Debugging the Java HotSpot VM

Just-in-time compilers

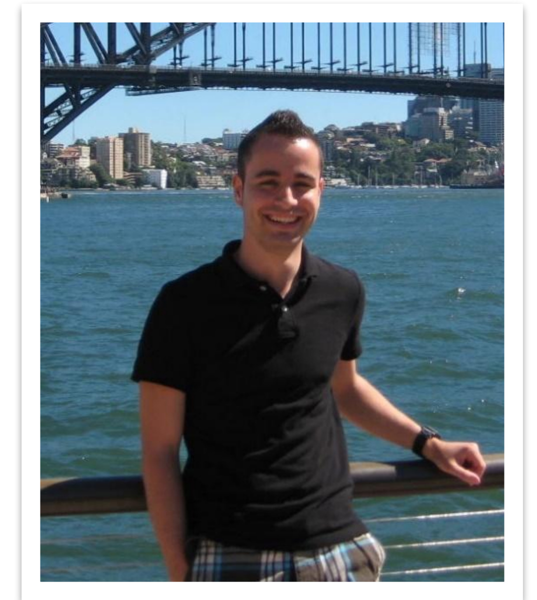
Tobias Hartmann
Compiler Engineer
Java Platform Group, Oracle

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

About me

- **Joined Oracle in 2014**
- **Software engineer in the Java HotSpot VM Compiler Team**
  - Based in Baden, Switzerland
- **German citizen**
- **Master's degree in Computer Science from ETH Zurich**
- **Worked on various compiler-related projects**
  - Currently working on future Inline Type support for Java



ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Compiler team bug triaging

- **All incoming bugs** and RFEs are **triaged**
  - Labels, affects and fix version, priority (ILW), links, …
  - Initial triaging done by compiler triaging team (Rahul, Tobias)
  - Signoff by SQE engineer
- **OpenJDK bugs and sustaining bugs are not triaged**
- **Differentiate between test and product bugs**
  - All targeted product bugs are immediately assigned
  - If you get one, **how to debug?**

https://wiki.se.oracle.com/display/JPG/JVM+Compiler+Team
https://wiki.se.oracle.com/display/JPG/Compiler+Bug+Triaging
https://wiki.se.oracle.com/display/JPGRM/ILW+and+Priority+Mapping+of+Bugs

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

How to debug the JIT compilers

- **Compilers are complex, unstable and hard to test**
  - Old bugs may be triggered by completely unrelated (even non-VM) changes
  - Several levels of abstraction: Java code, bytecode, IR, optimizations, code generation
- **The following slides show some tips and tricks**
  - Based on my experience
  - Not all steps are applicable to all problems
  - I‘m mostly using Ubuntu 20.04. on x86-64 with Eclipse

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

How to debug the JIT compilers

Outline

1) Analyze the log files ⬅
2) Try to reproduce the problem
3) Find the change that introduced the problem
4) Basic analysis
5) Advanced analysis

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

1) Analyze the log files

- **Look at the failure / test history in MDash**
  - Gather information from artifacts
  - Problem is project related? Ask experts for help.
- **hs_err_pid*.log contains lots of information**
  - Stack trace, current compilation, command line arguments, VM version, events, ...
  - Manually decode assembly instructions [1] or use hs_err decoder [2]
- **Test / application log file(s)**
  - jtreg, bigapps, console output, environment info
- **Core file**
  - Often contains not enough information

[1] https://www.onlinedisassembler.com/odaweb/
[2] http://jenkins.s0.javaplatfo1lhr.oraclevcn.com:999/kjdb_web/

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

1) Analyze the log files: Failure types

- **Wrong result**
  - Wrong numeric value, impossible exception thrown, wrong branch taken, …
  - Severe and usually hard to debug
  - Might be security vulnerability
- **Crash or assertion**
  - During compilation, in compiled code, while deoptimizing, …
- **Performance issue**
  - Regression, test time out, endless loop, code cache exhaustion, …
  - Might be test bug or expected regression (correctness vs. performance)

