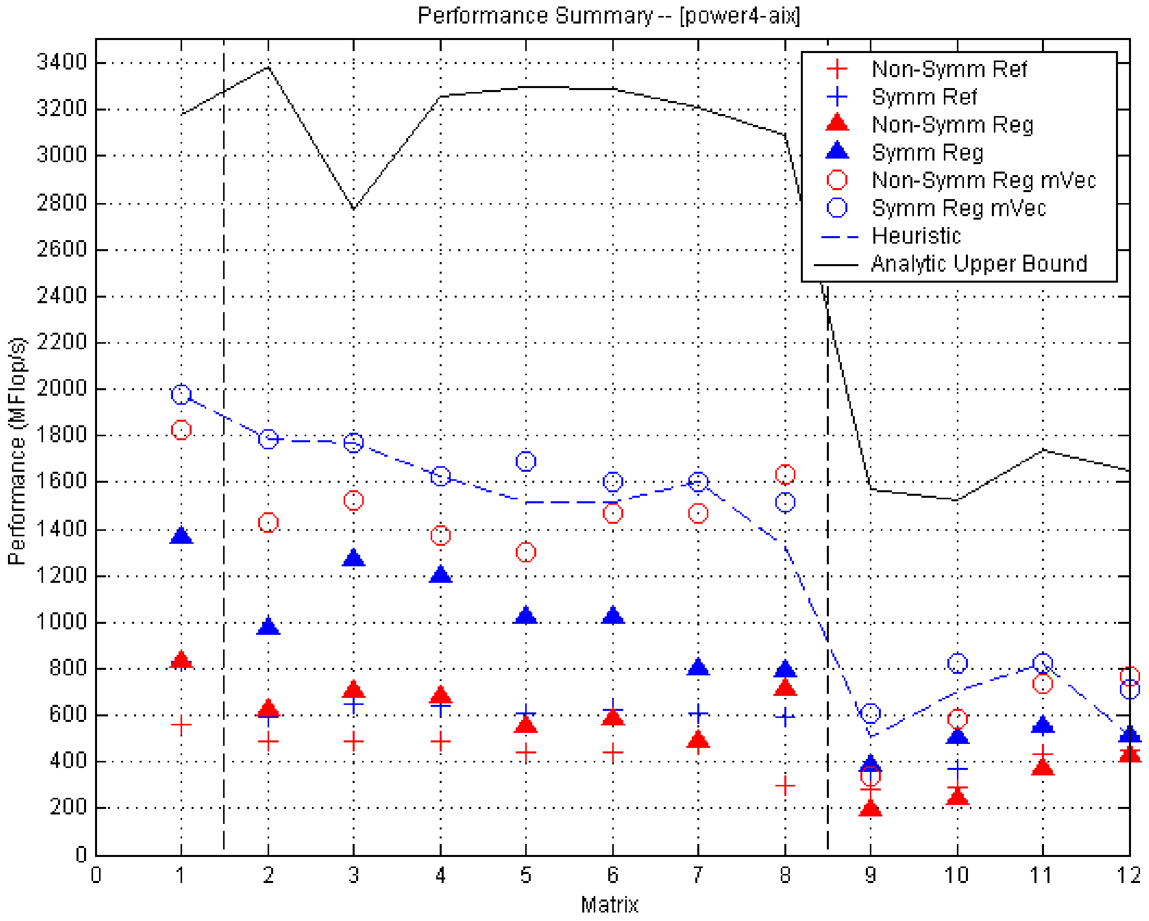
Performance Results: Sun Ultra 2i



Performance Results: Intel Itanium 2



Performance Results: IBM Power 4



Conclusions

- Matrix Symmetry Optimizations
  - Symmetric Performance: 2.6x speedup (1.1x median)
  - Overall Performance: 7.3x speedup (4.15x median)
  - Symmetric Storage: 64.7% savings (56.5% median)
  - Cumulative performance effects
- Automated Empirical Tuning
  - Always achieves at least 85-93% of best performance from exhaustive search
- Performance Modeling
  - Models account for symmetry, register blocking, multiple vectors
  - Measured performance is 68% of predicted performance (PAPI)

