Combining Models and Guided Empirical Search to Optimize for Multiple Levels of the Memory Hierarchy

Chun Chen

Jacqueline Chame

Mary Hall

Information Sciences Institute

University of Southern California

LACSI Workshop
Oct. 11, 2005

USC VITERBI SCHOOL OF ENGINEERING

Trade-offs in the Memory Hierarchy

- **The best performance comes from balancing all optimization goals**
  - Register loads/stores
  - L1 cache misses
  - L2 cache misses
  - TLB misses
  - Prefetching instructions
  - Instruction scheduling
- **Hard problem**
  - Complex interaction
  - e.g. Matrix Multply: Well studied, but still need hand-tuning for best performance

Current Approaches to Performance Tuning

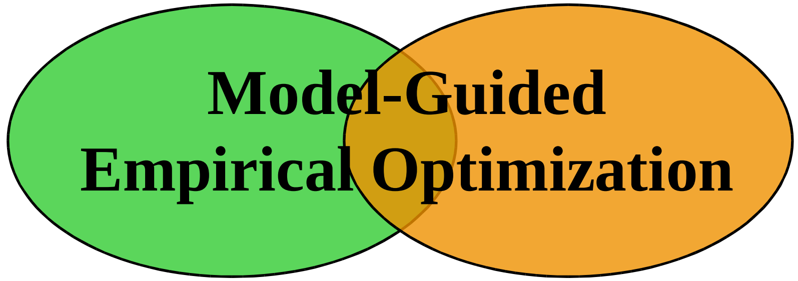
Model-Guided Optimization

Empirical Optimization

- **Model-Guided Optimization**
  - Optimization decisions are based on static models of architecture and optimization impact
  - Optimizations are often performed in isolation and in a fixed order
- **Empirical Optimization**
  - Optimization decisions are guided by feedback from executing actual code segments on target machine
  - Examples: self-tuning libraries (ATLAS, PhiPAC, FFTW etc.)

