A Java Virtual Machine for RISC-V

Porting the Jikes Research VM

Martin Maas, UC Berkeley

(maas@eecs.berkeley.edu)

5th RISC-V Workshop, Nov 30, 2016

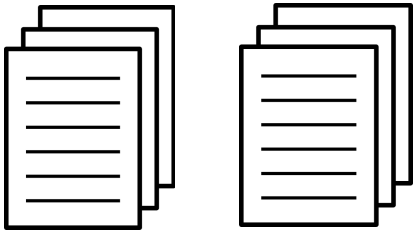
Mountain View, CA

Why Do We Need a JVM Port?

Run Java Applications

High-performance production JVM

Academic Research



Easy-to-modify research JVM

Which JVM should we port?

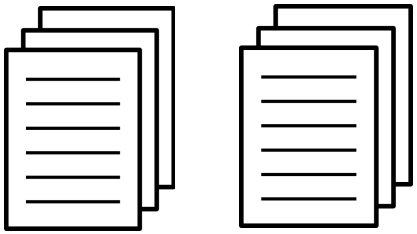
Both!

Why Do We Need a JVM Port?

Run Java Applications

High-performance production JVM

Academic Research



Easy-to-modify research JVM

OpenJDK/Hotspot JVM

Implementer: Michael Knyszek
Status: Runs with zero backend, C1 port in progress

Jikes Research VM

Implementer: Martin Maas
Status: Runs with baseline (non-optimizing) JIT compiler

Why Do We Need a JVM Port?

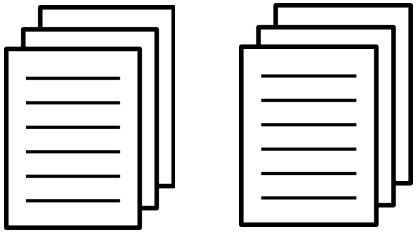
Run Java Applications

High-performance production JVM

OpenJDK/Hotspot JVM

Implementer: Michael Knyszek
Status: Runs with zero backend, C1 port in progress

Academic Research



Easy-to-modify research JVM

Jikes Research VM

Implementer: Martin Maas
Status: Runs with baseline (non-optimizing) JIT compiler

Talk Agenda

1. **The Jikes Research VM**
   How does it work, what does it do?
2. **Porting a JVM to RISC-V**
   Lessons learned from porting a JVM
3. **Status & Next Steps**
   Screenshots and current status of the port

