ATLAS Library Generator

Lecture based on these papers:

- "A comparison of empirical and model-driven optimization"
  Yotov et al, PLDI 2003
- "Is search really necessary to generate high-performance BLAS?"
  Yotov et al, Proceedings of IEEE, 93(2), 2005
- "Think globally, search locally"
  Yotov et al., ICS 2005

Overview

- ATLAS: portable BLAS generator
  - Implemented by Dongarra (UTK), Demmel(UCB) et al.
  - We will focus on MMM
- Importance: popularized the idea of auto-tuning
  - Generate-and-test
  - Program generator to generate program variants
  - Test performance of variant by running program
- Program variants in ATLAS
  - Iterative blocked kernels for different levels of memory hierarchy
- Auto-tuning useful for library generators
  - Large upfront cost for generate-and-test

BLAS

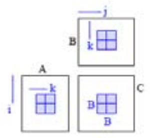
- Let us focus on MMM:

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

- Properties
  - Very good reuse: $O(N^2)$ data, $O(N^3)$ computation
  - Many optimization opportunities
    - Few "real" dependencies
  - Will run poorly on modern machines
    - Poor use of cache and registers
    - Poor use of processor pipelines
- Key optimization
  - Blocking/tiling to improve temporal locality

Why blocking?

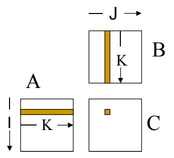
```
for bi = 1,N,B
 for bj = 1,N,B
  for bk = 1,N,B
   for i = bi, min(bi+B-1,N)
     for j = bj, min(bj+B-1,N)
      for k = bk, min(bk+B-1,N)
          y(i) = y(i) + A(i,j)*x(j)
```



- Assume blocks fit in cache during block computation
- # of cache misses for block data = $3B^2/L$ (L: line size)
- # of block computations = $(N/B)^3$
- Total number of misses = $(N/B)^3 * (3B^2/L) = 3N^3/BL$
- High-level picture:
  - number of cache misses decreases with block size as long as working set of block computation fits in cache

Optimal block size

```
for I = 1, B
   for J = 1, B
      for K = 1, B
         C(I,J) = C(I,J) + A(I,K)*B(K,J)
```

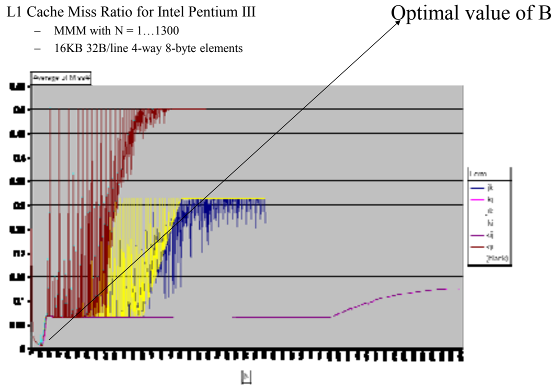


- Easy computation
  - Need space in cache for 3 blocks of B^2
  - So choose largest B for which $3B^2 < C$
- Careful accounting: to avoid capacity misses, need space in cache for
  - block of B
  - row of A
  - one element (line) of C
  - loop order determines which matrices
- For our problem:
  - $B^2 + B + L < C$ (with optimal replacement)
  - $B^2 + 2B < C$ (with LRU replacement)
  - In either case, we get $B \sim \text{sqrt}(C)$

MMM experiments

L1 Cache Miss Ratio for Intel Pentium III

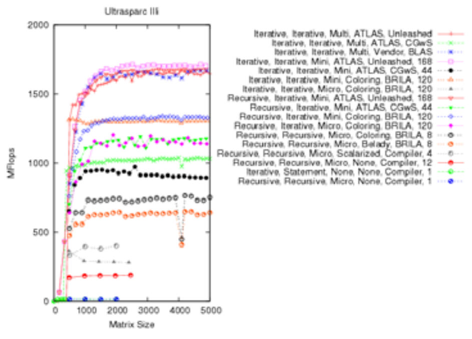
- MMM with N = 1…1300
- 16KB 32B/line 4-way 8-byte elements



