Automatic Tuning of Scientific Applications

Apan Qasem       Ken Kennedy
Rice University
Houston, TX
Recap from Last Year

- A framework for automatic tuning of applications
  - Fine grain control of transformations
  - Feedback beyond whole program execution time
  - Parameterized Search Engine
  - Target: Whole Applications
  - Search Space
    - Multi-loop transformations: e.g. Loop Fusion
    - Numerical Parameters
Recap from Last Year

- Experiments with Direct Search
  - Direct Search able to find suitable tile sizes and unroll factors by exploring only a small fraction of the search space
    - 95% of best performance obtained by exploring 5% of the search space
  - Search space pruning needed for making search more efficient
    - Wandered into regions containing mostly bad values
    - Direct search required more than 30 program evaluations in many cases
Today’s talk

Search Space Pruning
Search Space Pruning

• Key Idea:

  Search for architecture-dependent model parameters rather than transformation parameters

• Fundamentally different way of looking at the optimization search space
• Implemented for loop fusion and tiling
  [Qasem and Kennedy, ICS06]
Architectural Parameters

- Register Set
- Effective Register Set
- L1 Cache
- Effective Cache Capacity
- Cost Model

Search Space

- Tile Size
- Fusion Config.

New Search Space has only two dimensions!

Estimates of architectural parameters

Gray Code Representation

(L + 1) dimensions

(N- p) x 2^{p} points
Our Approach

• Build a *combined cost model* for fusion and tiling to capture the interaction between the two transformations
  - Use reuse analysis to estimate trade-offs

• Expose *architecture-dependent parameters* within the model for tuning through empirical search
  - Pick T such that
    • Working Set < Effective Cache Capacity
  - Search for suitable Effective Cache Capacity
Tuning Parameters

- Use a *tolerance term* to determine how much of a resource we can use at each tuning step.

**Effective Register Set**

\[ \text{Effective Register Set} = \left[ T \times \text{Register Set Size} \right] \]

\[ 0 < T \leq 1 \]

**Effective Cache Capacity**

\[ \text{Effective Cache Capacity} = E(a, s, T) \]

\[ 0.01 \leq T \leq 0.20 \]
Search Strategy

- Start off conservatively with a low tolerance value and increase tolerance at each step
- Each tuning parameter constitutes a single search dimension
- Search is *sequential* and *orthogonal*
  - stop when performance starts to worsen
  - use reference values for other dimensions when searching a particular dimension
Benefits of Pruning Strategy

- Reduce the size of the exploration search space
  - Single parameter captures the effects of multiple transformations
    - Effective cache capacity for fusion and tiling choices
    - Register pressure for fusion and loop unrolling
  - Search Space does not grow with program size
    - One parameter for all tiled loops in the application
- Correct for inaccuracies in model
Performance Across Architectures

Platforms

- MIPS
- Itanium
- Alpha
- PowerPC
- Pentium III
- Geo Mean

Mean Speedup over baseline

- Model-based
- Native
Performance Comparison with Direct Search

- advect3d
- erle
- liv18
- mgrid

model-based versus direct

AutoTuning Workshop 06
Rice University
Tuning Time Comparison with Direct Search

![Bar Chart showing tuning time comparison between model-based and direct methods for advect3d, erle, liv18, and mgrid.](chart.png)
Conclusions and Future Work

- Approach of tuning for architectural parameters can significantly reduce the optimization search space, while incurring only a small performance penalty

- Extend pruning strategy to cover more
  - transformations
    - Unroll-and-jam
    - Array Padding
  - architectural parameters
    - TLB
Questions
Extra Slides Begin Here
Tuning Time Comparison

Fraction of Tuning Time

Benchmarks

advec3d  lud  mm  vpenta  mgrid  swim

DS 30
DS 60
DS 90
DS 120
Framework Overview

User → Tuning Level

Source → LoopTool

LoopTool → Transformed Source → Binary

Transformed Source → Native Compiler

Native Compiler → Performance Measurement Tools

Performance Measurement Tools → Feedback

Search Space → Parameterized Search Engine

Parameterized Search Engine → Next Iteration Parameters

Next Iteration Parameters → LoopTool

Architectural Specs → Tuning Level

Architectural Specs → Native Compiler
Why Direct Search?

• Search decision based solely on function evaluations
  – No modeling of the search space required
• Provides approximate solutions at each stage of the calculation
  – Can stop the search at any point when constrained by tuning time
• Flexible
  – Can tune step sizes in different dimensions
• Parallelizable
• Relatively easy to implement
L_A: do j = 1, N  
   do i = 1, M  
     b(i,j) = a(i,j) + a(i,j-1)  
   enddo  
enddo

L_B: do j = 1, N  
   do i = 1, M  
     c(i,j) = b(i,j) + d(j)  
   enddo  
enddo

(a) code before transformations
\[ \text{L}_{AB}: \quad \text{do } j = 1, N \]
\[ \quad \text{do } i = 1, M \]
\[ \quad b(i,j) = a(i,j) + a(i,j-1) \]
\[ \quad c(i,j) = b(i,j) + d(j) \]

- **Lost reuse of** \(a()\)
- **Saved loads of** \(b()\)
- **Increased potential for conflict misses**

(b) code after two-level fusion
\[ \text{do } i = 1, M, T \]
\[ \text{do } j = 1, N \]
\[ \text{do } ii = i, i + T - 1 \]
\[ b(ii,j) = a(ii,j) + a(ii,j-1) \]
\[ c(ii,j) = b(ii,j) + d(j) \]
\]

How do we pick $T$?

**Not too difficult if caches are fully associative**

**Can use models to estimate effective cache size for set-associative caches**

**Model unlikely to be totally accurate**
- Need a way to correct for inaccuracies
Register Set

L1 Cache

Cost Models

Tile Size

Fusion Config.

\[ \begin{align*}
T_0 & = T_{0N} \\
T_1 & = T_{1N} \\
T_2 & = T_{2N} \\
T_3 & = T_{3N} \\
F_0 & = F_2 \\
F_1 & = F_2 \\
F_2 & = F_2 \\
\end{align*} \]

\( (N - p)_L \times (2^L - q) \) points
Register Set

L1 Cache

Effective Register Set

Effective Cache Capacity

Cost Model

Tile Size

Fusion Config.

Estimates of machine parameters

(L + 1) dimensions

(N - p) x (2^L - q) points

2 dimensions