Debugging the RVM

This page contains some debugging hints for Jikes RVM. It is assumed that you are familiar with debugging techniques. If you aren't, it is advisable to read a book about the subject.

Debugging Helpers provided by Jikes RVM

Assertions

All debugging should be done with assertion-enabled builds if possible. You can also try using ExtremeAssertion builds.

Options

The Jikes RVM and MMTk provide several options to print out debugging information.

If you're debugging a problem in the optimizing compiler, you can also print out the IR.

You can also use the options to change the behaviour in various ways (e.g. switch off certain optimizations) if you have a suspicion about the causes of the problem.

Debugger Thread

Jikes has an interactive debugger that you can invoke by sending SIGQUIT to Jikes while it's running:

pkill -SIGQUIT JikesRVM

In previous versions of Jikes, that stopped all threads and provided an interactive prompt, but currently it just dumps the state of the VM and continues immediately (that's a known issue: RVM-570).

Debug fields in classes

Several classes in the code base provide static boolean fields like DEBUG or VERBOSE which can be set to get more debugging information.

Shutdown hooks

You can write custom shutdown hooks to dump gathered information when the VM terminates. Note that shutdown hooks won't be run if the VM is terminated via a signal (see RVM-555)

Do not use the ExitMonitor from the Callbacks class because it's less reliable.

Tests

The test coverage is poor at the moment. Nevertheless, if you're very lucky, one of the smaller test cases will fail. See Testing the RVM for details on how to run the tests and define your own.

Tools

There are different tools for debugging Jikes RVM:

GDB

There is a limited amount of C code used to start Jikes RVM. The rvm script will start Jikes RVM using GDB (the GNU debugger) if the first argument is `-gdb`. Break points can be set in the C code, variables, registers can be expected in the C code.

```
rvm -gdb <RVM args> <name of Java application> <application args>
```

The dynamically created Java code doesn't provide GDB with the necessary symbol information for debugging. As some of the Java code is created in the boot image, it is possible to find the location of some Java methods and to break upon them. To build with debug symbols, you'll need to set the appropriate property as described in Building the RVM.

Details of how to manually walk the stack in GDB can be found here

rdb

rdb is a debugger that was developed specifically for Jikes RVM. It allows you to inspect the bootimage. If you're running Mac OS, you can also use it to debug a running Jikes RVM.

Other Tools

Other tools, such as valgrind, are occasionally useful in debugging or understanding the behaviour of JikesRVM. The rvm script facilitates using these tools with the '-wrap' argument.

```
rvm -wrap "<wrapper-script-and-args>" <rest of command line>
```

For example, cachegrind can be invoked by

```
rvm -wrap "/path/to/valgrind --tool=cachegrind" <java-command-line>
```

The command and arguments immediately after the -wrap argument will be inserted into the script on the command line that invokes the boot image runner. One useful variant is

```
rvm -wrap echo <rest of command line>
```

Specific Debugging Hints

Optimizing Compiler Problems

To debug problems in the optimizing compiler, use a configuration whose bootimage is compiled with the baseline compiler and which contains the AOS (prototype-opt, BaseAdaptive*). Faster configurations (such as development) have the drawback of a longer bootimage compilation time which won't be amortized unless the problem occurs late.

It is advisable to use -X:vm:errorsFatal=true when debugging optimizing compiler problems. This will prevent the optimizing compiler from reverting to the baseline compiler for certain kinds of errors.

It is strongly recommended to run with advice file generation (see Experimental Guidelines). The produced advice files can then be used to reproduce the bug. If this step is successful, the advice files should be minimized to determine the set of methods that cause the failures. This can be done automatically (e.g. via delta debugging) or by hand.

You can also switch on paranoid IR verification in IR.java. Note that this is not well tested at the moment because we don't run any regression tests with it. Use a BaseAdaptive* configuration if you switch this on (bootimage builds with the optimizing compiler and paranoid IR fail at the time of this writing).

Deadlocks

To debug a deadlock, run the VM under a time limit and send SIGQUIT (to force a thread dump) a few seconds before killing the VM. On Linux, you can use the timelimit program (should be available in the repositories for Debian-based distributions).

Excluding Garbage Collection problems

The garbage collectors that are included with the Jikes RVM are generally stable. Therefore, if you see a problem that does not occur during the collection itself, it is likely not a garbage collection problem. You can exclude problems related to garbage collection by building with other

collectors. For example, you can choose a collector that doesn't move objects (e.g. MarkSweep) or a collector that doesn't require write barriers (e.g. Immix instead of GenImmix).