High-level picture of high-performance MMM code

- Block the code for each level of memory hierarchy
  - Registers: requires loop unrolling
  - L1 cache
  - …..
- Choose block sizes at each level using the theory described previously
  - Useful optimization: choose block size at level L+1 to be multiple of the block size at level L

Importance of multi-level blocking



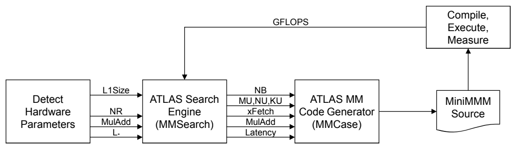
"An Experimental Comparison of Cache-oblivious and Cache-conscious programs"
Yotov et al, SPAA 2007

ATLAS

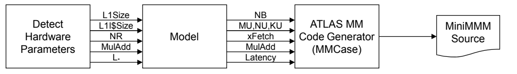
- Library generator for MMM and other BLAS
- Blocks only for registers and L1 cache
- Uses search to determine block sizes, rather than the analytical formulas we used
  - Search takes more time, but we do it once when library is produced
- Let us study structure of ATLAS in little more detail

Study in Yotov et al. paper

- Original ATLAS Infrastructure


GFLOPS
Compile, Execute, Measure
Detect Hardware Parameters
L1Size
NR
MulAdd
L*
ATLAS Search Engine (MMSearch)
NB
MU,NU,KU
xFetch
MulAdd
Latency
ATLAS MM Code Generator (MMCase)
MiniMMM Source

- Model-Based ATLAS Infrastructure



Parameters in ATLAS code

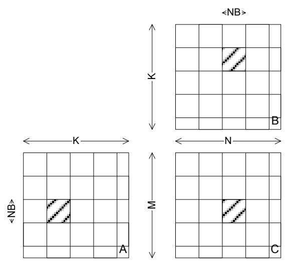
- Cache-level blocking (tiling)
  - Atlas blocks only for L1 cache
  - NB: L1 cache time size
- Register-level blocking
  - Important to hold array values in registers
  - MU,NU: register tile size
- Filling the processor pipeline
  - Unroll and schedule operations
  - Latency, xFetch: scheduling parameters
- Versioning
  - Dynamically decide which way to compute
- Back-end compiler optimizations: nothing to do with ATLAS
  - 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

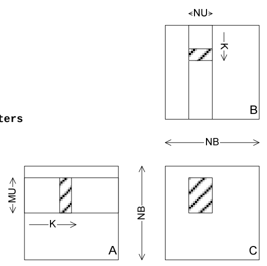


Register-level blocking

- Micro-MMM
  - A: MUx1
  - B: 1xNU
  - C: MUxNU
  - MUxNU+MU+NU registers
- Unroll loops by MU, NU, and KU
- Mini-MMM with Micro-MMM inside