USC VITERBI SCHOOL OF ENGINEERING

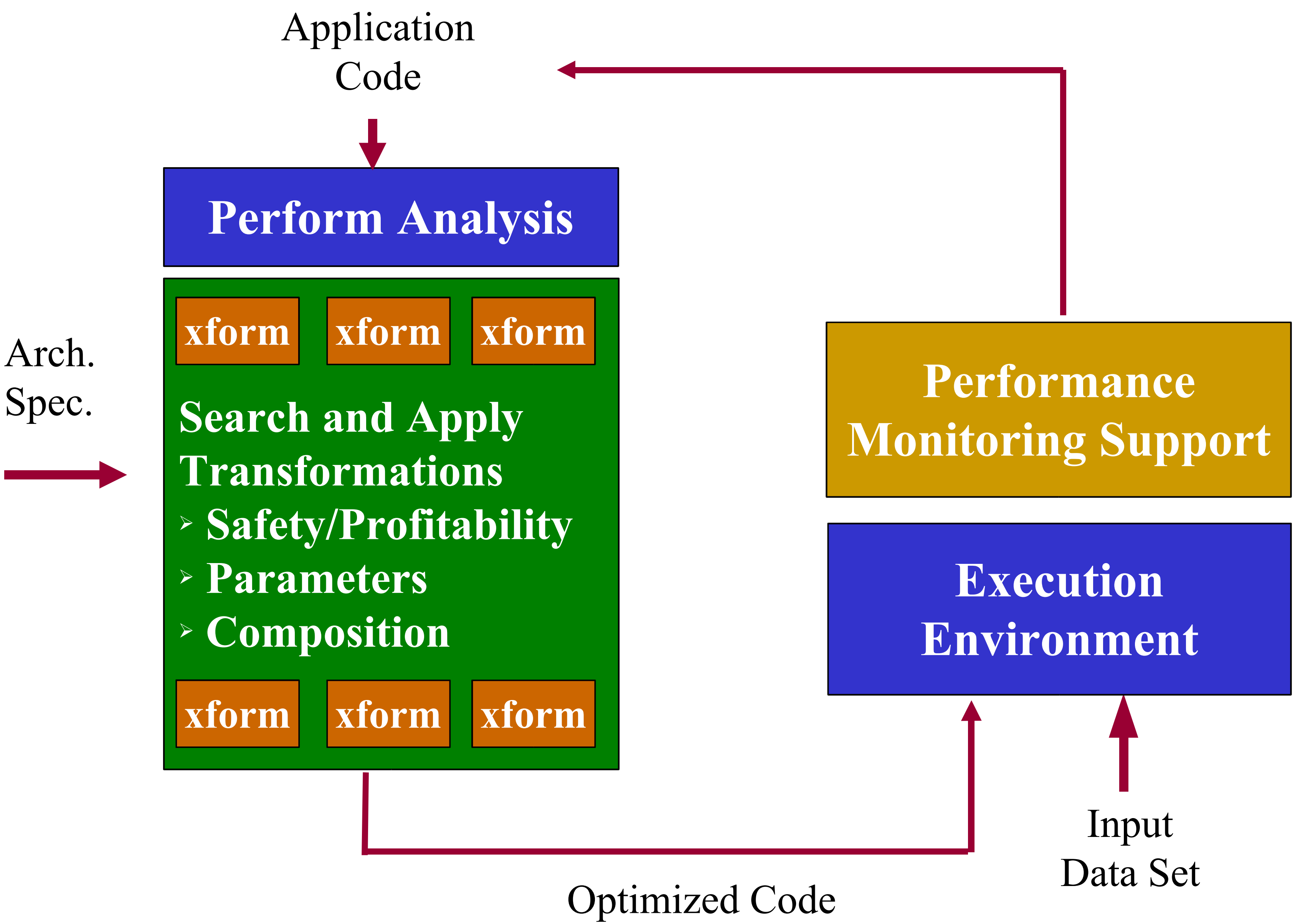
Model-Guided Empirical Optimization



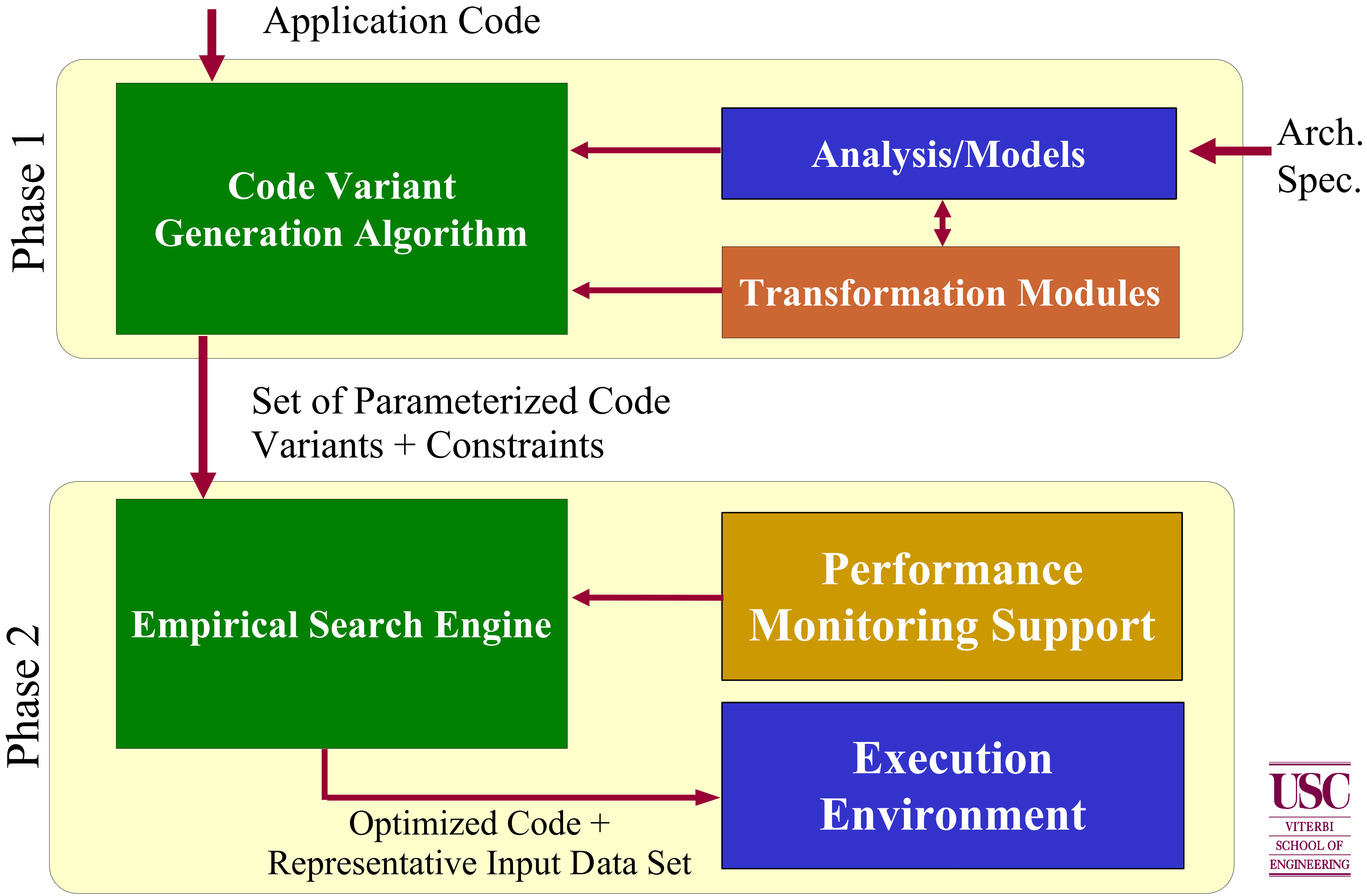
- **Goal:**
  - Compiler derived but with the performance of hand-tuned versions
  - Increase *machine* and *programmer* efficiencies
- **Exploit complementary strengths of both approaches**
  - Compiler models prune from search space unprofitable solutions
  - Empirical data provide accurate measure of optimization impact
- **Key Concepts**
  - Select among implementation ***variants*** of the same computation
  - Derive integer values of optimization ***parameters***
  - Only search promising code variants and a restricted parameter space

