
An Experimental Comparison of Empirical and Model-based Optimization

Maria Jesus Garzaran

University of Illinois at Urbana-Champaign

Joint work with:

Kamen Yotov², Xiaoming Li¹, Gang Ren¹, Michael Cibulskis¹,
Gerald DeJong¹, David Padua¹,
Keshav Pingali², Paul Stodghill², Peng Wu³

¹UIUC, ²Cornell University, ³IBM T.J.Watson

Motivation

- **Autonomic computing**
 - self-configuration
 - self-optimization
 - self-x
 - **Question**
 - **How important is empirical search?**
 - **Model-based vs. empirical optimization**
 - Not in conflict!
 - Use models to prune search space
 - Search → **Intelligent Search**
 - Use empirical observations to refine models
 - Learn from experience
-

Optimization: Compilers vs. Library Generators

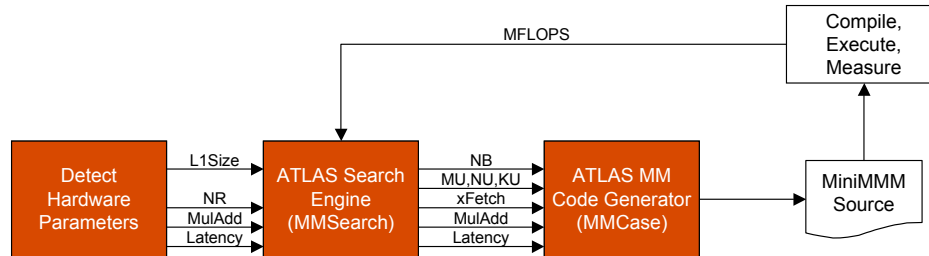
- | | |
|------------------------------------|---|
| ■ Traditional Compilers | ■ ATLAS / FFTW / Spiral |
| □ Model-based, optimizing compiler | □ Search-based, simple compiler |
| □ Pros | □ Cons |
| ■ Fast | ■ Slow |
| ■ General-purpose | ■ Problem-specific |
| □ Cons | □ Pros |
| ■ Produces sub-optimal performance | ■ Believed to get near-optimal performance |
| ■ Architecture specific | ■ Adapt to the architecture |
-

Previous Work

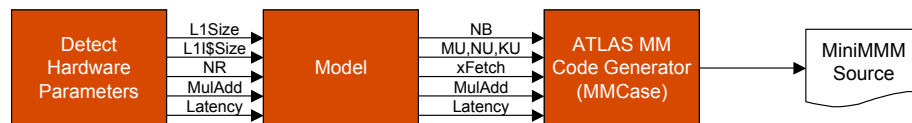
- Compared performance of
 - Sophisticated compilers (like SGI MIPSpro)
 - ATLAS
 - Found that ATLAS code is better
 - Hard to answer why... Maybe ATLAS:
 - has transformations compilers do not know about
 - is performing same transformations but in different order (phase-ordering problem)
 - chooses transformation parameters better
-

Our Approach

■ Original ATLAS Infrastructure



■ Model-Based ATLAS Infrastructure



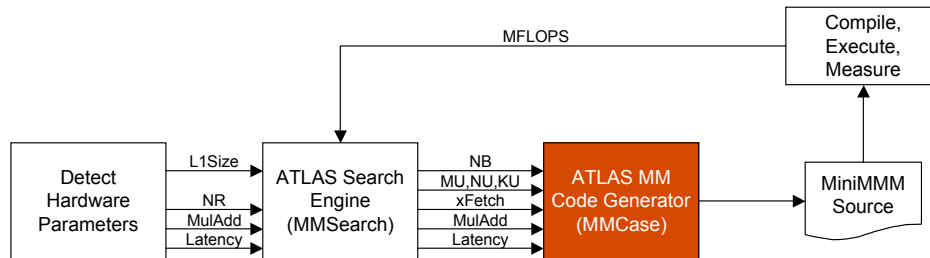
Detecting Machine Parameters

■ Micro-benchmarks

- **L1Size**: L1 Data Cache size
 - Similar to Hennessy-Patterson book
- **NR**: Number of registers
 - Use several FP temporaries repeatedly
- **MulAdd**: Fused Multiply Add (FMA)
 - “ $c+=a*b$ ” as opposed to “ $c+=t; t=a*b$ ”
- **Latency**: Latency of FP Multiplication
 - Needed for scheduling multiplies and adds in the absence of FMA