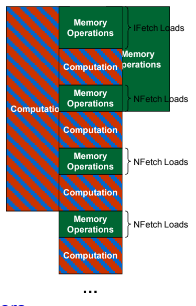
```
for (int j = 0; j < NB; j += NU)
  for (int i = 0; i < NB; i += MU)
    load C[i..i+MU-1, j..j+NU-1] into registers
    for (int k = 0; k < NB; k++)
           ┌load A[i..i+MU-1,k] into registers
KU times  ─┤load B[k,j..j+NU-1] into registers
           └multiply A's and B's and add to C's
    store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

- Special clean-up code required if NB is not a multiple of MU,NU,KU
- MU, NU, KU: optimization parameters



Scheduling

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



- Latency, xFetch: optimization parameters

Search Strategy

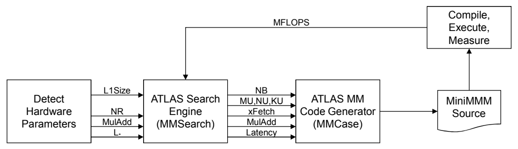
- Multi-dimensional optimization problem:
  - Independent parameters: NB,MU,NU,KU,…
  - Dependent variable: GFlops
  - Function from parameters to variables is given implicitly; can be evaluated repeatedly
- One optimization strategy: orthogonal line search
  - Optimize along one dimension at a time, using reference values for parameters not yet optimized
  - Not guaranteed to find optimal point, but might come close

Find Best NB

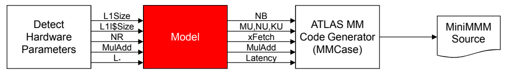
- Search in following range
  - 16 <= NB <= 80
  - NB^2 <= L1Size
- In this search, use simple estimates for other parameters
  - (eg) KU: Test each candidate for
    - Full K unrolling (KU = NB)
    - No K unrolling (KU = 1)

Model-based optimization

- Original ATLAS Infrastructure


MFLOPS
Compile, Execute, Measure
Detect Hardware Parameters
L1Size
NR
MulAdd
L*
ATLAS Search Engine (MMSearch)
NB
MU,NU,KU
xFetch
MulAdd
Latency
ATLAS MM Code Generator (MMCase)
MiniMMM Source

- Model-Based ATLAS Infrastructure

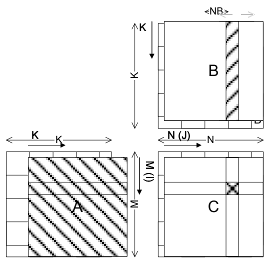


Modeling for Optimization Parameters

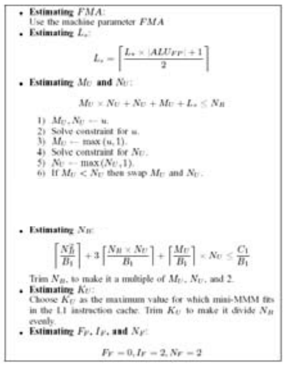
- Optimization parameters
  - NB
    - Hierarchy of Models (later)
  - MU, NU
    - $MU * NU + MU + NU + Latency \le NR$
  - KU
    - maximize subject to L1 Instruction Cache
  - Latency
    - $\lceil (L_* + 1)/2 \rceil$
  - MulAdd
    - hardware parameter
  - xFetch
    - set to 2

Modeling for Tile Size (NB)

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



Summary of model



- **Estimating** FMA:
  Use the machine parameter FMA
- **Estimating** L_s:

$$L_s = \left\lceil \frac{L_* \times |ALU_{FP}| + 1}{2} \right\rceil$$

- **Estimating** M_U **and** N_U:

$$M_U \times N_U + N_U + M_U + L_s \le N_R$$

1) $M_U, N_U \leftarrow u$.
2) Solve constraint for u.
3) $M_U \leftarrow \max(u, 1)$.
4) Solve constraint for N_U.
5) $N_U \leftarrow \max(N_U, 1)$.
6) If $M_U < N_U$ then swap M_U and N_U.

- **Estimating** N_B:

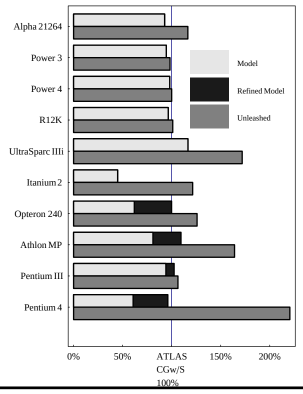
$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + 3\left\lceil \frac{N_B \times N_U}{B_1} \right\rceil + \left\lceil \frac{M_U}{B_1} \right\rceil \times N_U \le \frac{C_1}{B_1}$$

Trim N_B, to make it a multiple of M_U, N_U, and 2.
- **Estimating** K_U:
  Choose K_U as the maximum value for which mini-MMM fits in the L1 instruction cache. Trim K_U to make it divide N_B evenly.
- **Estimating** F_F, I_F, **and** N_F:

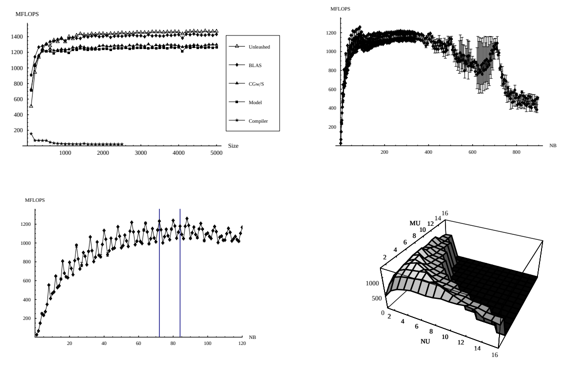
$$F_F = 0, I_F = 2, N_F = 2$$

Experiments