USC VITERBI SCHOOL OF ENGINEERING

Today's Development Tools

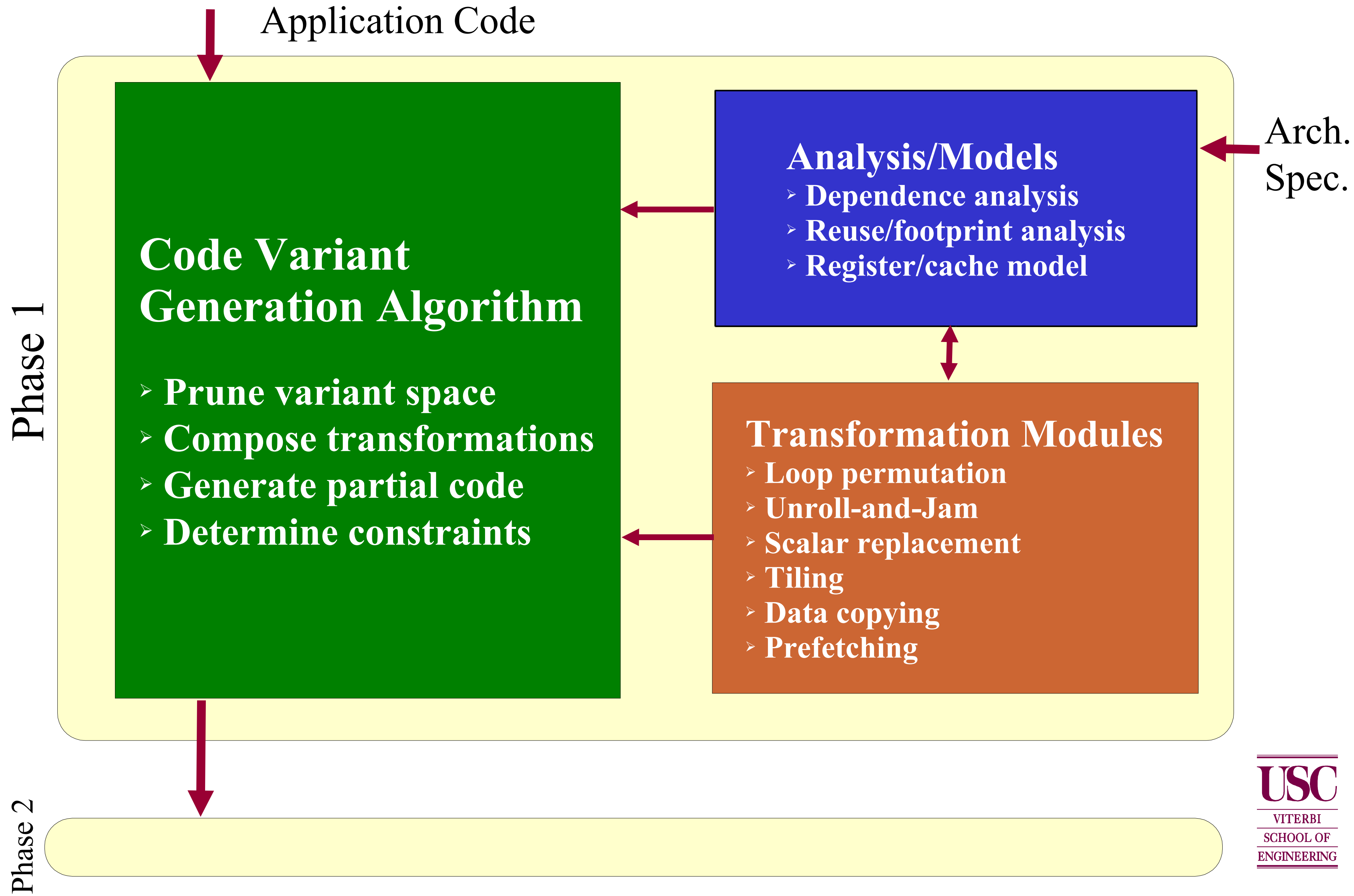


Our Development Tool Strategy



USC VITERBI SCHOOL OF ENGINEERING

Phase 1: Code Variant Generation



USC VITERBI SCHOOL OF ENGINEERING

Transformation Variants and Parameters

Transformations	Definition	Goal	Variants	Parameters
Loop permutation	Change the loop order	Enable U&J and Tiling + Reduce TLB misses	Different loop orders	-
Unroll and Jam	Unroll outer loops and fuse inner loops	Reuse in registers	-	Unroll factors
Scalar replacement	Replace array accesses with scalar variables			
Tiling	Divide iteration space into tiles	Reuse in cache	-	Tile sizes
Data copying (w/ tiling)	Copy subarray into contiguous memory space	Avoid conflict misses + Avoid TLB thrashing	Yes/no on specific data structures	-
Prefetching	Prefetch data into cache before actual references	Hide memory latency	-	Prefetch distances

- All loops are unrolled and tiled and all data are prefetched.
- For degenerate cases, Unroll factor=1, Tile size=1 and Prefetch distance=0, code transformations are not applied.