Compiler View

■ ATLAS Code Generation



■ Focus on MMM (as part of BLAS-3)

- Very good reuse $O(N^2)$ data, $O(N^3)$ computation
- No “real” dependencies (only input / reuse ones)

Optimizations

- **Cache-level blocking (tiling)**
 - Atlas blocks only for L1 cache
- **Register-level blocking**
 - Highest level of memory hierarchy
 - Important to hold array values in registers
- **Software pipelining**
 - Unroll and schedule operations
- **Versioning**
 - Dynamically decide which way to compute
- **Back-end compiler optimizations**
 - Scalar Optimizations
 - Instruction Scheduling

Cache-level blocking (tiling)

Tiling in ATLAS

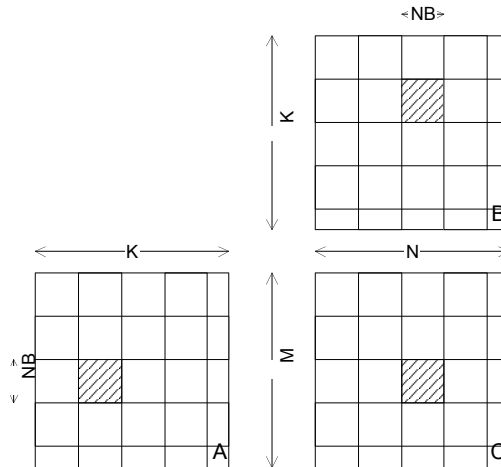
- Only square tiles (NBxNBxNB)
- Working set of tile fits in L1
- Tiles are usually copied to continuous storage
- Special “clean-up” code generated for boundaries

Mini-MMM

```

for (int j = 0; j < NB; j++)
  for (int i = 0; i < NB; i++)
    for (int k = 0; k < NB; k++)
      C[i][j] += A[i][k] * B[k][j]
  
```

NB: Optimization parameter



Register-level blocking

Micro-MMM

- MUx1 elements of A
- 1xNU elements of B
- MUxNU sub-matrix of C
- $MU * NU + MU + NU \leq NR$

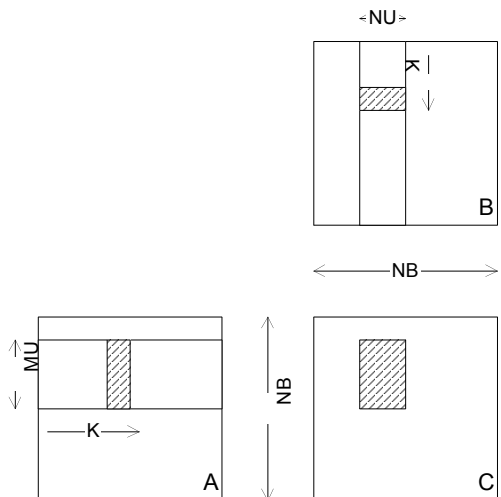
Mini-MMM revised

```

for (int j = 0; j < NB; j += NU)
  for (int i = 0; i < NB; i += MU)
    load C[i..i+MU-1, j..j+NU-1] into registers
    for (int k = 0; k < NB; k++)
      load A[i..i+MU-1, k] into registers
      load B[k, j..j+NU-1] into registers
      multiply A's and B's and add to C's
    store C[i..i+MU-1, j..j+NU-1]
  
```

Unroll K look KU times

MU, NU, KU: optimization parameters



Final Generated Code

- for (j=1; j<=M; j +=NB)
- for (i=1; i<=N; i +=NB)
- for (k=1; k<=K; k +=NB)
- **// mini-MMM code**
- for (jj=1; jj<=j+NB- 1; jj+=NU)
- for (ii=1; ii<=i+NB- 1; ii +=MU)
- for (kk=1; kk<=k+NB- 1; kk++)
- **// micro-MMM code**
- C[ii][jj]+= A[ii][kk] * B[kk][jj]
- C[ii+1][jj]+= A[ii+1][kk] * B[kk][jj]
- C[ii+2][jj]+= A[ii+2][kk] * B[kk][jj]
- C[ii][jj+1]+= A[ii][kk] * B[kk][jj+1]
- C[ii+1][jj+1]+= A[ii+1][kk] * B[kk][jj+1]
- C[ii+2][jj+1]+= A[ii+2][kk] * B[kk][jj+1]