- Ten modern architectures
- Model did well on
  - RISC architectures
  - UltraSparc: did better
- Model did not do as well on
  - Itanium
  - CISC architectures
- Substantial gap between ATLAS CGw/S and ATLAS Unleashed on some architectures



Some sensitivity graphs for Alpha 21264

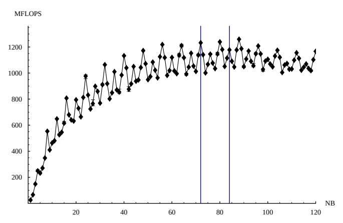


Eliminating performance gaps

- Think globally, search locally
- Gap between model-based optimization and empirical optimization can be eliminated by
  - Local search:
    - for small performance gaps
    - in neighborhood of model-predicted values
  - Model refinement:
    - for large performance gaps
    - must be done manually
    - (future) machine learning: learn new models automatically
- Model-based optimization and empirical optimization are not in conflict

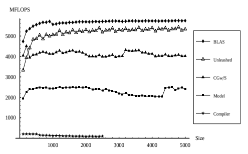
Small performance gap: Alpha 21264

ATLAS CGw/S:
  mini-MMM: 1300 MFlops
  NB = 72
  (MU,NU) = (4,4)
ATLAS Model
  mini-MMM: 1200 MFlops
  NB = 84
  (MU,NU) = (4,4)

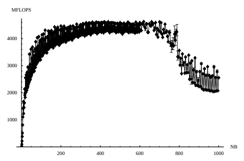


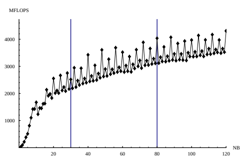
- Local search
  - Around model-predicted NB
  - Hill-climbing not useful
  - Search interval:[NB-lcm(MU,NU),NB+lcm(MU,NU)]
- Local search for MU,NU
  - Hill-climbing OK

Large performance gap: Itanium



MMM Performance





NB Sensitivity

Performance of mini-MMM
- ATLAS CGw/S: 4000 MFlops
- ATLAS Model: 1800 MFlops

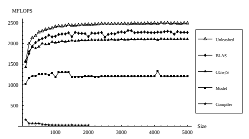
Problem with NB value
ATLAS Model: 30
ATLAS CGw/S: 80 (search space max)

Local search will not solve problem.

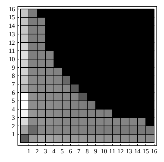
Itanium diagnosis and solution

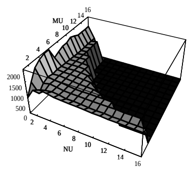
- Memory hierarchy
  - L1 data cache: 16 KB
  - L2 cache: 256 KB
  - L3 cache: 3 MB
- Diagnosis:
  - Model tiles for L1 cache
  - On Itanium, FP values not cached in L1 cache!
  - Performance gap goes away if we model for L2 cache (NB = 105)
  - Obtain even better performance if we model for L3 cache (NB = 360, 4.6 GFlops)
- Problem:
  - Tiling for L2 or L3 may be better than tiling for L1
  - How do we determine which cache level to tile for??
- Our solution: model refinement + a little search
  - Determine tile sizes for all cache levels
  - Choose between them empirically

Large performance gap: Opteron



MMM Performance





MU,NU Sensitivity

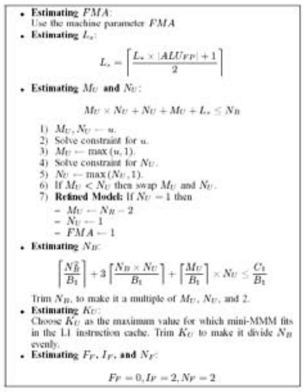
Performance of mini-MMM
- ATLAS CGw/S: 2072 MFlops
- ATLAS Model: 1282 MFlops

Key differences in parameter values:MU/NU
- ATLAS CGw/S: (6,1)
- ATLAS Model: (2,1)

Opteron diagnosis and solution

- Opteron characteristics
  - Small number of logical registers
  - Out-of-order issue
  - Register renaming
- For such processors, it is better to
  - let hardware take care of scheduling dependent instructions,
  - use logical registers to implement a bigger register tile.
- x86 has 8 logical registers
  - → register tiles must be of the form (x,1) or (1,x)

Refined model



- **Estimating** FMA:
  Use the machine parameter FMA
- **Estimating** L_s:

$$L_s = \left\lceil \frac{L_* \times |ALU_{FP}| + 1}{2} \right\rceil$$

- **Estimating** M_U **and** N_U:

$$M_U \times N_U + N_U + M_U + L_s \leq N_R$$