USC VITERBI SCHOOL OF ENGINEERING

Code Variant Generation Algorithm

- **Key Insights:**
  - Target data structures to specific levels of the memory hierarchy based on reuse analysis
  - Compose code transformations and determine constraints

For each memory hierarchy level in (Register, L1, L2, ...), using models to

1. Select the data structure D which has maximum reuse from reuse analysis (if possible, one that has not been considered)

2. Permute the relevant loops and apply tiling (unroll-and-jam for registers) according to newly selected reuse dimension

3. Generate copy variant if copying is beneficial

4. Determine constraints based on D and current memory hierarchy level characteristics, using register/cache/TLB footprint analysis

5. Mark D as considered

Transformations of Matrix Multiply

Transformations	Variants	Parameters
Loop permutation	IJK(original), IKJ,JIK,JKI,KJI,KIJ	
Unroll and Jam		UI, UJ, UK
Scalar replacement		
Tiling		TI, TJ, TK
Data copying (w/ tiling)	Copy A? Copy B? Copy C?	
Prefetching		PA, PB, PC

```
DO I = 1, N
 DO J = 1, N
  DO K = 1, N
    C[I,J]= C[I,J]+A[I,K]*B[K,J]
```

Transformations of Matrix Multiply

C has most reuse
Make K outermost loop

Transformations	Variants	Parameters
Loop permutation	~~IJK(original)~~, ~~IKJ,JIK,JKI~~,KJI,KIJ	
Unroll and Jam		UI, UJ, UK
Scalar replacement		
Tiling		TI, TJ, TK
Data copying (w/ tiling)	Copy A? Copy B? Copy C?	
Prefetching		PA, PB, PC

USC VITERBI SCHOOL OF ENGINEERING

Transformations of Matrix Multiply

Transformations	Variants	Parameters
Loop permutation	~~IJK(original)~~, ~~IKJ,JIK,JKI~~,KJI,KIJ	
Unroll and Jam		UI*UJ<= 32, UK=1 (no unrolling)
Scalar replacement		
Tiling		TI, TJ, TK
Data copying (w/ tiling)	Copy A? Copy B? ~~Copy C?~~	
Prefetching		PA, PB, PC=0

Unroll-and-Jam I and J in registers

C in registers
No copy or prefetch

VITERBI SCHOOL OF ENGINEERING

Transformations of Matrix Multiply

Transformations	Variants	Parameters
Loop permutation	~~IJK(original)~~, ~~IKJ,JIK,JKI~~,KJI,KIJ	
Unroll and Jam		UI*UJ<= 32, UK=1 (no unrolling)
Scalar replacement		
Tiling		TI*TK<=size(L1), TJ=1 (no tiling)
Data copying (w/ tiling)	Copy A, Copy B? ~~Copy C?~~	
Prefetching		PA, PB, PC=0

A has next most reuse
Tile I and K to reuse
A in L1 cache

Copy A to reduce
conflict misses

USC
VITERBI
SCHOOL OF
ENGINEERING

Transformations of Matrix Multiply

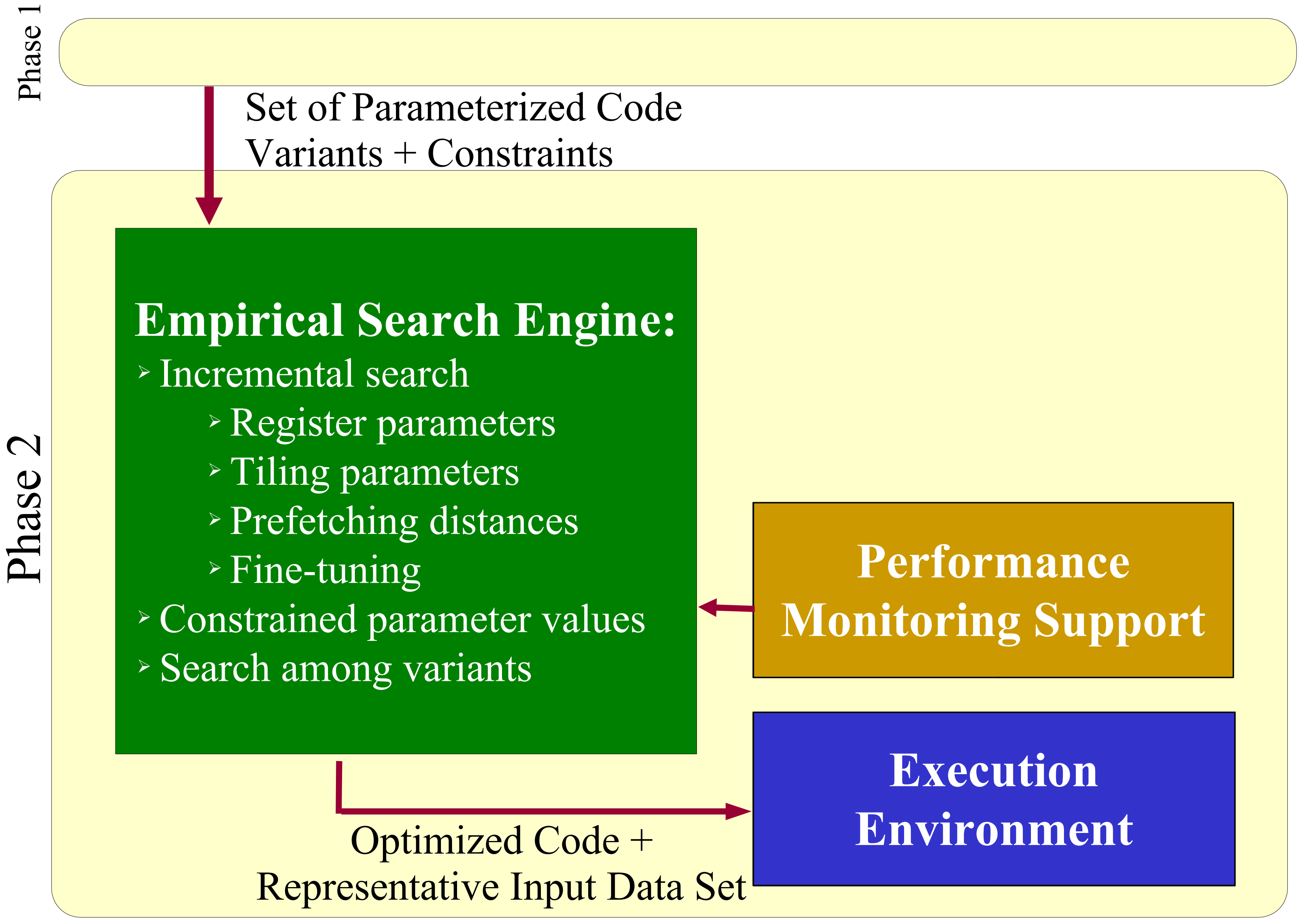
Transformations	Variants	Parameters
Loop permutation	~~IJK(original),~~ ~~IKJ,JIK,JKI,~~KJI,KIJ	
Unroll and Jam		UI*UJ<= 32, UK=1 (no unrolling)
Scalar replacement		
Tiling		TI*TK<=size(L1), TK*TJ<=size(L2)
Data copying (w/ tiling)	Copy A, Copy B, ~~Copy C?~~	
Prefetching		PA, PB, PC=0

