Reducing Compilation Overhead in J9/TR

Marius Pirvu, Derek Inglis, Vijay Sundaresan
Agenda

- Adaptive compilation in the TR JIT
- Class Load Phase
- Asynchronous compilation
- Limiting the negative effect of very long compilations

Ongoing work
- Improvements to async compilation
- AOT
Adaptive Compilation in TR JIT

- Methods start out being interpreted
- After N invocations methods get compiled at ‘warm’ level
- Sampling thread used to identify hot methods
- Methods may get recompiled at ‘hot’ or ‘scorching’ levels
- Transition to ‘scorching’ goes through a temporary profiling step
Performance

Comparison of compilation time for different optimization levels

- **Interpreted**
- **noOpt**
- **cold**
- **warm**
- **hot**
- **scorching**
- **Adaptive**

Compilers and benchmarks:
- **compress**
- **jess**
- **db**
- **javac**
- **mpegaudio**
- **mtrt**
- **jack**
- **SPECjbb**

Compilation time (sec)
Performance comparison of different optimization levels

- compress
- jess
- db
- javac
- mpegaudio
- mtrt
- jack
- SPECjbb

- Interpreted
- noOpt
- cold
- warm
- hot
- scorching
- Adaptive
How About Applications Without Hotspots

- WebSphere AppServer startup
  - Very flat profile

<table>
<thead>
<tr>
<th></th>
<th>No Adaptive Compilation</th>
<th>With Adaptive Compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Methods compiled</strong></td>
<td>Warm=36700</td>
<td>Warm=2750, hot=11</td>
</tr>
<tr>
<td><strong>Time spent compiling</strong></td>
<td>115 sec</td>
<td>17.3 sec</td>
</tr>
<tr>
<td><strong>Startup-time</strong></td>
<td>161 sec</td>
<td>21.5 sec</td>
</tr>
</tbody>
</table>
Class-Load-Phase

- Intuition: Methods compiled during startup phase may not be important during application run phase
- Detect phases when class loading is intense
- Reduce optimization level to “cold” during such phases

![Effect of ClassLoadPhase](image)
Asynchronous Compilation

- **Synchronous compilation**
  - Application thread places compilation request and blocks waiting for the compilation to finish

- **Asynchronous compilation**
  - Application thread does not wait for the compilation result

- **JIT compilations performed on a separate compilation thread**
Asynchronous Compilation

- **Implementation**
  - Synchronous compilation still needed in some cases (e.g. pre-existence)
  - Synchronous and asynchronous compilation must coexist
  - Queue of compilation requests

- **Advantages**
  - Takes advantage of available processors on SMP machines by increased parallelism
  - Allows performance improvements in uniprocessors by changing compilation thread priority
  - Allows reordering of compilation requests
Performance Results – Short Running Apps.

- Asynchronous compilation on SMP reduces execution time of short benchmarks
Limiting Negative Effect of Long Compilations

- Compilations may impede GC operation
  - GC requires exclusive VM access
  - Cannot allow class unloading while compilation in progress \( \Rightarrow \) Compilations require VM access

- Solution
  - Compilation thread periodically releases and reacquires VM access allowing GC to cut-in
  - Upon re-acquiring VM access, check if GC unloaded any classes
  - If classes were unloaded, abort current compilation and retry
Ongoing Improvements to Asynchronous Compilation

▪ Idea
  – Use thread priorities to smooth out the effects of compilation - effectively interleave compilation with execution

▪ Implementation
  – Don’t use more than X% CPU for compilation
  – Use the queue of methods as a buffer
    – accumulate work during periods of heavy utilization
    – solve the backlog when CPU is lightly used or idle (due to IO for instance)
  – Prioritize compilation requests in the queue
Performance - Uniprocessors

![Graph showing normalized execution time for SPECjvm98, first run with Sync and Dynamic categories for different benchmarks such as compress, jess, db, javac, mpegaudio, mtrt, jack.]
Ahead Of Time (AOT) Compilation

- Using the JIT as a static compiler
- Fully compliant code
- Used by J2ME customers to decrease footprint by eliminating the JIT
- Will be used by the Real Time offering to eliminate the possibility of nondeterministic behaviour introduced by the JIT
- Experimenting with combining AOT and JIT compilation to improve startup times
Eclipse 3.0.1 Startup Times

- HotSpot Server: 7.13s
- HotSpot Client: 5.74s
- J9 default: 5.08s
- J9 with Shared Classes: 4.53s
- J9 with Shared Classes and AOT: 4.03s

Hardware: 1P@1.2GHz Pentium3 M, Windows32 Options: -Xmx512m -Xms512m for all tests