- Innermost loop unrolled, assuming MU=3, and NU=2

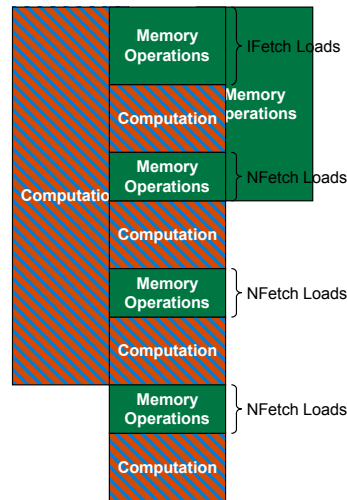
Scheduling

- FMA Present?
- Schedule Computation
 - Using Latency
- Schedule Memory Operations
 - Using FFetch, IFetch, NFetch
- Mini-MMM revised

```

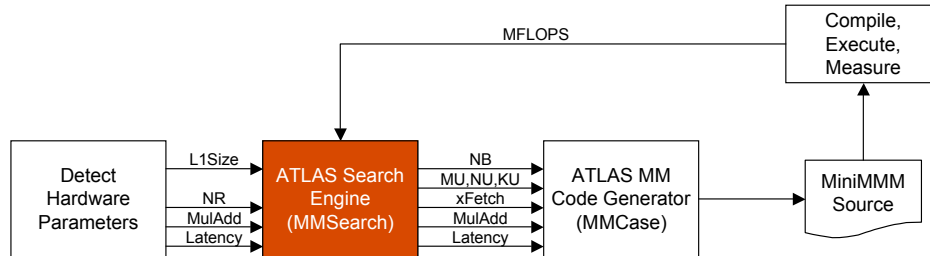
for (int j = 0; j < NB; j += NU)
  for (int i = 0; i < NB; i += MU)
    load C[i..i+MU-1, j..j+NU-1] into
    registers
    for (int k = 0; k < NB; k += KU)
      load A[i..i+MU-1,k] into registers
      load B[k,j..j+NU-1] into registers
      multiply A's and B's and add to C's
      ...
      load A[i..i+MU-1,k+KU-1] into registers
      load B[k+KU-1,j..j+NU-1] into registers
      multiply A's and B's and add to C's
    store C[i..i+MU-1, j..j+NU-1]
    
```

- Latency, xFetch: optimization parameters



Searching for Optimization Parameters

■ ATLAS Search Engine



■ Multi-dimensional search problem

- Optimization parameters are independent variables
- MFLOPS is the dependent variable
- Function is implicit but can be repeatedly evaluated

Search Strategy

■ Orthogonal Range Search

- Optimize along one dimension at a time, using reference values for not-yet-optimized parameters
- Not guaranteed to find optimal point
- **Input**
 - Order in which dimensions are optimized
 - NB, MU & NU, KU, xFetch, Latency
 - Interval in which search is done in each dimension
 - For NB it is $16 \leq NB \leq \min(\sqrt{L1Size}, 80)$ step 4
 - Reference values for not-yet-optimized dimensions
 - Reference values for KU during NB search are 1 and NB

Finding other parameters

- Find best MU, NU : try all MU & NU that satisfy

$$1 \leq MU, NU \leq NB$$

$$MU * NU + MU + NU \leq NR$$

- In this step, use best NB from previous step

- Find best KU

$$ku = 1, \dots, \frac{NB}{2} \text{ step 4, and } NB$$

- Find best Latency

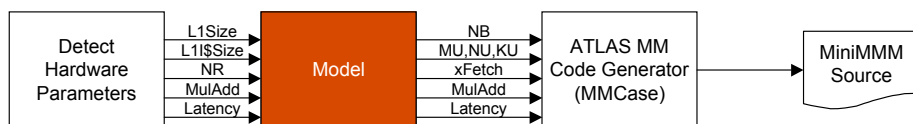
values between 1 and 6

- Find best xFetch

$$IFetch: [2, MU+NU], Nfetch: [1, MU+NU-IFetch]$$

Modeling for Optimization Parameters

- Our Modeling Engine



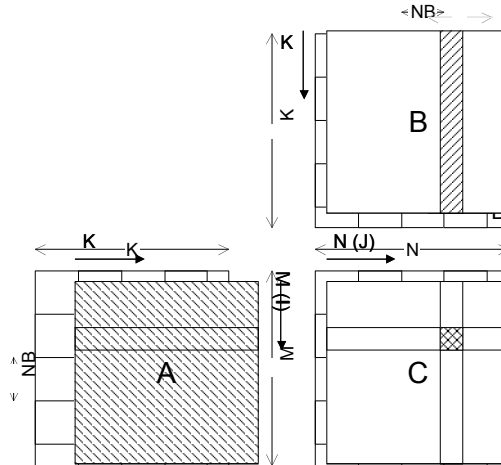
- Optimization parameters

- NB: Hierarchy of Models (later)
- MU, NU: $MU * NU + MU + NU + Latency \leq NR$
- KU: maximize subject to L1 Instruction Cache
- Latency, MulAdd: from hardware parameters
- xFetch: set to 2

