An Experimental Comparison of Empirical and Model-based Optimization

Kamen Yotov
Cornell University
Joint work with:
Xiaoming Li, Gang Ren, Michael Cibulskis, Gerald DeJong, Maria Garzaran, David Padua,
Keshav Pingali, Paul Stodghill, Peng Wu

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 $\rightarrow$ Intelligent Search
  - Use empirical observations to refine models
    - Learn from experience
## Optimization: Compilers vs. Library Generators

<table>
<thead>
<tr>
<th>Traditional Compilers</th>
<th>ATLAS / FFTW / Spiral</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-based, optimizing compiler</td>
<td>Search-based, simple compiler</td>
</tr>
<tr>
<td><strong>Pros</strong></td>
<td><strong>Cons</strong></td>
</tr>
<tr>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>General-purpose</td>
<td>Problem-specific</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td><strong>Pros</strong></td>
</tr>
<tr>
<td>Produces sub-optimal performance</td>
<td>Believed to get near-optimal performance</td>
</tr>
<tr>
<td>Architecture specific</td>
<td>Adapt to the architecture</td>
</tr>
</tbody>
</table>

## 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 (NB x NB x NB)
  - 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**
  - MU x 1 elements of A
  - 1 x NU elements of B
  - MU x NU sub-matrix of C
  - MU * NU + MU + NU ≤ 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
Scheduling

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

```c
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
    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

Modeling for Optimization Parameters

- **Our Modeling Engine**

- **Optimization parameters**
  - NB: Hierarchy of Models (later)
  - MU, NU: $MU \times NU + MU + NU + Latency \leq NR$
  - 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
  - $3\cdot NB^2 \leq C$
    - Whole work-set fits in L1
  - $NB^2 + NB + 1 \leq C$
    - Fully Associative
    - Optimal Replacement
    - Line Size: 1 word
  - $\left\lceil \frac{NB}{B} \right\rceil + \left\lceil \frac{NB}{B} \right\rceil + 1 \leq \frac{C}{B}$ or $\left\lceil \frac{NB}{B} \right\rceil + NB + 1 \leq \frac{C}{B}$
    - Line Size > 1 word
  - $\left\lceil \frac{NB}{B} \right\rceil \geq 2\left(\left\lceil \frac{NB}{B} \right\rceil + 1\right) \leq \frac{C}{B}$ or $\left\lceil \frac{NB}{B} \right\rceil + 3NB + 1 \leq \frac{C}{B}$
    - LRU Replacement

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

<table>
<thead>
<tr>
<th></th>
<th>NB</th>
<th>MU/NU/KU</th>
<th>F/I/N-Fetch</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATLAS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGI:</td>
<td>64</td>
<td>4/4/64</td>
<td>0/5/1</td>
<td>3</td>
</tr>
<tr>
<td>Sun:</td>
<td>48</td>
<td>5/3/48</td>
<td>0/3/5</td>
<td>5</td>
</tr>
<tr>
<td>Intel:</td>
<td>40</td>
<td>2/1/40</td>
<td>0/3/1</td>
<td>4</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGI:</td>
<td>62</td>
<td>4/4/62</td>
<td>1/2/2</td>
<td>6</td>
</tr>
<tr>
<td>Sun:</td>
<td>88</td>
<td>4/4/78</td>
<td>1/2/2</td>
<td>4</td>
</tr>
<tr>
<td>Intel:</td>
<td>42</td>
<td>2/1/42</td>
<td>1/2/2</td>
<td>3</td>
</tr>
</tbody>
</table>
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 NB on SGI

- $3NB^2 \leq C$
- $NB^2 + NB + 1 \leq C$
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?