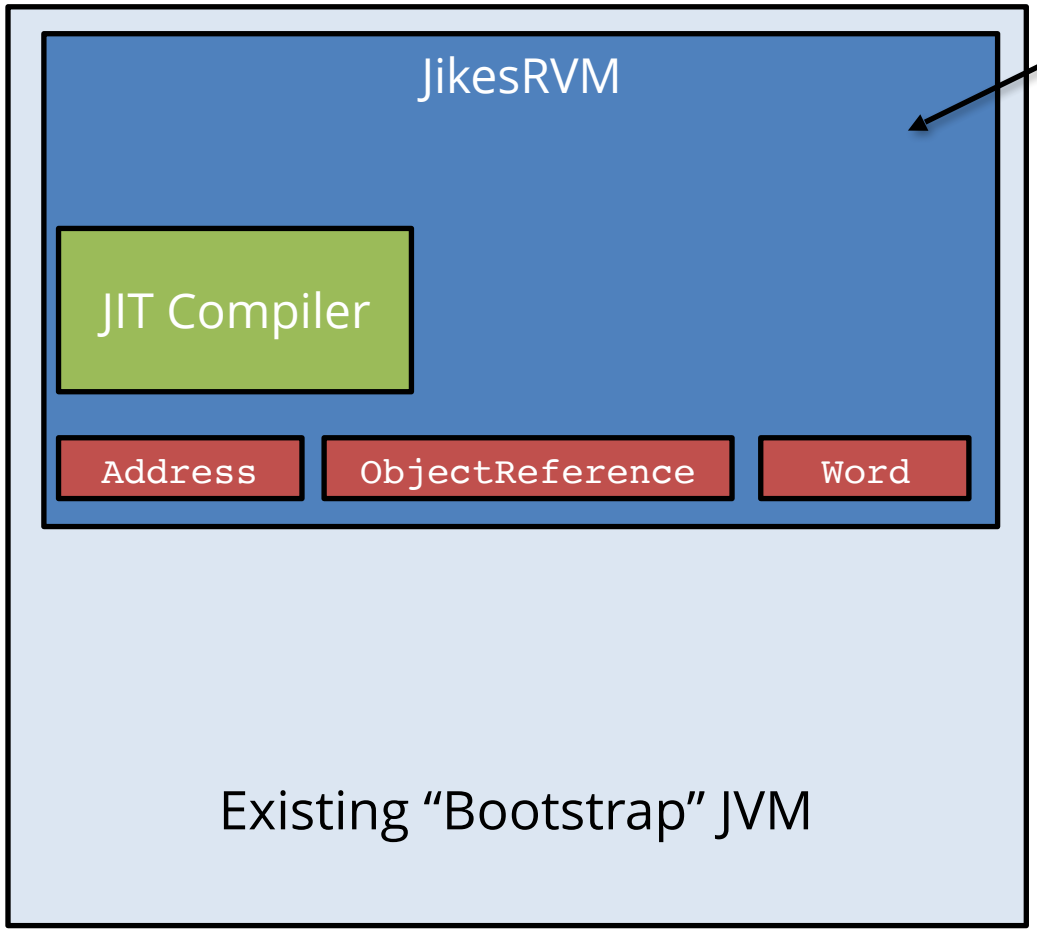
PART I
The Jikes Research VM

JikesRVM History

- Originally written at IBM in 1997, maintained by research groups (ANU, UT Austin, etc.)
- Used in **over 200 research papers**
  - 2012: ACM SIGPLAN Software Award
- **JikesRVM is a JVM written in Java** (this is also called a “metacircular” runtime system)

Compiling the Jikes Research VM

A JVM written in Java, how does this work?