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

How to debug the JIT compilers

Outline

1) Analyze the log files
2) Try to reproduce the problem
3) Find the change that introduced the problem
4) Basic analysis
5) Advanced analysis

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

2) Try to reproduce the problem

- **Use exact same configuration**
  - Compilers are highly dependent on the architecture
  - VM build, host, test, command line arguments, jtreg version
  - Only then try to narrow problem down with own build or different flags
- **Write shell script to run test multiple times (days)**
  - Some failures are extremely rare
  - Use re-run command generated by test suite (for example, jtreg or UTE)
  - May require running multiple tests on the same VM (AgentVM mode)

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

2) Try to reproduce the problem

- **Find other instances of the same problem**
  - Search JIRA / Mach5 for error message, stack trace or test (suite)
  - Phone Home tool [1]
- **Check if C1 or C2 problem**
  - Try with `–Xint`, `-XX:TieredStopAtLevel=1` and `-XX:-TieredCompilation`
- **Replay compilation via `–XX:+ReplayCompiles`**
  - Requires replay_pid*.log generated during crash
  - Mighty tool, if it works (~60% of the cases)
  - Simplify replay file by removing inlining statements (can be scripted)

[1] http://phonehome.se.oracle.com:8080/Ph-Dashboard/PHSearchAction

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

2) Try to reproduce the problem

- **Only reproduces on a specific machine?**
  - Set `-XX:+PrintFlagsFinal` and compare output between machines
  - Could be due to different GC settings or processor features affecting all compilation phases due to ergonomic flag settings (for example, `-XX:UseAVX`)
  - C/C++ compiler bug or undefined behavior?
- **Try to write a reproducer**
  - May use inlining and profile information from replay file as guidance
  - Then simplify as much as possible
  - Helps with analysis and to avoid later regressions
  - Personal experience: **Don't give up early!** It may take a while but it's worth it.

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

How to debug the JIT compilers

Outline

1) Analyze the log files
2) Try to reproduce the problem
3) Find the change that introduced the problem
4) Basic analysis
5) Advanced analysis

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

3) Find the change that introduced the problem

- **Find build** that introduced the problem
  - Binary search with reproducer on VM builds
  - Be careful, other changes may only hide the problem!
- **Look at list of compiler changes** for that build

```
project = JDK AND resolution = Fixed AND component = hotspot AND Subcomponent =
compiler AND fixVersion = "9" AND "Resolved In Build" = bXX ORDER BY resolved DESC
```

  - May want to exclude test bugs and OpenJDK bugs

```
… AND (labels is EMPTY OR labels not in (teststabilization, testbug, noreg-self,
noreg-doc, openjdk))
```

- **Analyze the change** or ask author for help

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

How to debug the JIT compilers

Outline

1) Analyze the log files
2) Try to reproduce the problem
3) Find the change that introduced the problem
4) Basic analysis
5) Advanced analysis

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

4) Basic analysis: Simplification

- **Try to narrow problem down to a single compiled method**
  - Useful flags:
    `-XX:+PrintCompilation`, `-XX:+PrintInlining`, `-XX:CompileCommand=quiet`,
    `-XX:CompileCommand=exclude/dontinline/inline,*`
  - Use compiler directives for more fine-grained control
- **Simplify IR and generated code by disabling optimizations**
  - Example C2 flags to disable:
    `-XX:-OptimizePtrCompare`, `-XX:-OptoPeephole`, `-XX:LoopUnrollLimit=0`, `-XX:LoopMaxUnroll=0` ,
    `-XX:-SuperWordLoopUnrollAnalysis`, `-XX:-UseCountedLoopSafepoints`,
    `-XX:-UseLoopPredicate`, `-XX:-LoopUnswitching`, `-XX:-UseSuperWord`, `-XX:-SubsumeLoads`,
    `-XX:-OptimizeStringConcat`, `-XX:-SplitIfBlocks`, `-XX:-RangeCheckElimination` ,
    `-XX:-PartialPeelAtUnsignedTests`
  - Find more flags in `globals.hpp`, `c1/c2_globals.hpp`

