Intelligent Compilation

John Cavazos

Computer & Information Sciences Department
University of Delaware
Current Auto-tuning Systems

- Small domain-specific kernels
  - Typically NOT whole applications
- Large effort
  - Integrate a compiler into your application
- Empirical data thrown away
  - Re-tune to each new architecture/program
Intelligent Compilers

- Auto-tuning generates empirical data
- Machine learning uses data to build models
- Compiler *learns* to optimize

Applications

Intelligent Compiler
(Auto-tuning + Machine Learning)

Multicore Environment

Auto-tuning feedback
Intelligent Compilers

- Learned performance models predict:
  - Which optimizations to apply?
  - What order to apply them?
  - How to apply each optimization?
Applying Machine Learning

- Phrase problem as machine learning problem
- Determine inputs and outputs of ML model
- Generate training data
- Train and test model
  - Learning algorithms may require “tweaking”
Case Study

- Whole Program Optimization
- Improve highly-tuned commercial compiler
- Use performance counter characterization

• John Cavazos et al., *Rapidly Selecting Good Compiler Optimizations using Performance Counters*, CGO 2007
Using Performance Counters

- **Model Input**
  - Aspects of programs captured with perf. counters

- **Model Output**
  - Set of optimizations to apply
  - Automatically construct model
  - Model predicts optimizations to apply
### Performance Counters

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Avg Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPU_IDL</td>
<td>Floating Unit Idle</td>
<td>0.473</td>
</tr>
<tr>
<td>VEC_INS</td>
<td>Vector Instructions</td>
<td>0.017</td>
</tr>
<tr>
<td>BR_INS</td>
<td>Branch Instructions</td>
<td>0.047</td>
</tr>
<tr>
<td>L1_ICH</td>
<td>L1 Icache Hits</td>
<td>0.0006</td>
</tr>
</tbody>
</table>
Characterization of mcf
Characterization of mcf

Number of memory accesses per instruction larger than average
Training Data From 57 Benchmarks

- Search for best optimization sequences
  - Enumerate *many* optimization sequences
- Training data
  - Inputs – Performance counters data
  - Outputs – Best optimizations for program

<table>
<thead>
<tr>
<th>programs</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>.108; .25; 0; 0; ...; .08; 0; 0;</td>
<td>1; 0; 1; 1; ... 1; 1; 1; 0</td>
</tr>
<tr>
<td>bar</td>
<td>.93; .21; 0; 1; ...; .50; 0; 0;</td>
<td>1; 1; 0; 0; ... 1; 0; 0; 1</td>
</tr>
<tr>
<td>...</td>
<td>......</td>
<td>....</td>
</tr>
</tbody>
</table>
Training Performance Counter Model

**Input:** Performance counter values for a benchmark.

**Output:** Best Transformations for benchmark
Using Performance Counter Model

New program interested in obtaining good performance.
Using Performance Counter Model

Baseline run to capture performance counter values.
Using Performance Counter Model

Feed performance counter values to model.
Using Performance Counter Model

Model outputs distribution used to generate sequences
Using Performance Counter Model

Optimization sequences drawn from distribution.
Performance Counter Model

- Trained on data from auto-tuning
  - 500 evaluations for each benchmark
- Leave-one-out cross validation
  - Train on N-1 benchmarks
  - Test on Nth benchmark
- Logistic Regression
Logistic Regression

- **Variation of ordinary regression**

- **Inputs**
  - Continuous, discrete, or a mix
  - 50 performance counters
    - All normalized to cycles executed

- **Outputs**
  - Restricted to two values (0,1)
  - Probability optimization is beneficial
Experimental Methodology

- PathScale (industrial-strength compiler)
  - Compare to highest optimization level
  - Control 121 compiler flags
- AMD Athlon processor
  - Real machine; Not simulation
- 57 benchmarks
Evaluated Search Strategies

- Combined Elimination [CGO 2006]
  - Pure search technique
    - Eliminate negative optimizations in one go
  - Out performed other pure search techniques

- Performance Counter (PC) Model
Obtained > 25% on 7 programs, 17% over highest opt
Conclusion

- Machine learning is successful
  - Out-performs hand-tuned compiler models
- Simple code features can drive optimizations
  - ML determines which features are important
  - Optimizations applied only when beneficial