Modeling for Tile Size (NB)

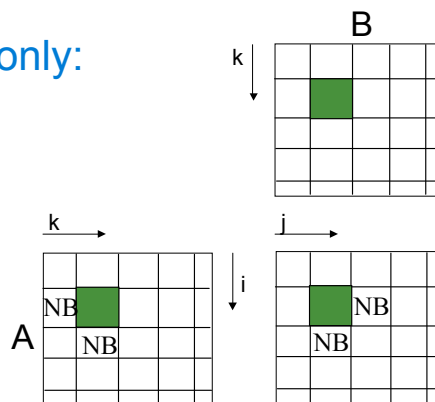
Models of increasing complexity

- $3*NB^2 \leq C$
 - Whole work-set fits in L1
- $NB^2 + NB + 1 \leq C$
 - Fully Associative
 - Optimal Replacement
 - Line Size: 1 word
- $\left\lceil \frac{NB^2}{B} \right\rceil + \left\lceil \frac{NB}{B} \right\rceil + 1 \leq \frac{C}{B}$ or $\left\lceil \frac{NB^2}{B} \right\rceil + NB + 1 \leq \frac{C}{B}$
 - Line Size > 1 word
- $\left\lceil \frac{NB^2}{B} \right\rceil + 2 \left\lceil \frac{NB}{B} \right\rceil + \left(\left\lceil \frac{NB}{B} \right\rceil + 1 \right) \leq \frac{C}{B}$ or $\left\lceil \frac{NB^2}{B} \right\rceil + 3NB + 1 \leq \frac{C}{B}$
 - LRU Replacement



Largest NB for no capacity/conflict misses

- Tiles are copied into contiguous memory
- Condition for cold misses only:
 - $3*NB^2 \leq L1Size$



Largest NB for no capacity misses

- **MMM:**

```
for (int j = 0; j < N; j++)  
  for (int i = 0; i < N; i++)  
    for (int k = 0; k < N; k++)  
      c[i][j] += a[i][k] * b[k][j]
```

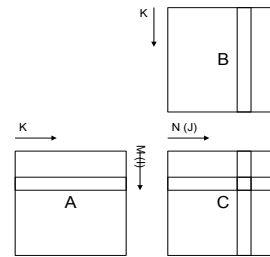
- **Cache model:**

- Fully associative
- Line size 1 Word
- Optimal Replacement

- **Bottom line:**

$NB^2 + NB + 1 \leq L1Size$

- One full matrix
- One row / column
- One element



Extending the Model

- **Line Size > 1**

- Spatial locality
- Array layout in memory matters

- **Bottom line: depending on loop order**

- either
- or

$$\left\lceil \frac{NB^2}{B} \right\rceil + \left\lceil \frac{NB}{B} \right\rceil + 1 \leq \frac{C}{B}$$

$$\left\lceil \frac{NB^2}{B} \right\rceil + NB + 1 \leq \frac{C}{B}$$

Extending the Model (cont.)

- LRU (not optimal replacement)

- MMM sample:

```
for (int j = 0; j < N; j++)
  for (int i = 0; i < N; i++)
    for (int k = 0; k < N; k++)
      c[i][j] += a[i][k] * b[k][j]
```

- Bottom line:

$$\text{IJK, IKJ} \quad \left\lceil \frac{NB^2}{B} \right\rceil + 3NB + 1 \leq \frac{C}{B}$$

$$\text{JIK, JKI} \quad \left\lceil \frac{NB^2}{B} \right\rceil + 3 \left\lceil \frac{NB}{B} \right\rceil + 1 \leq \frac{C}{B}$$

$$\text{KIJ} \quad \left\lceil \frac{NB^2}{B} \right\rceil + 2NB + \left(\left\lceil \frac{NB}{B} \right\rceil + 1 \right) \leq \frac{C}{B}$$

$$\text{KJI} \quad \left\lceil \frac{NB^2}{B} \right\rceil + 2 \left\lceil \frac{NB}{B} \right\rceil + (NB + 1) \leq \frac{C}{B}$$

$$A_{1,1} B_{1,j} A_{1,2} B_{2,j} \llcorner A_{1,NB} B_{NB,j} C_{i,j}$$

$$\boxed{A_{1,1} A_{1,2} ? A_{1,NB} C_{1,j}}$$

$$\boxed{\rightarrow A_{2,1} A_{2,2} ? A_{2,NB} C_{2,j}}$$

?