```
1) $M_U, N_U \leftarrow u$.
2) Solve constraint for $u$.
3) $M_U \leftarrow \max(u, 1)$.
4) Solve constraint for $N_U$.
5) $N_U \leftarrow \max(N_U, 1)$.
6) If $M_U < N_U$ then swap $M_U$ and $N_U$.
7) Refined Model: If $N_U = 1$ then
    – $M_U \leftarrow N_R - 2$
    – $N_U \leftarrow 1$
    – $FMA \leftarrow 1$
```

- **Estimating** N_B:

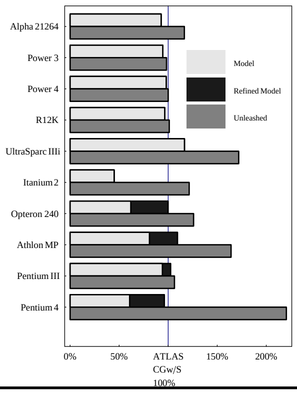
$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + 3\left\lceil \frac{N_B \times N_U}{B_1} \right\rceil + \left\lceil \frac{M_U}{B_1} \right\rceil \times N_U \leq \frac{C_1}{B_1}$$

  Trim N_B, to make it a multiple of M_U, N_U, and 2.
- **Estimating** K_U:
  Choose K_U as the maximum value for which mini-MMM fits in the L1 instruction cache. Trim K_U to make it divide N_B evenly.
- **Estimating** F_F, I_F, **and** N_F:

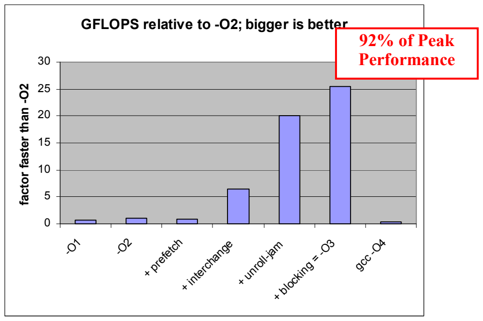
$$F_F = 0, I_F = 2, N_F = 2$$

Bottom line

- Refined model is not complex.
- Refined model by itself eliminates most performance gaps.
- Local search eliminates all performance gaps.



Performance of MMM code produced by Intel's Itanium compiler (-O3)



Goto BLAS obtains close to 99% of peak, so compiler is pretty good!

Things to think about

- What is the space of program variants?
  - Space must include the optimal point or at least points close to it in performance
  - Question: what kinds of MMM implementations are not explored by ATLAS?
- What is the search strategy and is it guaranteed to find the optimal (or at least very good) point?
  - ATLAS uses orthogonal line search
  - One general problem: you spend much more time executing non-optimal program variants than the optimal one!
    - Some notion of importance sampling might be useful if search time matters
- What were the key approximations made in the analytical performance model?
  - Need to look at model used for each parameter
  - Are these approximations reasonable?