Using Support Vector Machines to Learn How to Compile a Method

Ricardo Nabinger Sanchez, J. Nelson Amaral, Duane Szafron
University of Alberta, Edmonton, AB, Canada
Marius Pirvu, Mark Stoodley
IBM Toronto Software Laboratory, Markham, ON, Canada

March 2008, Xangrilá, RS. Ricardo Sanchez
May 2009-August 2009: Ricardo spends the Summer working with mainly with Marius Pirvu at the IBM Toronto Software Laboratory

October 2008, IBM Toronto Software Laboratory, Markham, ON, Kevin Stoodley, Mark Stoodley, and Marius Pirvu: Can we use machine learning to improve compilation decisions in Testarossa?

November 2008: University of Alberta, Edmonton, AB, Canada
Duane Szafron, Michael Bowling, Ricardo Sanchez
We should try Support Vector Machines.
The Research Question

Can Support Vector Machines (SVMs) improve on the selection of code transformations done by human developers?

Characterize methods using *features*.

Learn to associate features with compilation strategies.

Strategies can be selected on a per-method basis.
Testarossa

- .class
- Intermediate Language Generator
- Compilation Control
- Optimizer
- Code Generator
- Compiled Methods
  - s390
  - MIPS
  - x86
  - ARM
  - PPC
  - ⋅⋅⋅
Support Vector Machines

A parameter $C$ in the implementation of the SVM specifies the maximum separation margin.
SVMs in Testarossa

• 51 features to describe each method
  – 51-dimension space search
• More than 70 code transformations
  – More than $2^{70}$ classes
• Why not non-linear kernels
  – Data is already highly dimensional
  – No need to project it to higher dimensions
Data Collection

Interpreted Methods

Virtual Machine

Compiled Methods

Compilation Plan

Modifier Queue

Modifier

Just-In-Time Compiler

Strategy Control
Measuring Time

call foo()

Enter hook

In-method execution

return

Exit hook

Processor ticks (TSC)

Time-Stamp Counter

\[ t_{in} \rightarrow T \rightarrow t_{out} \]
Goal

Method Features
- Many-Iteration Loops?
- Allocates Dynamic Memory?
- Virtual Method Overridden?
- Uses Floating Point?
- Number of Non-Scalar Objects
- Number of Long doubles
- Number of int
- ...

Machine-Learned Model

Code Transformations
- Constant Folding?
- Partial Redundancy Elimination?
- Loop Unrolling?
- Tree Simplification?
- Dead Tree Elimination?
- Scalar Value Expansion?
- Method Specialization?
- ...

Ranking Plans

Let \((i,p,h)\) represent a method \(i\) compiled with a compilation plan \(p\) at a level of hotness \(h\):

\[
\text{Value}(i,p,h) = \frac{\text{Total running time of all invocations of } (i,p,h)}{\text{Number of invocations of } (i,p,h)} + \frac{\text{Compilation time for } (i,p,h)}{\text{Threshold for } h}
\]

For each method \(i\), select the top \(t\) plans for training of the SVM.

The valued of the lowest plan must be at least \(f\%\) of the best.

In this research \(t = 3\), and \(f = 95\%\).
Using Learned Model

Interpreted Methods

Virtual Machine

Compiled Methods

Compilation Plan

Modifier
Queue

Modifier

00110100011001..001

00110000011000..000

00110000011000..000

Just-In-Time Compiler

Strategy Control

Learned Model
Socket-Based Communication Between Compiler and Model

Used *named pipes* (Unix) to communicate between Compiler and Model
## Data Set Sizes

<table>
<thead>
<tr>
<th>Compilation Level</th>
<th>Merged Data</th>
<th>Ranked Data (training)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data Instances</td>
<td>Unique Classes</td>
</tr>
<tr>
<td>Cold</td>
<td>1,551,545</td>
<td>1,421,717</td>
</tr>
<tr>
<td>Warm</td>
<td>1,577,157</td>
<td>1,455,947</td>
</tr>
<tr>
<td>Hot</td>
<td>2,543,564</td>
<td>2,229,364</td>
</tr>
</tbody>
</table>
Experimental Platform

• AMD Blade Server
  – 16 nodes
    • 2 Quad-Core Opteron/Node
    • 2 GHz
    • 8 GiB of RAM
    • 20 GiB swap space
    • CentOS GNU/Linux

• Development version of Testarossa
StartUp × Throughput Performance

StartUp Performance:
- Start JVM
- Finish one iteration of Benchmark

Throughput Performance:
- Start JVM
- Finish ten iterations of Benchmark
StartUp Performance (SPECjvm98)
Compilation Time Reduction for StartUp (SPEC jvm98)
Throughput Performance (SPECjvm98)
StartUp Performance DaCapo

DaCapo 9.12 Performance (Start-up)

Relative Performance vs. Testarossa

- H1 (co, db, mp, mt)
- H2 (co, db, mp, rt)
- H3 (co, db, mt, rt)
- H4 (co, mp, mt, rt)
- H5 (db, mp, mt, rt)
Throughput DaCapo

DaCapo 9.12 Performance (10 iterations)
Influence of Inlining

For the previous performance results we collected method features and applied the model before inlining.

Inlining may change method features significantly.

What would the results be if method features were measured after inlining?
StartUp Performance (SPECjvm98)

Before Inlining

After Inlining
JavaC StartUp Performance

Jess StartUp Performance
Throughput Performance (SPECjvm98)

Before inlining

After inlining

Throughput Performance (SPECjvm98)
javac
Throughput

jess
Throughput
What have we learned?

• **Overall**: SVM-based models outperform Testarossa’s heuristics for start-up performance.
  – But it underperforms Testarossa for throughput performance.

• **Surprising**: significant reduction in compilation time.

• **Puzzling**: Collecting method features after inlining did not yield greater performance gains.

• **Pleasantly positive**: model generalized from SPEC benchmarks to DaCapo.