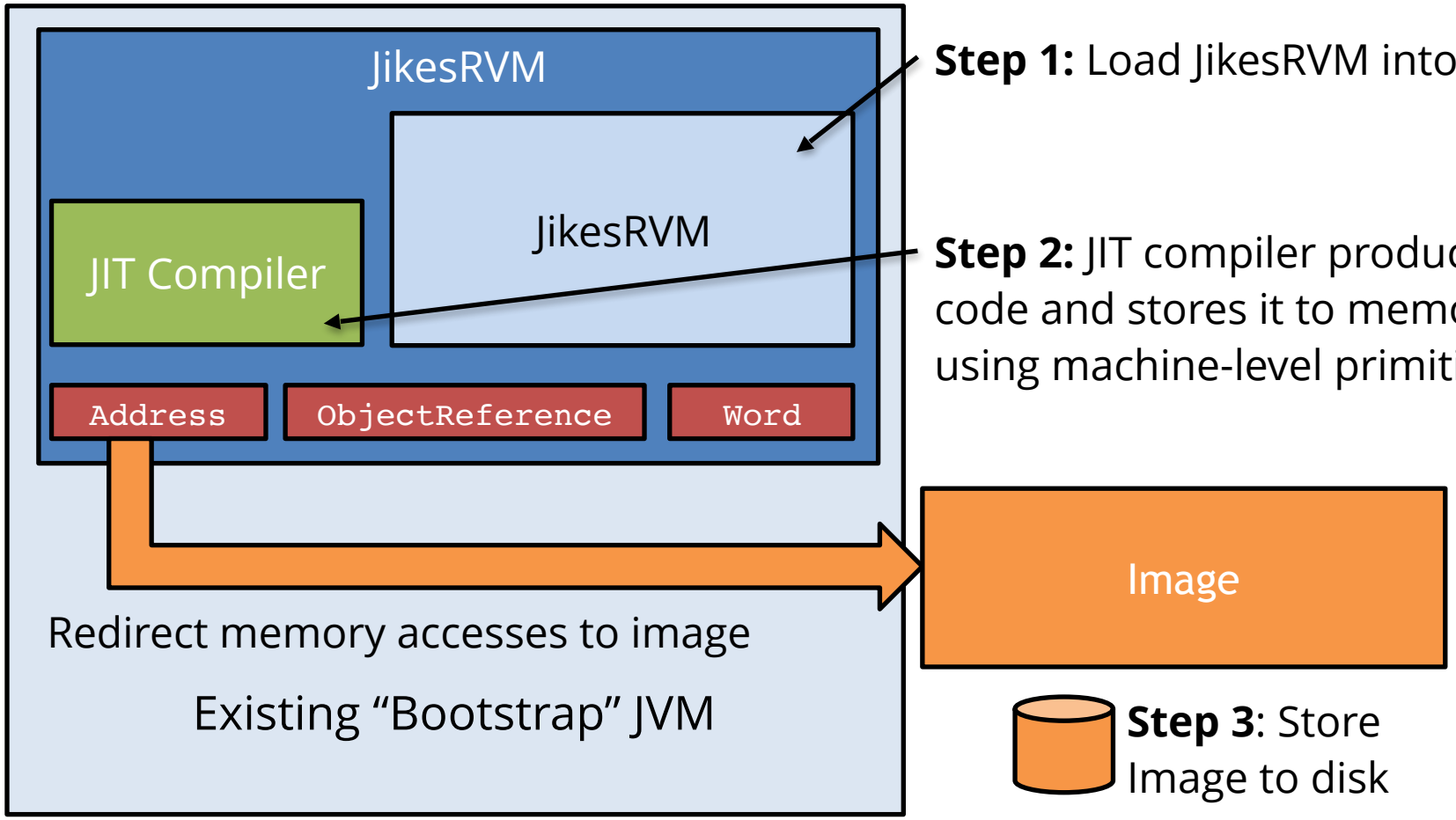
Just a **Java application**

Special classes encapsulate **low-level primitives:**

- Regular classes for the bootstrap JVM
- Handled specially by JikesRVM JIT compiler

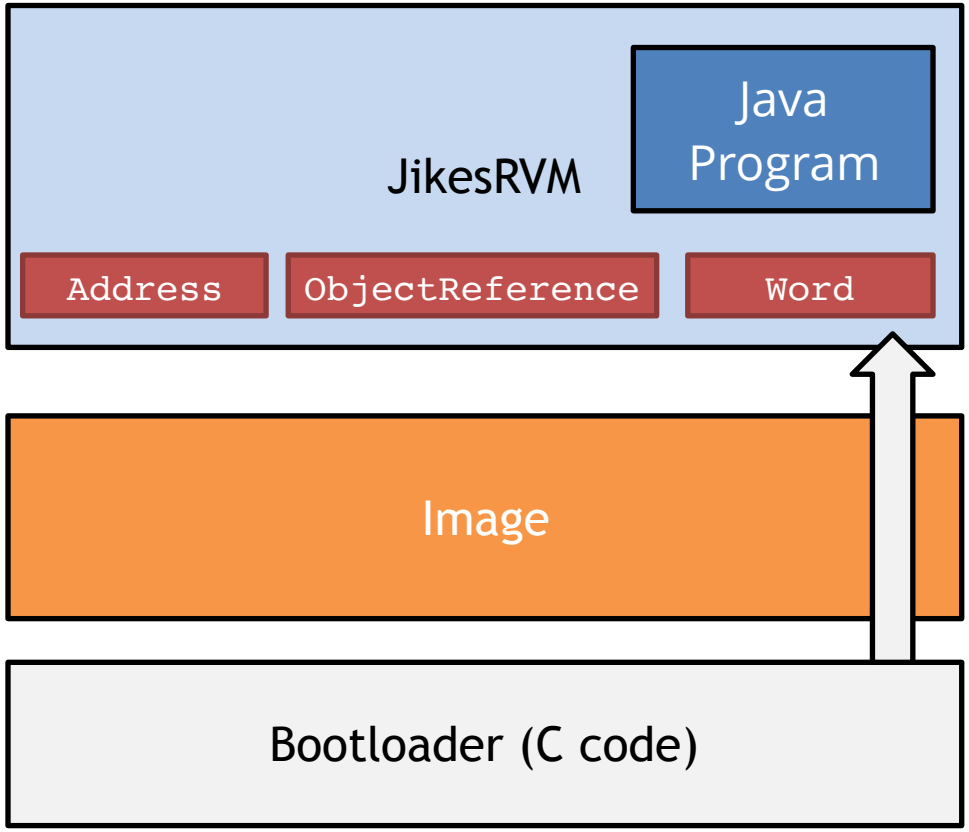
Compiling the Jikes Research VM

A JVM written in Java, how does this work?