$$\boxed{\rightarrow A_{NB,1} A_{NB,2} ? A_{NB,NB} C_{NB,j}}$$

$$\boxed{\rightarrow A_{NB,1} B_{1,j} A_{NB,2} B_{2,j} ? A_{NB,NB} B_{NB,j} C_{NB,j}}$$

Comments

- Lot of work in compiler literature on automatic tile size selection
- Not much is known about how well these algorithms do in practice
 - Compare to BLAS/ATLAS/...???
- Not obvious how one generalizes our models to more complex codes
 - Insight needed: how sensitive is performance to tile size??

Experiments

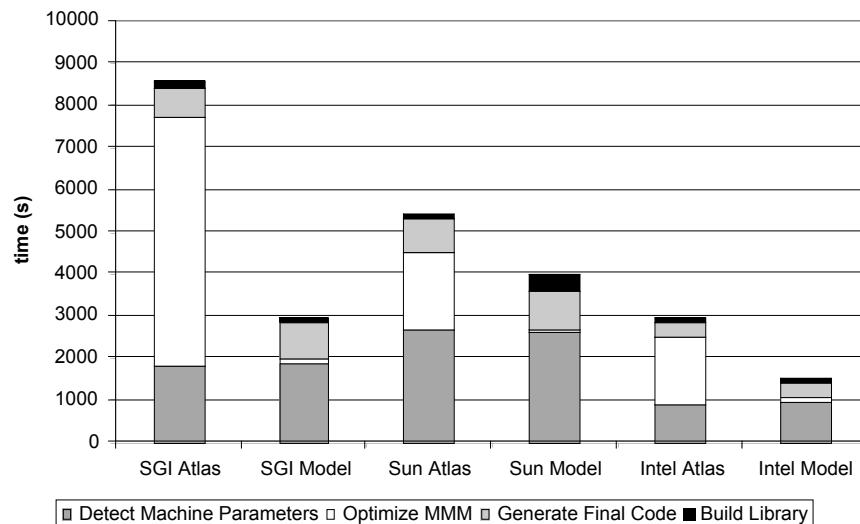
■ Architectures:

- SGI R12000, 270MHz
- Sun UltraSPARC III, 900MHz
- Intel Pentium III, 550MHz

■ Measure

- Mini-MMM performance
 - Complete MMM performance
 - Sensitivity to variations on parameters
-

Installation Time of ATLAS vs. Model



Optimization Parameter Values

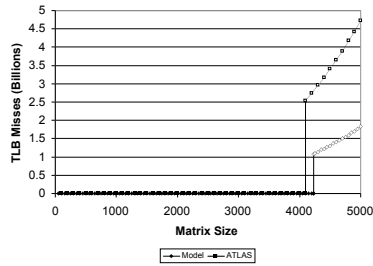
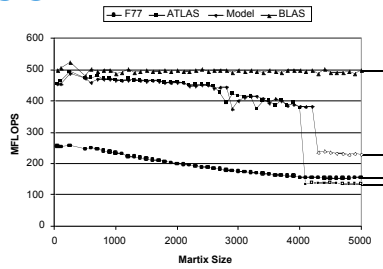
	NB	MU/NU/KU	F/I/N-Fetch	Latency
■ ATLAS				
□ SGI:	64	4/4/64	0/5/1	3
□ Sun:	48	5/3/ 48	0/3/5	5
□ Intel:	40	2/1/40	0/3/1	4
■ Model				
□ SGI:	62	4/4/62	1/2/2	6
□ Sun:	88	4/4/ 78	1/2/2	4
□ Intel:	42	2/1/42	1/2/2	3

MiniMMM Performance

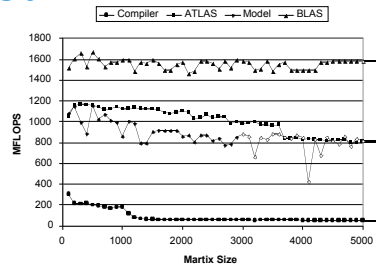
- **SGI**
 - ATLAS: 457 MFLOPS
 - Model: 453 MFLOPS
 - Difference: **1%**
 - **Sun**
 - ATLAS: 1287 MFLOPS
 - Model: 1052 MFLOPS
 - Difference: **20%**
 - **Intel**
 - ATLAS: 394 MFLOPS
 - Model: 384 MFLOPS
 - Difference: **2%**
-