B has next most reuse
Further tile J to reuse
B in L2 cache

Copy B to reduce
conflict misses

USC
VITERBI
SCHOOL OF
ENGINEERING

Phase 2: Empirical Search



USC VITERBI SCHOOL OF ENGINEERING

Search Space

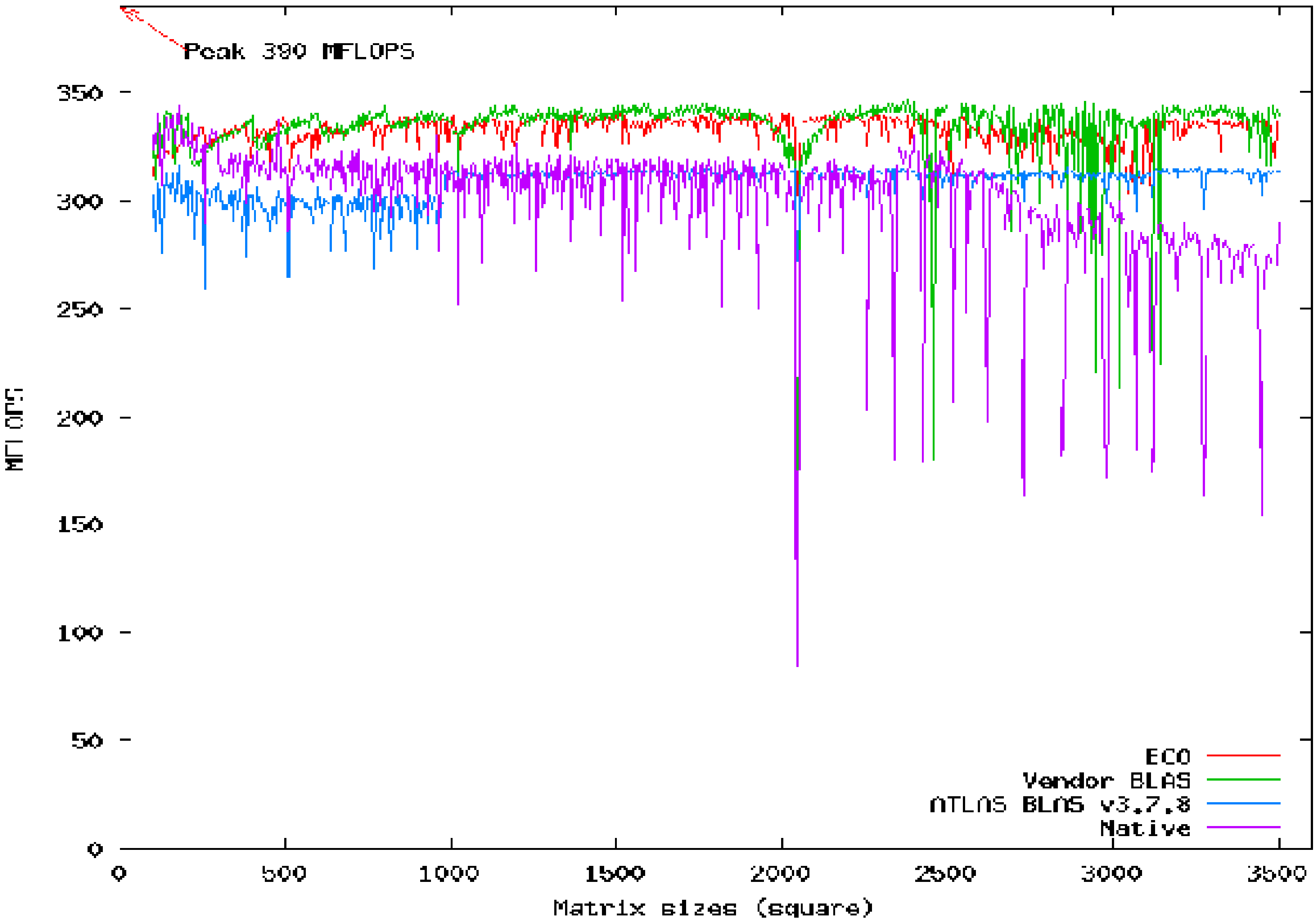
- **Set of variants**
  - Different loop orders, copy yes or no
  - Select variant with the best performance
- **Integer parameter values**
  - Unroll factors, tile sizes, prefetch distances
  - Each parameter has unique search properties
- **Constraints:**
  - Limit unrolling amount by register capacity
  - Limit tiling parameters by cache/TLB capacity and set associativity

