FAQ

Frequently Asked Questions

Questions

General

MMTk

Answers

General

What is the Jikes Research Virtual Machine?

Jikes RVM (Research Virtual Machine) is designed to execute Java™ programs that are typically used in research on fundamental virtual machine design issues. It provides the research communities with a flexible testbed to prototype new virtual machine technologies and experiment with different design alternatives. It runs on the Linux®/IA-32, AIX™/PowerPC™, OSX/PowerPC, and Linux/PowerPC platforms, and exhibits industry-strength performance for many benchmark programs on these platforms. Jikes RVM includes the latest VM technologies for dynamic compilation, adaptive optimization, garbage collection, thread scheduling, and synchronization.

A distinguishing characteristic of Jikes RVM is that it is implemented in the Java programming language, unlike other virtual machines that are implemented in "native code" (typically, C or C++). A Java implementation provides ease of portability, and a uniform memory space for virtual machine objects and application objects. Though there have been a few prior examples of virtual machines implemented in the Java programming language, all those cases dependent on the presence of a second underlying Java virtual machine and incurred large (100x-1000x) performance slowdowns as a result. Jikes RVM is unique in that it is the first self-bootstrapped virtual machine written entirely in the Java programming language, i.e., its Java code runs on itself, without requiring a second virtual machine.

What is the relationship to the Jalapeño project?

Jikes RVM is an enhanced open source version of the code developed under the Jalapeño research project from December 1997 to October 2001. A historical overview of the project is now available.

Is Jikes RVM for all programmers?

Jikes RVM was initially designed for researchers. Jikes RVM can run many, but not all Java programs. The Classpath libraries, which Jikes RVM uses, currently does not provide a complete Java implementation; Swing and AWT coverage is particularly incomplete. Jikes RVM also currently does not support other features such as bytecode verification.

Despite these limitations Jikes RVM can run many preexisting substantial Java applications without modification. Eclipse is an open source multi-million line Java program, which has in the past runs without modification on Jikes RVM. It differs from many Java GUI applications in that it does not use AWT or Swing.

Why is Jikes RVM of interest to the research community?

In 2001, early versions of Jikes RVM were made available to 16 universities under a pre-release license. These universities found that Jikes RVM provides the best available vehicle for research on the frontiers of virtual machine technology. Many of these universities discarded their own efforts in building a VM for research purposes, in favor of using Jikes RVM. Furthermore, since the open source release of Jikes RVM in Oct 2001, dozens of publications have appeared at top conferences written by users of the system. Furthermore, numerous courses have been taught using Jikes RVM. Over 80 universities have used Jikes RVM.

Properties of Jikes RVM that these researchers have found useful are:

- it is written in the Java programming language
- it is designed for research and experimentation, has a clear structure, is extensible and modular, including:
    - a flexible adaptive optimization architecture
    - a full-fledged optimizing compiler infrastructure
    - a modular, highly configurable memory management toolkit (MMTk)
- it is a state-of-the-art research implementation providing:
    - credible research results
    - competitive performance with top commercial systems
- it has an established user community, including
    - active mailing lists
    - bug-tracking software
    - Mercurial repository
    - a user's guide
    - a browsable API
    - and nightly regression tests

Why did IBM make Jikes RVM open source?

IBM is committed to open source and open standards. We would like to work with and contribute to open source communities. By open-sourcing Jikes RVM, we hope to accelerate progress in developing next-generation virtual machine technologies.

How can I contribute? Where should I start?

Feedback and contributions from the research community are the indispensable, invaluable resource that will ensure the project's continued success. No improvement is too small or too simple! The more volunteers can contribute, the better the system will serve the research community, resulting in faster and more solid research results. Details on how to get started contributing, including some low-hanging fruit, are provided here.

Why did you choose Linux?

The availability of source code for Linux gives researchers the option of experimenting with a larger testbed that includes both the operating system and the virtual machine. Also, Linux is immensely popular among the research community in universities.

Why does Jikes RVM run Java bytecodes?

Java bytecodes are perhaps the best-known virtual machine instruction set. The widespread availability of example programs and benchmark programs in Java bytecodes makes any research results obtained for Jikes RVM much more relevant to a broader audience, compared to a virtual machine for a lesser-known instruction set.

Who maintains Jikes RVM?

The system is maintained by the Jikes RVM team, a group that initially was comprised of a subset of the Jikes RVM developers at IBM Research. In August, 2002, this group was extended to include non-IBMers. We would like to evolve the Jikes RVM team to include more non-IBMers. See Project Organization for further details.

Why are different licenses used for Jikes RVM and the libraries?

Jikes RVM is released under the EPL (Eclipse Public License), which has been approved by the OSI (Open Source Initiative) as a fully certified open source license. Jikes RVM uses the GNU Classpath libraries, a set of open source libraries for the Java programming language with their own open source license. See License for more details.