MMM Performance

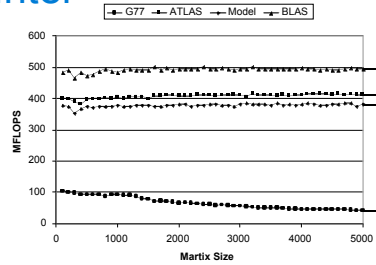
■ SGI



■ Sun

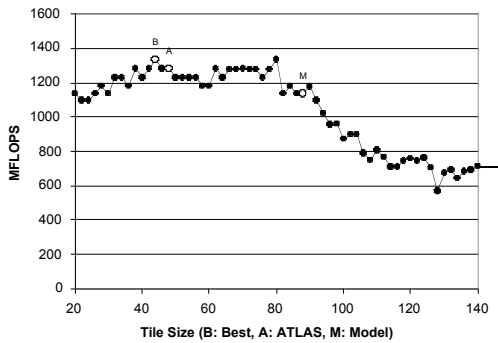


■ Intel

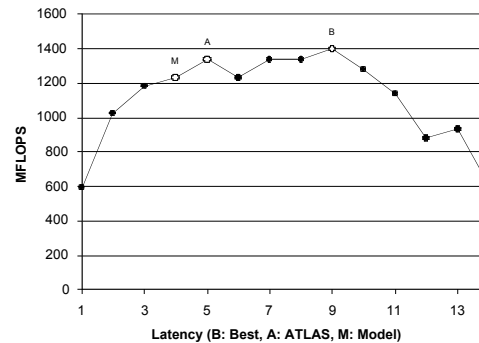


Sensitivity to NB and Latency on Sun

■ Tile Size (NB)