Experimental Results

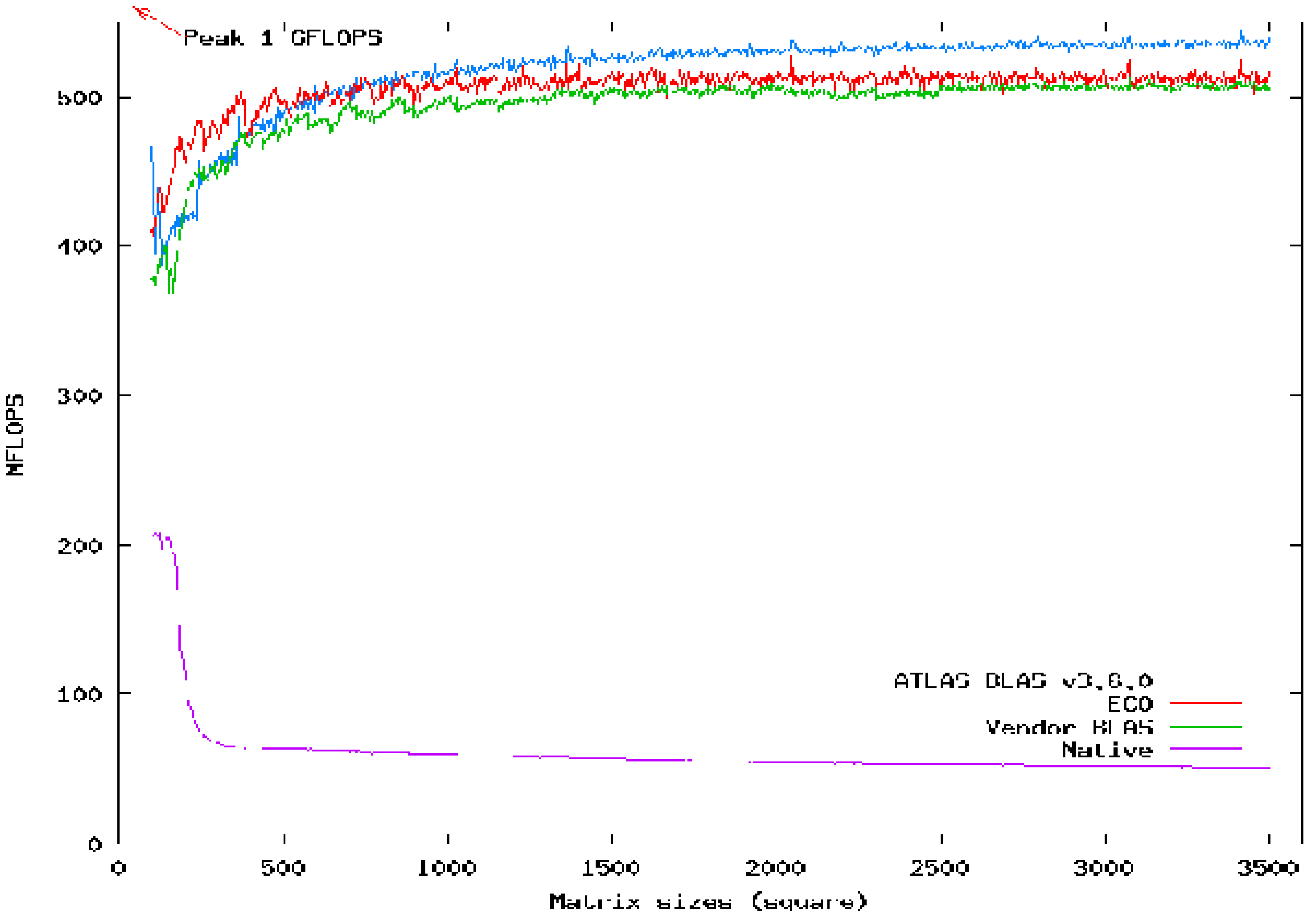
- **Implementation based on SUIF**
- **Computational kernels:**
  - Matrix Multiply
  - 3D Jacobi
- **Architectures:**
  - SGI R10K
  - Sun UltraSparc IIe
- **Comparison**
  - Native Compiler: using the best optimization level possible
  - ECO: implementation of our framework
  - ATLAS: a self tuning linear algebra library
  - Vendor BLAS: vendor provided hand-tuned library