Current & Future Directions

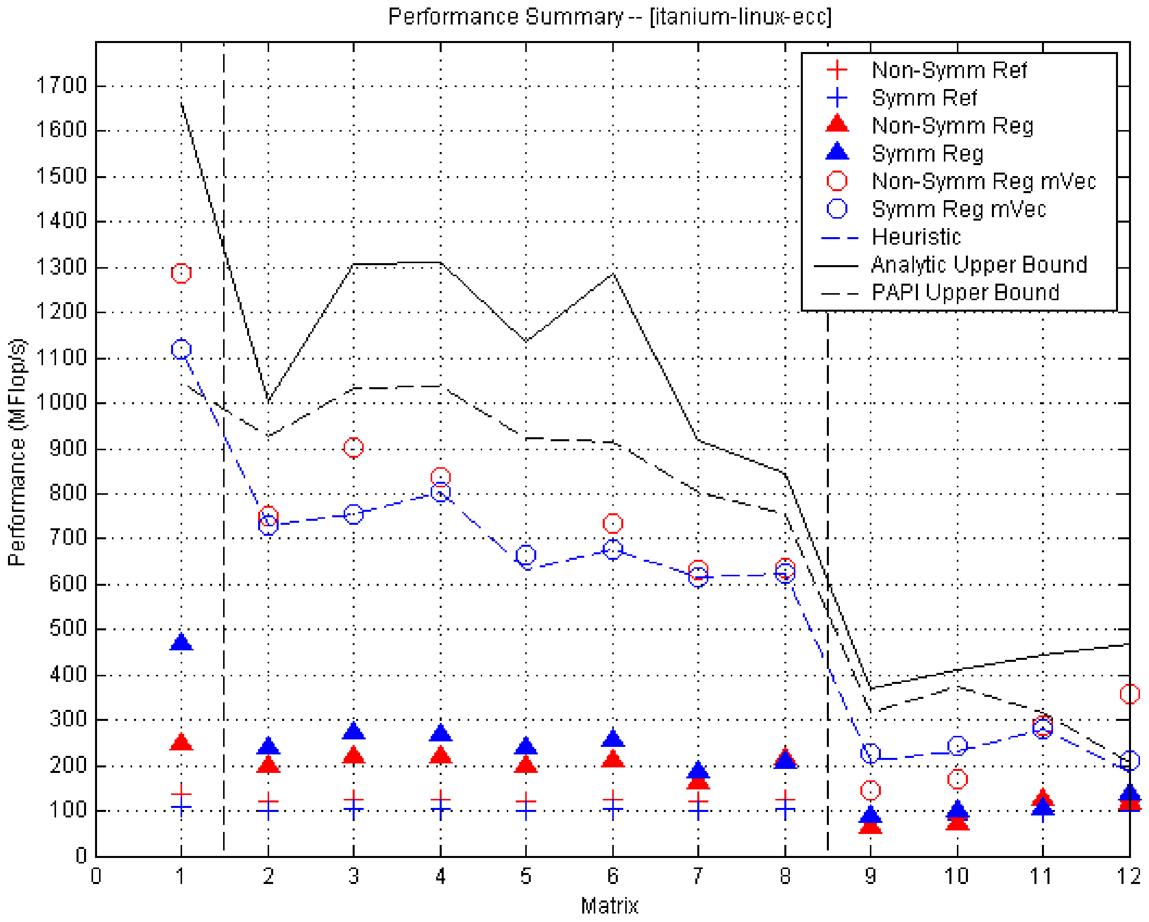
- Parallel SMP Kernels
  - Multi-threaded versions of optimizations
  - Extend performance models to SMP architectures
- Self-Adapting Sparse Kernel Interface
  - Provides low-level BLAS-like primitives
  - Hides complexity of kernel-, matrix-, and machine-specific tuning
  - Provides new locality-aware kernels

Appendices

- Berkeley Benchmarking and Optimization Group
  - http://bebop.cs.berkeley.edu
- Conference Paper: "Performance Models for Evaluation and Automatic Tuning of Symmetric Sparse Matrix-Vector Multiply"
  - http://www.cs.berkeley.edu/~blee20/publications/lee2004-icpp-symm.pdf
- Technical Report: "Performance Optimizations and Bounds for Sparse Symmetric Matrix-Multiple Vector Multiply"
  - http://www.cs.berkeley.edu/~blee20/publications/lee2003-tech-symm.pdf

Appendices

Performance Results: Intel Itanium 1


Performance Summary -- [itanium-linux-ecc]
Performance (MFlop/s)
Matrix
Non-Symm Ref
Symm Ref
Non-Symm Reg
Symm Reg
Non-Symm Reg mVec
Symm Reg mVec
Heuristic
Analytic Upper Bound
PAPI Upper Bound