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

4) Basic analysis: Hints

- **Inlined method is intrinsified?**
  - Disable intrinsic via `-XX:DisableIntrinsic=_Name`
- **Wrong numeric value in Java code?**
  - Could be a concurrency issue or a missing sign extension
  - Run with GC verification flags to find bad oops
- **Performance problem?**
  - Profile application to find responsible method (for example, by using JFR)
  - Enable debug output for C2 optimizations and compare to baseline
  - Compare generated assembly code to baseline

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

4) Basic analysis: Hints

- **Problem related to String operations?**
  - Disable **Compact Strings**

    `-XX:-CompactStrings`
  - Disable **Indify String Concat**

    `-XDstringConcat=inline` or `-XDstringConcat=indy` or `-XDstringConcat=indyWithConstants`
  - Disable **String intrinsics**

```
-XX:DisableIntrinsic=_toBytesStringU, _getCharsStringU, _inflateStringC,
_inflateStringB, _compressStringC, _getCharStringU, _compressStringB, _compareToL,
_compareToU, _compareToLU, _compareToUL, _putCharStringU, _arraycopy, _indexOfL,
_indexOfU, _indexOfUL, _indexOfIL, _indexOfIU, _indexOfIUL, _indexOfU_char,
_equalsL, _equalsU, _hasNegatives, _encodeByteISOArray
```

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

4) Basic analysis: Hints

- **Problem related to loop optimizations?**
  - Debug flags
    `-XX:+TraceLoopOpts -XX:+TraceSuperWord -XX:+TraceNewVectors`
  - Disable vectorization
    `–XX:-UseSuperWord`
  - Disable AVX
    `–XX:UseAVX=0`
  - Other flags
    `-XX:-RangeCheckElimination -XX:-PartialPeelLoop –XX:-UseLoopPredicate –XX:LoopMaxUnroll=0 –XX:-OptimizeFill …`

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

4) Basic analysis: Hints

- **Physical memory corruption?**
  - Unlikely if failure showed up on multiple machines
  - Check RAM on machine
- **VM binary corrupted?**
  - Compare binary to sane build
- **Static C/C++ compiler bug or undefined behavior?**
  - Try with different version or disable optimizations
  - Overflow in C/C++ code? For example, `long == int` on Windows.
- **OS / kernel bug**
  - INTJDK-7624851: corruption of XMM registers with some Linux kernel versions

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

How to debug the JIT compilers

Outline

1) Analyze the log files
2) Try to reproduce the problem
3) Find the change that introduced the problem
4) Basic analysis
5) Advanced analysis

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

5) Advanced analysis: Bytecodes

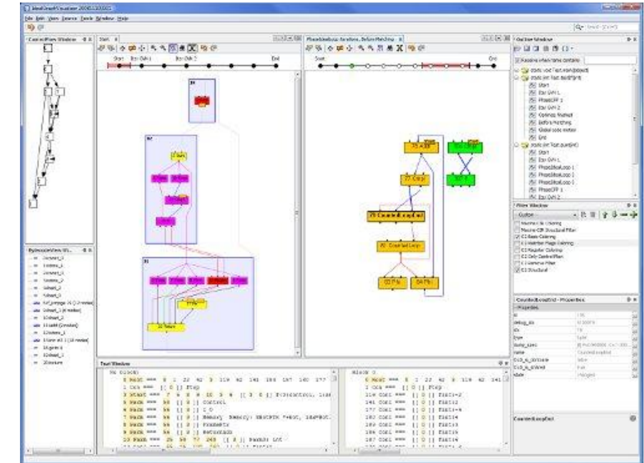
- **Print bytecodes during compilation**
  - Use `-XX:+CIPrintMethodCodes`
  - May use AsmTools (`jasm`) to create test from bytecodes
- **Disassemble class files**
  - `javap -c -p`
- **Debug method handles**
  - Use `-Djava.lang.invoke.MethodHandle.DUMP_CLASS_FILES=true`
- **Inspect object header layout**
  - http://openjdk.java.net/projects/code-tools/jol/