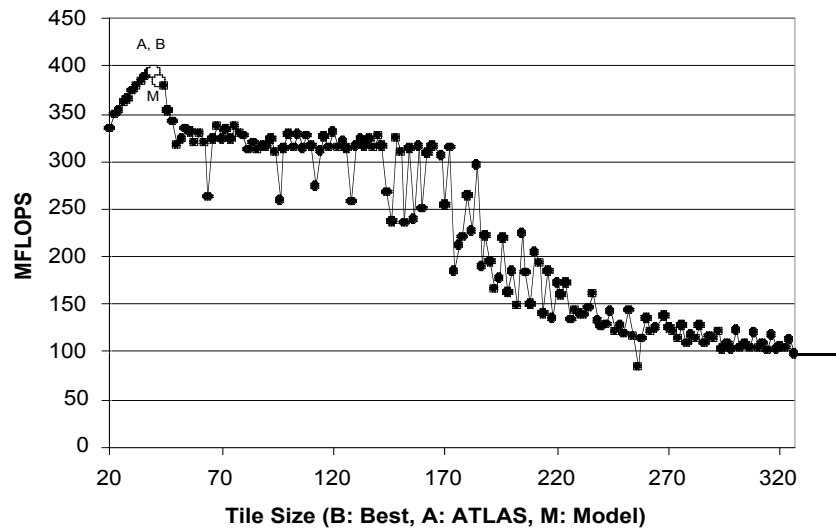


■ Latency

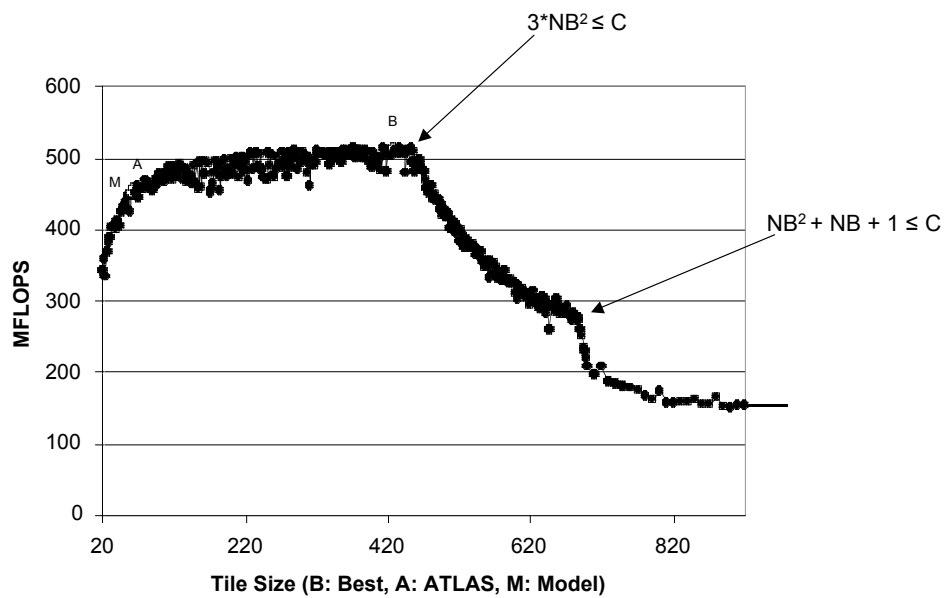


■ MU & NU, KU, Latency, xFetch for all architectures

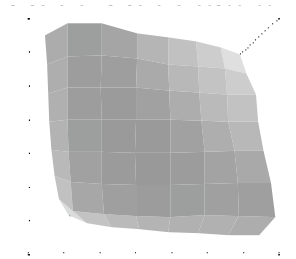
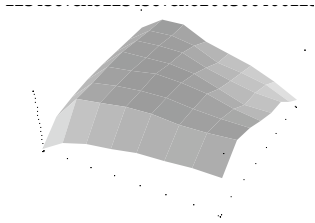
Sensitivity to tile size: Intel



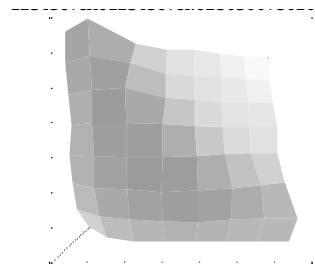
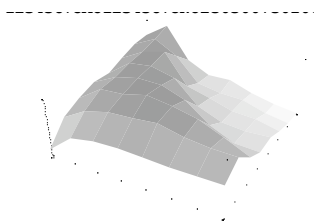
Sensitivity to NB on SGI



