Phase-based Adaptive Recompilation in a JVM

Dayong Gu    Clark Verbrugge

Sable Research Group, School of Computer Science
McGill University, Montréal, Canada
{dgu1, clump}@cs.mcgill.ca

April 7, 2008
Outline

1. Motivation
2. Hardware Information Analysis
3. Adaptive Recompilation
4. Conclusions and Future Work
Motivation

Java Programs

JVM
(Virtual Hardware)
Motivation

Hardware Impact

- The impact of hardware on program behaviour can be significant
- Strong correlation exists: hardware performance $\leftrightarrow$ program behaviour

Hardware Event Counters

- Hardware counters widely exist in modern processors
- Accessible from software libs: PAPI, PMAPI,PCL,...
**Motivation**

**Hardware Impact**
- The impact of hardware on program behaviour can be significant
- Strong correlation exists: hardware performance $\leftrightarrow$ program behaviour

**Hardware Event Counters**
- Hardware counters widely exist in modern processors
- Accessible from software libs: **PAPI**, **PMAPI**, **PCL**, ...

**Use hardware event data to improve adaptive optimizations in JVM**
There are different types of program phases
There are different types of program phases

<table>
<thead>
<tr>
<th>Flat Phases</th>
<th>Periodic Phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>refer to the contiguous intervals where show a stable, flat performance on a type of sampled, profiling data</td>
<td>long term repetitions in execution, showing unstable performance but sharing similar patterns</td>
</tr>
</tbody>
</table>
There are different types of program phases

**Flat Phases**
refer to the contiguous intervals where show a stable, flat performance on a type of sampled, profiling data

**Periodic Phases**
long term repetitions in execution, showing unstable performance but sharing similar patterns

Detect this long term periodic phases from hardware data and employ the phase information in adaptive optimizations
Contributions

- Highlight the hardware impact on Java program execution
- Develop online **pattern** creation algorithm to represent hardware event
- Detect long term periodic **phases** from hardware patterns
- Implement an **adaptive recompilation strategy** using phase information
Hardware Event Data Example

![Graph showing hardware event data over time](image-url)
Recurrent Phases

![Graph showing Recurrent Phases over time](image)

- **Hardware Event**
- **Time**

Sable Group, McGill University
Hardware Pattern

![Hardware Pattern Graph](image-url)

- Hardware Event
- Time

Sable Group, McGill University
Hardware Pattern
Hardware Pattern

![Graph showing hardware pattern over time]

Hardware Event

Time

Sable Group, McGill University

15/33
Pattern Building Algorithm (Simplified)

1. Get HW data D
2. Get the variation: D Vs former Data
3. Large enough?
   - Yes: Begin a new pattern building
   - No: Continue the current pattern building; Add info. about D
4. Pattern Finish?
   - Yes: Report pattern
   - No: Conditions:
     1. Small variations in a row
     2. Pattern reaches the max length
     3. Meet a more important variation (stop in the middle)
(1) Hardware Data
Pattern Building Example

(1) Hardware Data

(2) Variation
(1) Hardware Data

(2) Variation

(3) Level

Sable Group, McGill University
Pattern Building Example

(1) Hardware Data

(2) Variation

(3) Level

(4) Pattern

Sable Group, McGill University
Patterns are stored and analyzed
The number of occurrences determines the **hotness** of a pattern
The hottest pattern is used to represent the current program **phase**
The **phase information** is used to achieve a better adaptive hot method recompilation strategy in Jikes RVM
Recompilation in Jikes RVM

- Get method samples
- Compute the *Past* time in a method
- Estimate the compilation cost $C_i$ to optimization level $i$

Recompile to level $i$, if $(SpeedupRate_i \times Past) > C_i$
Recompilation in Jikes RVM

- Get method samples
- Compute the \textit{Past} time in a method
- Estimate the compilation cost \(C_i\) to optimization level \(i\)

Recompile to level \(i\), if \((\text{SpeedupRate}_i \times \text{Past}) > C_i\)
Assume \textit{Future} \equiv \textit{Past}
Optimization Opportunity

Sable Group, McGill University
Recompilation is not free, cannot always be aggressive
Recompilation is not free, cannot always be aggressive

Our approach:

<table>
<thead>
<tr>
<th>Program State</th>
<th>Sampled Methods</th>
<th>Hardware Event Behaviour</th>
<th>Recompilation Aggressiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>The “Beginners”</td>
<td>No recurrence of patterns</td>
<td>Low</td>
</tr>
<tr>
<td>Young</td>
<td>Important methods</td>
<td>Recurrences of patterns</td>
<td>High</td>
</tr>
<tr>
<td>Mature</td>
<td>Optimized methods</td>
<td>Less fresh patterns</td>
<td>Moderate</td>
</tr>
<tr>
<td></td>
<td>Less important methods</td>
<td>More old patterns</td>
<td></td>
</tr>
<tr>
<td>Rejuvenated</td>
<td>New important methods</td>
<td>More fresh patterns again</td>
<td>High</td>
</tr>
</tbody>
</table>
Recompilation is not free, cannot always be aggressive

Our approach:

<table>
<thead>
<tr>
<th>Program State</th>
<th>Sampled Methods</th>
<th>Hardware Event Behaviour</th>
<th>Recompilation Aggressiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>The “Beginners”</td>
<td>No recurrence of patterns</td>
<td>Low</td>
</tr>
<tr>
<td>Young</td>
<td>Important methods</td>
<td>Recurrences of patterns</td>
<td>High</td>
</tr>
<tr>
<td>Mature</td>
<td>Optimized methods</td>
<td>Less fresh patterns</td>
<td>Moderate</td>
</tr>
<tr>
<td></td>
<td>Less important methods</td>
<td>More old patterns</td>
<td></td>
</tr>
<tr>
<td>Rejuvenated</td>
<td>New important methods</td>
<td>More fresh patterns again</td>
<td>High</td>
</tr>
</tbody>
</table>

*Future ≠ Past*

- Fixed aggressiveness $\Rightarrow$ Adaptive aggressiveness
Optimization Opportunity

- Optimize code to higher levels earlier
- Possibly save recompilation overhead for intermediate levels
- Save unnecessary recompilation for the “beginners”
Opt. Level

Samples

Opt0
Opt1
Opt2
Opt3

Sable Group, McGill University
Online Optimization

Opt. Level

Samples

Sable Group, McGill University
Offline Head Space Study
Experimental Set-up

- **Benchmarks:**
  - SPECjvm98 suite
  - Dacapo benchmarks: ANTLR, BLOAT, FOP, PMD, XALAN
  - SOOT and PSEUDOJBB

- Test on Athlon 1.4G, 1GB memory, Debian Linux kernel 2.6.9

- Average of the middle 11 in 15 runs
Recompilation Results

Whole Execution Time Reduction
(incl. all overhead)

**Offline**  Use training runs  8.7%  21%
**Online**  Use HW pattern info.  4.4%  18%

Sable Group, McGill University
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Overhead (%)</th>
<th>Benchmark</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>2.02</td>
<td>antlr</td>
<td>2.12</td>
</tr>
<tr>
<td>db</td>
<td>1.39</td>
<td>bloat</td>
<td>1.65</td>
</tr>
<tr>
<td>jack</td>
<td>1.71</td>
<td>fop</td>
<td>1.69</td>
</tr>
<tr>
<td>javac</td>
<td>1.13</td>
<td>pmd</td>
<td>1.70</td>
</tr>
<tr>
<td>jess</td>
<td>0.49</td>
<td>xalan</td>
<td>1.07</td>
</tr>
<tr>
<td>mpegaudio</td>
<td>1.76</td>
<td>soot</td>
<td>1.85</td>
</tr>
<tr>
<td>mtrt</td>
<td>0.82</td>
<td>PseudoJbb</td>
<td>0.77</td>
</tr>
<tr>
<td>raytrace</td>
<td>1.30</td>
<td><strong>Average</strong></td>
<td><strong>1.43</strong></td>
</tr>
</tbody>
</table>

The “overhead” includes all sources from hardware monitoring, pattern construction, information analysis, and building control events to adaptive engine
Understanding repetitive program behaviour and exploring phases in program execution is important.

We implemented a technique for determining program phases from hardware data.

We applied the phase information in adaptive recompilation.

Hardware information can be used in a wide range of areas:
- Runtime profiling, selecting GC points
- Program understanding, system reconfiguration, instruction/data relocation and prefetch ...
Future Work

- Test other hardware events/combinations
- Use offline analysis results for repeatable executions
- Attach hardware variation with software structures
- Advanced static analysis can be helpful
- Develop other adaptive applications
Thank you!

Questions?