Using Cache Models and Empirical Search in Automatic Tuning of Applications

Apan Qasem   Ken Kennedy   John Mellor-Crummey
Rice University
Houston, TX
Outline

• Overview of Framework
  – Fine grain control of transformations
  – Loop-level feedback
  – Search strategy

• Experimental Results

• Empirical Tuning of Loop Fusion

• Future Work
A Framework for Autotuning

- **F77 Source**
- **Annotated Source**
- **LoopTool**
- **Transformed Source**
- **Vendor Compiler**
- **Binary**
- **hpcrun**
- **bloop**

- **Architectural Specs**
- **Initial Parameters**
- **Search State**
- **Program Structure**
- **Profile Information**
- **Simulate d**
- **Annotated**
- **Direct Search**
- **Itanium2**
- **SGI**
- **Alpha**

- **Feedback**

- **Next Iteration**
- **Parameters**
- **Search Space**
Fine Grain Control of Transformations

- Not enough control over transformations in most commercial compilers
- Example sweep3d
  - 68 loops in the sweep routine
  - Using the same unroll factor for all 4 loops results in a 2% speedup
  - Using 4 different unroll factors for 4 loops results in a 8% speedup
Fine Grain Control of Transformations

- LoopTool: A Source to source transformer
  - Performs transformations such as loop fusion, tiling, unroll-and-jam
  - Applies transformations from source code annotation
  - Provides loop level control of transformations

```c
    
    cdir$ unroll 4
    do j = 1, N
    cdir$ block 16
    do i = 1, M
    cdir$ block 16
    do k = 1, L
    S1(k, i, j)
    enddo
    enddo
    enddo
```
Feedback

- Feedback beyond whole program execution time
  - Need feedback at loop level granularity
    - Can potentially optimize independent code regions concurrently
  - Need other performance metrics:
    - Cache misses
    - TLB misses
    - Pipeline stalls
Feedback

• HPCToolKit
  – hpcren – profiles executions using statistical sampling of hardware performance counters
  – bloop – retrieves loop structure from binaries
  – hpctview – correlates program structure information with sample-based performance profiles
Search Space

- Optimization search space is very large and complex
  - Example mm
    - 2-level blocking (range 1–100)
    - 1-unrolled loop (range 1–20)
    - $100 \times 100 \times 20 = 200,000$ search points
  - Characteristics of the search space is hard to predict
  - Need a search technique that can explore the search space efficiently
Search Strategy

• Pattern-based direct search method
  - The search is derivative-free
    • Works well for a non-continuous search space
  - Provides approximate solutions at each stage of the calculation
    • Can stop the search at any point when constrained by tuning time
  - Provides flexibility
Performance Improvement Comparison

![Graph showing performance improvement comparison](image)

- **Exhaustive Search**
- **Direct Search**
- **Random Search**

**Y-axis:** Fraction of Best Speedup

**X-axis:** Number of Iterations
Empirical Tuning of Loop Fusion

• Important transformation when looking at whole applications

• Poses new problems
  – Multi-loop reuse analysis
  – Interaction with tiling
  – What parameters to tune?
    • Trying all combinations is perhaps too much
Profitability Model for Fusion

• Hierarchical reuse analysis to estimate cost at different memory levels
• A probabilistic model to account for conflict misses
  – Effective Cache Size
• Resource constraints
  – Register Pressure
  – Instruction Cache Capacity
• Integrate the model into a pair-wise greedy fusion algorithm
Tuning Parameters for Fusion

- Use search to tune the following parameters
  - Effective Cache Size
  - Register Pressure
  - Instruction Cache capacity

- Preliminary Experiments
  - Works reasonably well
Speedup from Loop Fusion

![Speedup from Loop Fusion](chart.png)

- **ccfm**
- **simple**
- **mips-pro**
- **nofuse**

**Benchmarks**
- advent3d
- erlbacher
- liv18
- mgrid

**Speedup over no-fuse**

- 0.7
- 0.8
- 0.9
- 1.0
- 1.1
- 1.2
- 1.3
- 1.4
- 1.5
Summary

- Autotuning framework able to improve performance on a range of architectures
- Direct search able to find *good* tile sizes and unroll factors by exploring only a small fraction of the search space
- Combining static models with search works well for loop fusion
Future Work

• Refine cache models to capture interactions between transformations
• Consider data allocation strategies for tuning
Extra Slides Begin Here
## Platforms

<table>
<thead>
<tr>
<th>Processor</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha 21264A @ 667MHz</td>
<td>8K</td>
<td>8M</td>
<td>-</td>
<td>Compaq Fortran V5.5-1877</td>
</tr>
<tr>
<td>Itanium2 @ 900 MHz</td>
<td>16K</td>
<td>256K</td>
<td>1.5M</td>
<td>Intel Fortran Compiler 7.1</td>
</tr>
<tr>
<td>SGI Origin R10K @ 195 MHz</td>
<td>32K</td>
<td>1M</td>
<td>-</td>
<td>MipsPro 7.3</td>
</tr>
<tr>
<td>Pentium 4 @ 2 GHz</td>
<td>8K</td>
<td>512K</td>
<td>-</td>
<td>Intel Fortran Compiler 7.1</td>
</tr>
</tbody>
</table>
Performance Improvement for Different Architectures

![Graph showing performance improvement for different architectures.](image)