Sensitivity to MU,NU: SGI



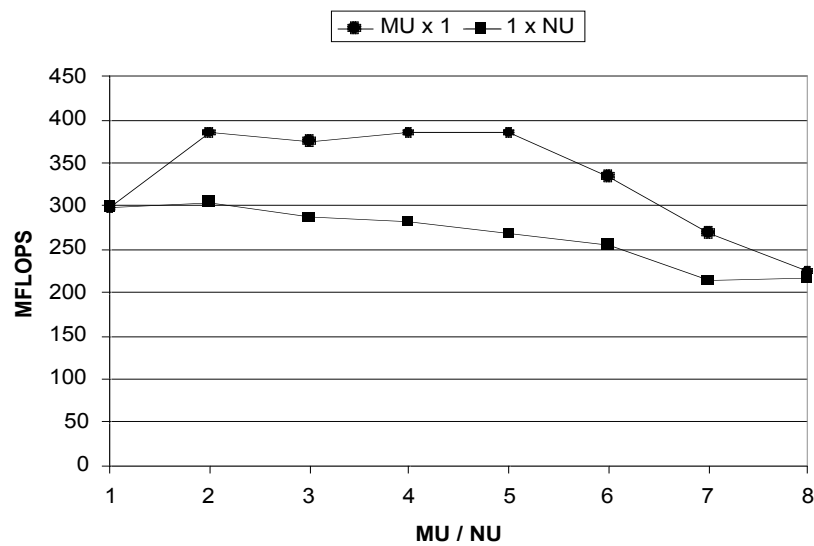
Sensitivity to MU,NU: Sun



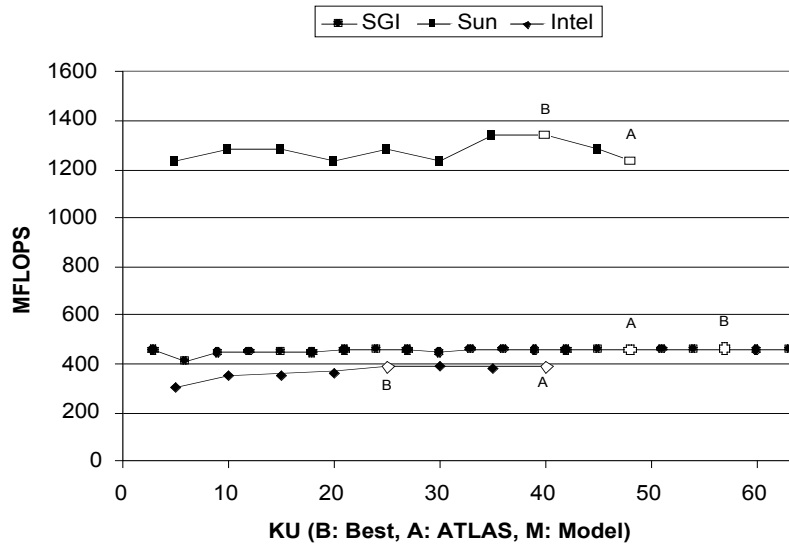
Sensitivity to MU,NU: Intel



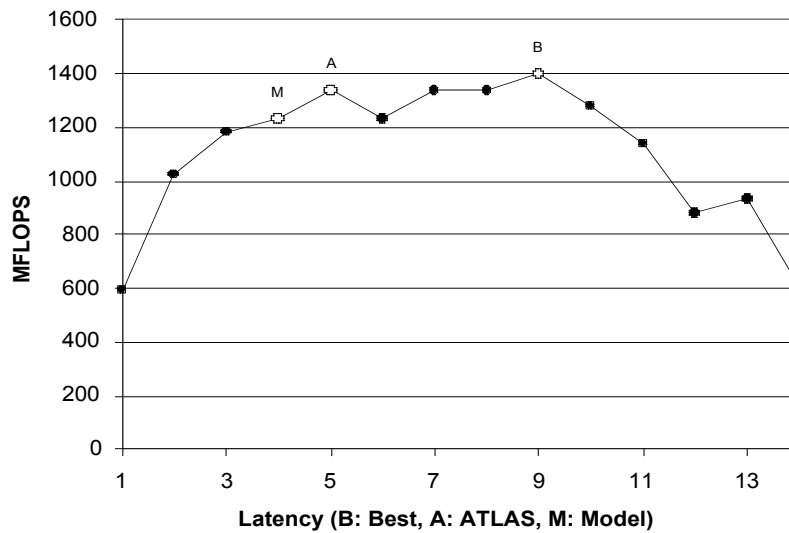
Shape of register tile matters



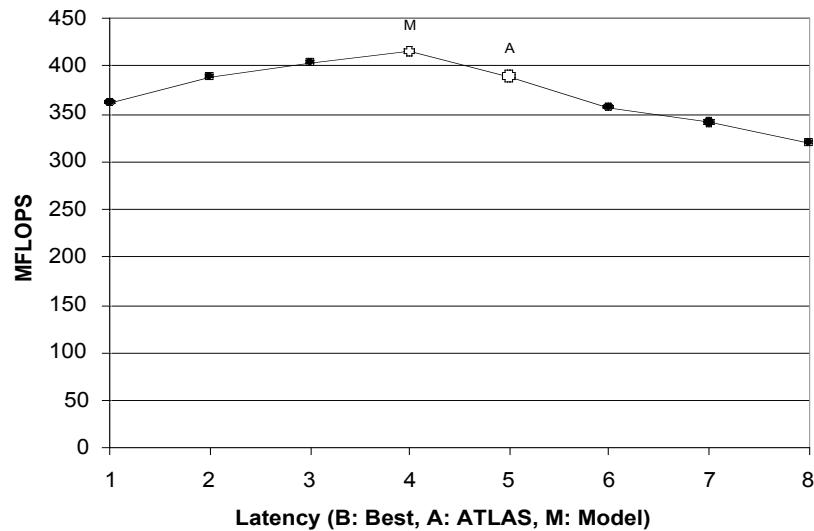
Sensitivity to KU



Sensitivity to Latency: Sun



Sensitivity to Latency: Intel



Conclusions

- **Compilers that**
 - implement well known transformations
 - use models to choose parameter values
 - can achieve performance close to ATLAS
 - **There is room for improvement in both models and empirical search**
 - Both are 20-25% slower than BLAS
 - Higher levels of memory hierarchy cannot be neglected
-

Future Work

- Do similar experiments with FFTW
 - FFTW uses search to choose between algorithms, not to find values for parameters
 - Use models to speed up empirical search in ATLAS
 - Effectively combining models with search
 - Understand what is missing from models and refine them
 - Study hand-written BLAS codes to understand the gap
 - Feed these insights back into compilers
 - Better models
 - Some search if needed
 - How do we make it easier for compiler writers to implement such transformations?
-