Does IBM Research plan to continue to use Jikes RVM?

IBM Research plans to continue using Jikes RVM as a testbed to develop new virtual machine technologies internally in IBM, and also through collaborations with universities. Some of the technology areas that we might pursue include: dynamic code optimization, virtual machine technologies for web services and grid computing, extensions for real-time processing, smart debugging, integration into the Eclipse framework, and technologies for improving the speed of XML processing.

How does this project relate to the Jikes compiler open source project?

Jikes is an open-source compiler that translates Java source files as defined in "The Java Language Specification" into the bytecode instruction set and binary format defined in "The Java Virtual Machine Specification". Jikes RVM is a VM that was designed and developed for use by the research community for investigating issues in virtual machine design and implementation. Jikes RVM takes over where the Jikes Java source to bytecode compiler leaves off; Jikes RVM compiles byte codes (.class files) into native code and provides an execution environment for that native code to run in.

Can I use Jikes RVM with a hardware simulator?

Some researchers wish to use Jikes RVM on top of a hardware simulator. There are at least three classes of simulator to consider.

- Binary re-writers such as Valgrind and PIN
- Functional simulators such as provided by SimICS and others.
- Cycle accurate simulators such as SimpleScalar and PTLSim

Binary re-writers *should* be able to run Jikes RVM out of the box. However, we know of at least three concerns:

1. Jikes RVM uses just-in-time compilation and also re-compilation. The simulator must be able to cope with self modifying code and dynamic code generation.
2. Jikes RVM makes use of the INT 0x40-43 instruction to generate synchronous traps for array bounds failures. This is not a commonly used instruction. Valgrind did not originally support the instruction, but subsequently did. Unfortunately support for the instruction was inadvertently dropped in a later release of Valgrind, but may have now been restored. To test whether this is a problem for your simulator, write a trivial program that generates and catches an array out of bounds exception, and check that your simulator can execute it correctly.
3. Jikes RVM statically nails down virtual address ranges (its boot image and heap are pre-determined to be in specific locations in the heap, see config files in build/hosts/*). These address ranges may clash with libraries required by the host runtime, with hard-to-diagnose consequences (since libraries may get clobbered!). This is true of both regular OS's and simulators.

Jikes RVM has been successfully used with the SimICS functional simulator.

For some research, cycle accurate simulation is necessary (as opposed to functional simulation). People at UMass and U Texas created DSS, which is an extension of SimpleScalar to support the needs of Jikes RVM. However, it does not support versions of Jikes RVM more recent than 2.4.4 because of a variety of new dependencies in Jikes RVM and GNU Classpath that emerged after 2.4.4. People at U Texas and ANU investigated and determined that the effort involved in bringing DSS up to speed was too great to warrant the effort involved.

A number of people at ANU, U Texas and Kent are using PTLSim. However PTLSim does not currently support Jikes RVM due to a bug in its IA32 signal handling code. PTLSim is however stable in IA32-64, so once a port of Jikes RVM to IA32-64 is done, it should be possible to simulate Jikes RVM over PTLSim.

MMTk

What is a Space?

A Space manages two distinct resources;

- a budget of pages in use at a particular time. This is ultimately limited by the maximum heap size specified on the command line.
- a virtual memory address range. This is determined by the space instance's constructor and thus specified at build time.

The management of the resources occurs according to a specific policy.

What is a "monotone" Space?

A "monotone" space (see org.mmtk.utility.heap.MonotonePageResource) will walk through their allotted address range consuming pages until their page budget is exhausted. They are typically then **completely** emptied ("evacuated" in GC terminology) and reset to do the same thing all over again. This is how each of the two semispaces in a semi space collector works, and how the nursery in a generational collector works.

What is a "freelist" Space?

A "freelist" space (see org.mmtk.utility.heap.FreeListPageResource) will grab free pages from within the address space and **incrementally** return those pages to the space (not all at once, which is what the monotone space will do).

What is precopying?

Precopying is the mechanism used by MMTk to ensure that any GC critical objects are not moved **during** the main part of the GC (such as the stacks etc which are used during GC, but are also subject to GC because of our Java-in-Java implementation). What we do instead is move those objects before the main part of the GC (this is called precopying). If precopying is not done correctly you may see assertion errors in ScanThread.assertImmovableInCurrentCollection(). The method checks several objects are not subject to movement. If one of these objects has not been precopied it will be subject to movement and will thus fail.

Why are some classes named with Local suffix?

There is a strong concept of "local" (unsynchronized) and "global" (synchronized) operations. In general, each allocator will acquire sufficient global resources to run unsynchronized for a while. When that local resource is expended, it will call to the (global) space into which it is allocating, and request some number of new pages. If that request is unsuccessful, a GC will be triggered. The codebase attempts to maximize the use of locals and minimize use of globals, since synchronization is expensive.