USC VITERBI SCHOOL OF ENGINEERING

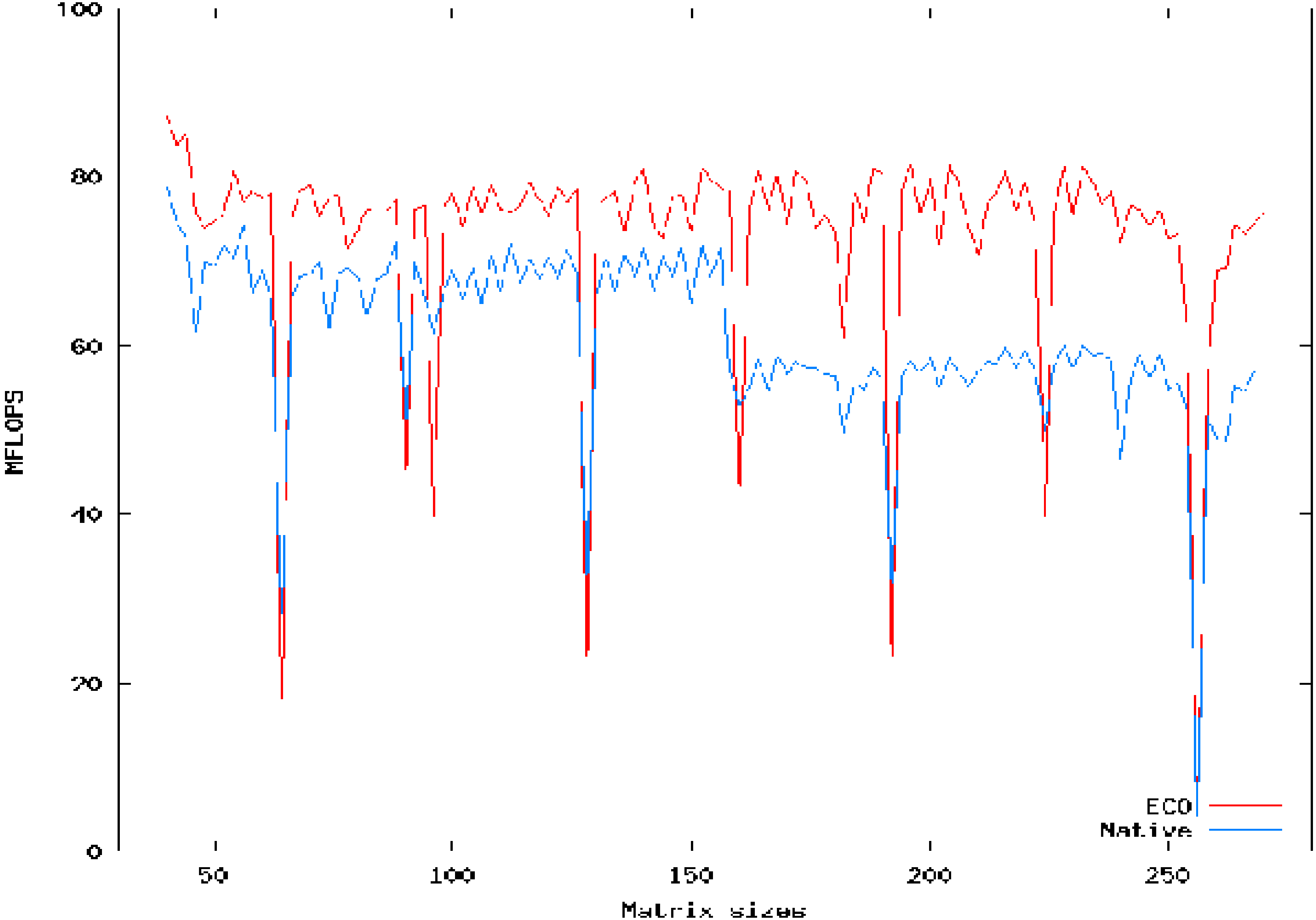
MM Performance Results, SGI R10K



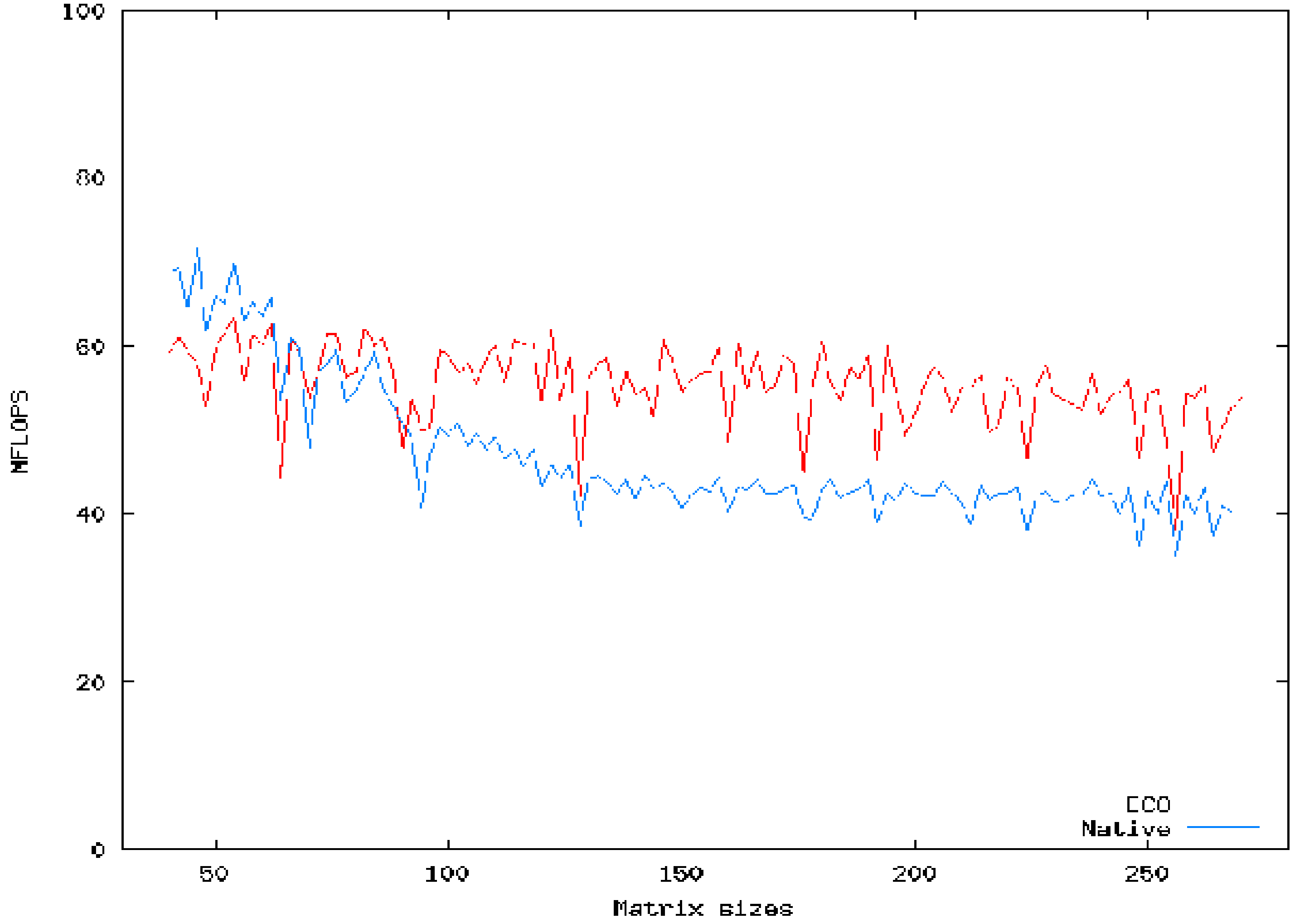
MM Performance Results, Sun US-2e



Jacobi Performance Results, SGI R10K



Jacobi Performance Results, Sun US-2e



Comparison of Search Cost

Code	SGI R10K	Sun US-2e
MM (ATLAS)	35 min	14 min
MM (ECO)	8 min (60 pts)	6 min (44 pts)
Jacobi (ECO)	3 min (94 pts)	5 min (148 pts)

Conclusion

- **Optimizing for multiple levels of the memory hierarchy is the key to high performance**
  - Combines static models and empirical search
- **Prunes search space**
  - Uses the combined knowledge from analyses, application and architecture
- **Performance from initial implementation**
  - Comparable or better than hand-tuned codes
  - Comparable or better than self-tuning libraries
  - Substantially outperforms native compilers

Future Work

- Extend compiler framework to imperfect loop nests and multiple loop nests (e.g., LU and composing BLAS routines)
- Systematic optimization: Apply search techniques from machine learning and derive a knowledge representation
- Combine compiler-guided and user-guided performance tuning (molecular dynamics, mixed dense/sparse codes, signal processing)

Relevant Publications

- **Combining Models and Guided Empirical Search to Optimize for Multiple Levels of the Memory Hierarchy,** by C. Chen, J. Chame and M. Hall. In *Proceedings of the Conference on Code Generation and Optimization,* March, 2005.
- **Empirical Optimization for a Sparse Linear Solver: A Case Study,** by Y. Lee, P. Diniz, M. Hall and R. Lucas. In *International Journal of Parallel Programming*, 2005.
- **A Code Isolator: Isolating Code Fragments from Large Programs,** by Y. Lee and M. Hall. In *Proceedings of the Workshop on Languages, Compilers for Parallel Computing, September*, 2004.
- **A Systematic Approach to Composing and Optimizing Application Workflows,** by E. Deelman, A. Galstyan, Y. Gil, M. Hall, K. Lerman, A. Nakano, P. Vashista, J. Saltz, In *Workshop on Patterns in High Performance Computing*, Urbana-Champaign, May, 2005
- **A Systematic Approach to Model-Guided Empirical Search for Memory Hierarchy Optimization,** by C. Chen, J. Chame, M. Hall, K. Lerman, In *Proceedings of the Workshop on Languages, Compilers for Parallel Computing*, October, 2005

USC VITERBI SCHOOL OF ENGINEERING