Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning

Sameer Kulkarni & John Cavazos
Intro

- Why we need code optimization?
  - Prog. Language design flaw
    - e.g. goto statement in C/Cpp
  - People are evil and chaotic
    - e.g. define unused variables inside a loop
Intro

- What do we have so far?
  - Optimization option with fixed order
Intro

- What do we have so far?

<table>
<thead>
<tr>
<th>OptKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optimization Level O0</td>
</tr>
<tr>
<td>CSE</td>
<td>Local common sub expression elimination</td>
</tr>
<tr>
<td>CNST</td>
<td>Local constant propagation</td>
</tr>
<tr>
<td>CPY</td>
<td>Local copy propagation</td>
</tr>
<tr>
<td>SA</td>
<td>CFG Structural Analysis</td>
</tr>
<tr>
<td>ET</td>
<td>Escape Transformations</td>
</tr>
<tr>
<td>FA</td>
<td>Field Analysis</td>
</tr>
<tr>
<td>BB</td>
<td>Basic block frequency estimation</td>
</tr>
<tr>
<td></td>
<td>Optimization Level O1</td>
</tr>
<tr>
<td>BRO</td>
<td>Branch optimizations</td>
</tr>
<tr>
<td>TRE</td>
<td>Tail recursion elimination</td>
</tr>
<tr>
<td>SS</td>
<td>Basic block static splitting</td>
</tr>
<tr>
<td>SO</td>
<td>Simple optimizations like Type prop, Bounds check elim, dead-code elim, etc.</td>
</tr>
<tr>
<td></td>
<td>Optimization Level O2</td>
</tr>
<tr>
<td>LN</td>
<td>Loop normalization</td>
</tr>
<tr>
<td>LU</td>
<td>Loop unrolling</td>
</tr>
<tr>
<td>CM</td>
<td>Coalesce Moves</td>
</tr>
</tbody>
</table>
Intro

- What do we have so far?
  - Optimization option with fixed order
  - Genetic algorithm that takes care of order
Results from GA
Problem with GA

- Expensive searching time for best result
- No fine-grain method level optimization
  - Unless you have method level timer
Possible solutions

- Predict the complete sequence
- Predict the current best optimization
Possible solutions

- Predict the complete sequence
- Predict the current best optimization
  - Markov Property
NEAT

- Neuro-Evolution of Augmenting Topologies
  - GA with ANN flavor
The ANN of NEAT

- Takes feature as input, predict best optimization to apply

- Sometimes the second best option to avoid dead loop
The ANN of NEAT

- Feature generation
- Apply neural network
  - One of the possible optimizations that can be applied
  - Optimization predicted to be the best by the neural network
  - Stop applying any more optimizations
- Multiple iterations of feature generation, neural network, prediction, and optimizations
Implementation

- 60 ANNs each generation
- 300 generations
- Trained on seven benchmarks from the Java Grande benchmark suite
- Tested on SPECjvm98, SPECjvm2008, and DaCapo
Result from NEAT
Result from NEAT

![Graph showing Optimization Level O3 speedup normalized by O3 for various workloads such as SPECjvm 98, SPECjvm 2008, and DaCapo. The graph compares Total Time and Running Time for different benchmarks and average values.](image-url)
GA vs NEAT

- Costs for training GAs and NEAT
  - GA : 11.4 days
  - NEAT : 13.2 days
  - GA per benchmark : 70 days
GA vs NEAT

![Diagram: Speedups vs Method importance]

- **Y-axis:** Speedup
- **X-axis:** No. of methods covering 60% of running time

Legend:
- GA Speedup
- NEAT Speedup