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

5) Advanced analysis: IR

- **Relies on simplification of reproducer in step 4)**
- **Dump ideal graph in text form**
  - Use `-XX:+PrintIdeal`
  - Hard to read but compact
- **Use Ideal Graph Visualizer**
  - Visual representation of IR
  - Supports several filters (customizable)
  - Use `-XX:PrintIdealGraphLevel=* -XX:PrintIdealGraphFile="foo.xml“`
- **Pro tip: Draw relevant parts on paper**



ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

5) Advanced analysis: Assembly

- **Opto assembly**
  - Requires debug build and supported by C2
  - Use `-XX:+PrintOptoAssembly`
- **Native assembly**
  - Requires `hsdis` disassembler to be on the library path
  - Use `-XX:+PrintAssembly`
- **Simplify assembly layout**
  - Use `-XX:-BlockLayoutRotateLoops -XX:-BlockLayoutByFrequency`

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

5) Advanced analysis: GDB

- **Start with GDB or attach it while JVM is already running**
  - Use `slowdebug` build
  - On error `-XX:+ShowMessageBoxOnError`
  - On exception `-XX:AbortVMOnException=java.lang.AssertionError`
- **Ignore SIGSEGV signals**
  - They are used for VM internal purposes (for example, implicit null checks)
  - `(gdb) handle SIGSEGV nostop noprint`
- **Inspect C2 graph**
  - Dump nodes: `(gdb) p node->dump(1)`
  - Use watchpoints to track changes. For example, watch `node->_in[0]`

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

5) Advanced analysis: GDB

- **Step through generated assembly**
  - Set breakpoint at nmethod entry point
  - Always use **hardware breakpoints** to avoid code modification
- **VM provides debug API**
  - See `debug.cpp`: `findpc(pc)`, `find_node(root, index)`
  - Methods can be called from GDB
- **Make sure that debug symbols are available**
  - Configure `--with-native-debug-symbols=external`
  - Unzip *.diz from existing build

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

5) Advanced analysis: rr

- **Records and deterministically replays the execution**
  - `rr java -version`
  - `rr replay`
- **Allows reverse execution/debugging**
  - `(rr) reverse-cont`
  - May require some `-XX:SuppressErrorAt=…` statements
  - **Reverse debug dying subgraphs** by incrementally watching node input array
- **Default flag values may change when executing with rr**
  - Set flags manually (check what changed with `-XX:+PrintFlagsFinal`)

http://rr-project.org/

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

More from my personal experience

- **It's not stupid if it works!**
  - Some failures are extremely intermittent and only reproduce on a remote machine
  - Log modification of node inputs and print stack trace to figure out location
  - Use `MacroAssembler::print_state64` or `stop` to print from compiled code
- **Don't hesitate to seek help if you get stuck**
- **Don't give up!**
  - Usually it takes just another hour - or week :)

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Documenting your progress

- **Set JIRA issues you are working on to “In Progress”**
  - Regularly give a status update in the comments
  - Update affects versions, labels, ...
- **Make sure the RFR/PR contains all required information**
  - You might have spent lots of time on the topic, reviewers didn’t
  - Also helps if you need to look at it again for a regression 6 months later ;)

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Questions?

