JIT Compilation Policy For Modern Machines

Prasad A. Kulkarni
OOPSLA'11

Presentation: Piotr Górski
JIT compilation

a method to improve the runtime performance of computer programs written in languages supporting „compile-once, run-anywhere” model for code generation and distribution
Problems

• JIT compilation contributes to the overall execution time of the application
• Compilation policies need to carefully tune if and when different program regions are compiled to achieve the best performance
Existing policies

• Selective compilation
• Online profiling to detect *hot methods*
  o Method counters
  o Interrupt-timer-based sampling
  o Assumption that current hot methods will remain hot in the future
  o Delays
    • Time to reach the compilation threshold
    • Time spent in compilation queue

• Proposed approaches
  o Offline profiling
  o Dynamically determining loop iteration bounds to predict future hot methods
Contribution of the paper

- Quantifying the impact of altering *if* and *when* methods are compiled on application performance
- Demonstrating the effect of multiple compiler threads on average program performance for single-core machines
- Explaining the impact of different JIT compilation strategies on multi-core machines
- Showing the benefit of prioritizing method compiles on program performance
## Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>#Methods</th>
<th>Name</th>
<th>#Methods</th>
<th>Name</th>
<th>#Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>.201_compress_100</td>
<td>517</td>
<td>compiler.compiler</td>
<td>3195</td>
<td>avrrora_default</td>
<td>1849</td>
</tr>
<tr>
<td>.201_compress_10</td>
<td>514</td>
<td>compiler.sunflow</td>
<td>3082</td>
<td>avrrora_small</td>
<td>1844</td>
</tr>
<tr>
<td>.202_jess_100</td>
<td>778</td>
<td>compress</td>
<td>960</td>
<td>batik_default</td>
<td>4366</td>
</tr>
<tr>
<td>.202_jess_10</td>
<td>759</td>
<td>crypto.aes</td>
<td>1186</td>
<td>batik_small</td>
<td>3747</td>
</tr>
<tr>
<td>.205_raytrace_100</td>
<td>657</td>
<td>crypto.rsa</td>
<td>960</td>
<td>eclipse_default</td>
<td>11145</td>
</tr>
<tr>
<td>.205_raytrace_10</td>
<td>639</td>
<td>crypto.signverify</td>
<td>1042</td>
<td>eclipse_small</td>
<td>5461</td>
</tr>
<tr>
<td>.209_db_100</td>
<td>512</td>
<td>mpegudio</td>
<td>959</td>
<td>fop_default</td>
<td>4245</td>
</tr>
<tr>
<td>.209_db_10</td>
<td>515</td>
<td>scimark.fft.small</td>
<td>859</td>
<td>fop_small</td>
<td>4601</td>
</tr>
<tr>
<td>.213_javac_100</td>
<td>1239</td>
<td>scimark.lu.small</td>
<td>735</td>
<td>h2_default</td>
<td>2154</td>
</tr>
<tr>
<td>.213_javac_10</td>
<td>1211</td>
<td>scimark.monte.carlo</td>
<td>707</td>
<td>h2_small</td>
<td>2142</td>
</tr>
<tr>
<td>.222_mpegaudio_100</td>
<td>659</td>
<td>scimark.sor.small</td>
<td>715</td>
<td>jython_default</td>
<td>3547</td>
</tr>
<tr>
<td>.222_mpegaudio_10</td>
<td>674</td>
<td>scimark.sparse.small</td>
<td>717</td>
<td>jython_small</td>
<td>2070</td>
</tr>
<tr>
<td>.227_mtrt_100</td>
<td>658</td>
<td>serial</td>
<td>1121</td>
<td>luindex_default</td>
<td>1689</td>
</tr>
<tr>
<td>.227_mtrt_10</td>
<td>666</td>
<td>sunflow</td>
<td>2015</td>
<td>luindex_small</td>
<td>1425</td>
</tr>
<tr>
<td>.228_jack_100</td>
<td>736</td>
<td>xml.transform</td>
<td>2592</td>
<td>lusearch_default</td>
<td>1192</td>
</tr>
<tr>
<td>.228_jack_10</td>
<td>734</td>
<td>xml.validation</td>
<td>1794</td>
<td>lusearch_small</td>
<td>1303</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pmd_default</td>
<td>3881</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pmd_small</td>
<td>3058</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sunflow_default</td>
<td>1874</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sunflow_small</td>
<td>1826</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tomcat_default</td>
<td>9286</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tomcat_small</td>
<td>9189</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>xalan_default</td>
<td>2296</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>xalan_small</td>
<td>2277</td>
</tr>
</tbody>
</table>
Effect of JIT compilation

Graph 1: Performance at each stage (interpreter performance) vs. execution count threshold.
Graph 2: Compiled/total methods vs. execution count threshold.
Effect of early compilation
JIT compilation on single-core machines

One thread
JIT compilation on single-core machines

Multiple threads
JIT compilation on multi-core machines
Simulation of multi-core VM execution on single-core processor

(a) Multi-core execution

Core1: A A A A A A A A A
Core2: C1 C1 C1 C1
Core3: C2 C2 C2
Core4: C3

(b) Single-core simulation of multi-core execution

Core1: A C1 A C1 C2 C3 A C1 C2 A A A C1 C2 A A C1 A
JIT compilation on many-core machines
Priority-based compiler queue

(a) FIFO-priority

(b) Ideal Priority

single-core machine configuration
Priority-based compiler queue

(a) FIFO-priority

many-core machine configuration

(b) Ideal Priority
Priority-based compiler queue
heuristic approach

• Offline profiling is rarely acceptable
• Need to devise a dynamic priority scheme

\[
\text{priority} = \frac{\text{method execution count}}{\text{current global count}} - X
\]
Priority-based compiler queue
heuristic approach

Single-core machine configuration

Many-core machine configuration

- Single-core machine configuration
  - graph showing performance metrics with different thread counts (CT) for various configurations.
- Many-core machine configuration
  - A graph showing performance metrics for different thread counts (CT) and configurations, indicating a decrease in performance as the number of threads increases.
Conclusions

• On single-core machines the same compilation threshold achieves the best overall program performance with a single and multiple compiler threads, and regardless of the priority queue algorithm.

• Multi-core and many-core hardware can enable more aggressive JIT compilation policies, which can be benefactorly to performance ...

• ... but achieving such benefits requires accurate assignment of method priorities.
Thank you