IBM Java Just-In-Time Compiler

Theresa Tai
March 2, 2005 (Session 8379)
zSeries New Technology Center
Poughkeepsie, New York
ttai@us.ibm.com
Objectives

- Understanding the IBM Just In Time Compiler (JIT)
- The Mixed Mode Interpreter (MMI)
- Hints and Tips for isolating JIT problems
JIT Overview

- What does JIT compiler do?
  - It dynamically generates machine code for frequently used bytecode sequences in Java applications while they are running

- Purpose
  - To improve performance by optimizing machine code execution

- Value
  - Without JIT
    - JVM starts up rather quickly but runs slowly
  - With JIT
    - JVM starts up with slight delayed but improve overall performance

- IBM SDK includes the JIT component

- IBM JVM is running with JIT enabled by default

- A comprehensive set of runtime and debug options available
JIT Technology

- **Problem Statement**
  - The JVM interpreting bytecodes can’t match the performance of native applications with machine code
  - Need to improve the performance of JVM startup (interpreting/compiling) and Java applications over the life of JVM

- **Solution**
  - A compiler that will allow JVM to start reasonably quickly
  - JIT **code optimization** processes
  - Improve overall Java applications performance

- **The Front-end**
  - Method analysis and optimization is common to all platforms

- **The Back-end**
  - Machine code generation is z/OS specific

- **The Other Half of JIT**
  - The introduction of MMI (Mixed Mode Interpreter)
MMI Technology

❖ Purpose:
   ▪ Designed to optimize the startup time and runtime performance of Java applications
   ▪ Using a fast Assembler bytecode interpreter

❖ Value
   ▪ MMI only JIT compile Java methods that are frequently used or execute over a long time
   ▪ Infrequently used methods which may not be compiled at all

❖ Threshold value
   ▪ Default threshold count 2000
   ▪ When count = 0, the method is JIT compiled
   ▪ MMI behavior can be changed by adjusting the threshold count value
   ▪ Must evaluate and track results to achieve optimal performance level
The JIT & MMI Operating Modes

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>JIT</th>
<th>MMI</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>ON</td>
<td>ON</td>
<td>Interpret Method and JIT compile, JVM start reasonably quickly, improve overall application performance</td>
</tr>
<tr>
<td>Interpreter Disabled</td>
<td>ON</td>
<td>OFF</td>
<td>JIT compile all methods immediately, JVM starts slowly but performance is satisfactory</td>
</tr>
<tr>
<td>JIT Disabled</td>
<td>OFF</td>
<td>OFF</td>
<td>Always interpret methods, runs in interpretive mode only. The JVM starts up quickly, but runtime performance is poor</td>
</tr>
</tbody>
</table>

Note: When turning JIT off, the Interpreter does not invoke JIT translated code.
Hints & Tips on Isolating JIT Problems

- Does it fail if JIT is disabled? export JAVA_COMPILER=
- Does it fail if MMI is disabled? java -Xcomp ...

Compilation Failures
- Disable JIT and re-compile, if problem persist, not a JIT problem
- Remember to re-enable JIT
- Set IBM_MIXED_MODE_THRESHOLD=0 (Disabling MMI)

Selectively Disabling JIT Based on Conditional Code Points
- Investigate and carefully pick-n-choose JIT compile options
  - Narrowing down to a single option
- Mitigate performance hit all at once

Workarounds are far less intrusive than –Xint (no JIT)
- Skipping a method does not mean it doesn’t run – it just doesn’t get compiled
- Reducing opt level of a method has minimal impact
- Workarounds are designed to provide relief while we work on a fix
JIT Compile Option Groups

- **JITC_COMPILEOPT=NMMI2JIT**
  - MMI to JIT transfer process

- **JITC_COMPILEOPT=NINLINING**
  - Tells JIT not to inline code

- **JITC_COMPILEOPT=NQOPTIMIZE**
  - Disable all Quad optimizations

- **JITC_COMPILEOPT=NALL**
  - Generate native machine code without any of the optimizations (identify failing optimization by disabling all optimizations)

- **JITC_COMPILEOPT=NOTHER**
  - Uncategorized options

- **JITC_COMPILEOPT=NGLOBAL**
  - Not method-specific
Things You Want to Know About JIT

- You can not use non-IBM JIT with the IBM JVM
- JIT can not de-compile what is already JIT compiled code
- Do not replace the JIT that are packaged with the JVM
- You can set the JIT initial status only at JVM start-up time
- The JIT can be started up only at the same time as the JVM
- JIT is not part of JVM but it is loaded along with the JVM executables
- IBM continues to perfect the functions and features of JIT technology
Summary

- Tip: Avoid disabling JIT and/or MMI in a production environment unless it is absolutely necessary
- Tip: Follow “process of elimination” approach to isolate problems
- Tip: Use the JIT compile options, debug options
- Tip: Use the IBM JIT Diagnostic Guide for problem determination and rely on the IBM defect support staff for advice before turning off JIT
- Tip: If you require a quick startup for your applications and does not care much about runtime performance – Use the Xquickstart option

- Verify JIT
  - `java -version` (either JIT enabled or JIT disabled)
- -Xint option disables the JIT Compiler
  - Java -Xint class
- Use binary reduction to narrow down to failing method (later methods are most suspect so start by skipping second half)
- Dynamic slip trap
  - `JITC_DEBUGOPT=slip{class}{method(sig)}`
- JIT can produce a limit file listing compilations:
  ```java
  java -Xjit:verbose,vlog=compiling.out MyClass my args
  ```