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Live Debugging Session

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Bug 1: SIGFPE in C2 compiled code

```
# A fatal error has been detected by the Java Runtime Environment:
#
#  SIGFPE (0x8) at pc=0x00007f14c8aed2c5, pid=491680, tid=491681

[...]

---------------  T H R E A D  ---------------

Current thread (0x00007f14d0030dc0):  JavaThread "main" [_thread_in_Java, id=491681, stack(0x00007f14d7133000,0x00007f14d7234000)]

Stack: [0x00007f14d7133000,0x00007f14d7234000],  sp=0x00007f14d7232810,  free space=1022k
Native frames: (J=compiled Java code, A=aot compiled Java code, j=interpreted, Vv=VM code, C=native code)
J 293 c2 Test.test(II)I (4 bytes) @ 0x00007f14c8aed2c5 [0x00007f14c8aed2a0+0x0000000000000025]
j  Test.main([Ljava/lang/String;)V+12
v  ~StubRoutines::call_stub
V  [libjvm.so+0x9e13bb]  JavaCalls::call_helper(JavaValue*, methodHandle const&, JavaCallArguments*, Thread*)+0x5fd
V  [libjvm.so+0xec37c0]  os::os_exception_wrapper(void (*)(JavaValue*, methodHandle const&, JavaCallArguments*, Thread*), 
JavaValue*, methodHandle const&, JavaCallArguments*, Thread*)+0x36
V  [libjvm.so+0x9e0dba]  JavaCalls::call(JavaValue*, methodHandle const&, JavaCallArguments*, Thread*)+0x8e
V  [libjvm.so+0xa88a0e]  jni_invoke_static(JNIEnv_*, JavaValue*, _jobject*, JNICallType, _jmethodID*, JNI_ArgumentPusher*, 
Thread*)+0x188
V  [libjvm.so+0xa9e656]  jni_CallStaticVoidMethod+0x333
C  [libjli.so+0x4a2f]  JavaMain+0xbf7
C  [libjli.so+0xaca5]  ThreadJavaMain+0x27
```

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Bug 2: C2 compilation fails with assert

https://bugs.openjdk.java.net/browse/JDK-8176441

```
# A fatal error has been detected by the Java Runtime Environment:
#
#  Internal Error (/oracle/jdk/open/src/hotspot/share/opto/phaseX.cpp:1099), pid=505221, tid=505233
#  assert(false) failed: modified node was not processed by IGVN.transform_old()

[...]

Current CompileTask:
C2:   1226   30    b  4       custom.A8::unwrappedGeneratedCode (2223 bytes)

Stack: [0x00007fd2e6196000,0x00007fd2e6297000],  sp=0x00007fd2e62912a0,  free space=1004k
Native frames: (J=compiled Java code, A=aot compiled Java code, j=interpreted, Vv=VM code, C=native code)
V  [libjvm.so+0xf1629e]  PhaseIterGVN::verify_PhaseIterGVN()+0xa4
V  [libjvm.so+0xf16694]  PhaseIterGVN::optimize()+0x1a4
V  [libjvm.so+0xd22d8d]  PhaseIdealLoop::build_and_optimize(LoopOptsMode)+0x1879
V  [libjvm.so+0x6e5cb9]  PhaseIdealLoop::PhaseIdealLoop(PhaseIterGVN&, LoopOptsMode)+0xb3
V  [libjvm.so+0x6e5db4]  PhaseIdealLoop::optimize(PhaseIterGVN&, LoopOptsMode)+0x46
V  [libjvm.so+0x6da541]  Compile::Optimize()+0xa07
V  [libjvm.so+0x6d39ea]  Compile::Compile(ciEnv*, ciMethod*, int, bool, bool, bool, bool, DirectiveSet*)+0x1090
V  [libjvm.so+0x5d02ff]  C2Compiler::compile_method(ciEnv*, ciMethod*, int, bool, DirectiveSet*)+0x15b
V  [libjvm.so+0x6eea36]  CompileBroker::invoke_compiler_on_method(CompileTask*)+0x88e
V  [libjvm.so+0x6ed6c3]  CompileBroker::compiler_thread_loop()+0x3df
V  [libjvm.so+0x10b5239]  compiler_thread_entry(JavaThread*, Thread*)+0x69
V  [libjvm.so+0x10b0414]  JavaThread::thread_main_inner()+0x14c
V  [libjvm.so+0x10b02c0]  JavaThread::run()+0x124
V  [libjvm.so+0x10ac3d2]  Thread::call_run()+0x180
V  [libjvm.so+0xeb8290]  thread_native_entry(Thread*)+0x1e4
```

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Bug 3: Test fails with unexpected result

```
public static int[] test() {
  int[] result = new int[100];
  for (int i = 0; i < result.length; ++i) {
    result[i] = 42;
  }
  return result;
}
```

```
Exception in thread "main" java.lang.RuntimeException: Test failed: result[51] = 7
      at Test.main(Test.java:17)
```

ORACLE

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

ORACLE®