Running the Jikes Research VM

Running the JVM (without the bootstrap JVM)



In JIT-generated code, primitives map to actual operations

Resume at “boot” function.

Loads boot image into memory implements native calls, fault handlers, etc.

WHY JIKESRVM?

JikesRVM allows writing systems code in a type-safe high-level language, and is easy to modify.

This makes it perfect for research!

PART II
Porting a JVM to RISC-V

Lesson #1: Dependencies

JVMs have a large number of dependencies. Use the **riscv-poky Linux distribution generator** to build a cross-compile SDK and a Linux image.

```
    http://yoctoproject.org/documentation

For more information about OpenEmbedded see their website:
    http://www.openembedded.org/

You had no conf/bblayers.conf file. The configuration file has been created for
you with some default values. To add additional metadata layers into your
configuration please add entries to this file.

The Yocto Project has extensive documentation about OE including a reference manual
which can be found at:
    http://yoctoproject.org/documentation

For more information about OpenEmbedded see their website:
    http://www.openembedded.org/



### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

maas@a6:/scratch/maas/poky/demo/riscv-poky/build$ bitbake
Parsing recipes:  29% |##############
```

Clone riscv-poky and run:
`bitbake meta-toolchain`

To add missing dependencies, edit:
meta-riscv/recipes-core/images/core-image-riscv.bb

Install the resulting SDK image

```
martin@virtualbox:~/src/riscv/toolchain$ ./poky-riscv-glibc-x86_64-meta-toolchain-riscv64-toolchain-2.0+snapshot.sh
Poky (Yocto Project Reference Distro) SDK installer version 2.0+snapshot
==========================================================================
Enter target directory for SDK (default: /opt/poky-riscv/2.0+snapshot): ~/src/riscv/toolchain/sdk
You are about to install the SDK to "/home/martin/src/riscv/toolchain/sdk". Proceed[Y/n]? Y
Extracting SDK.............done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
 $ . /home/martin/src/riscv/toolchain/sdk/environment-setup-riscv64-poky-linux
martin@virtualbox:~/src/riscv/toolchain$ . /home/martin/src/riscv/toolchain/sdk/environment-setup-riscv64-poky-linux
```

Lesson #2: Build Process

For fast turnaround, write a single script that:

(1) Builds the JVM and copies the result into the Linux image

```
[zip] java/lang/VMClassLoader.class already added, skipping
[zip] java/lang/VMDouble.class already added, skipping
[zip] java/lang/VMFloat.class already added, skipping
[zip] java/lang/VMMath.class already added, skipping
[zip] java/lang/VMRuntime.class already added, skipping
[zip] java/lang/VMString.class already added, skipping
[zip] java/lang/VMSystem.class already added, skipping
[zip] java/lang/VMThread.class already added, skipping
[zip] java/lang/VMThrowable.class already added, skipping
[zip] java/lang/management/VMManagementFactory.class already added, skipping
[zip] java/lang/ref/PhantomReference.class already added, skipping
[zip] java/lang/ref/Reference.class already added, skipping
[zip] java/lang/ref/ReferenceQueue.class already added, skipping
[zip] java/lang/ref/SoftReference.class already added, skipping
[zip] java/lang/ref/WeakReference.class already added, skipping
[zip] java/lang/reflect/VMArray.class already added, skipping
[zip] java/lang/reflect/VMConstructor.class already added, skipping
[zip] java/lang/reflect/VMField.class already added, skipping
[zip] java/lang/reflect/VMMethod.class already added, skipping
[zip] java/nio/VMDirectByteBuffer.class already added, skipping
[zip] java/security/VMAccessController.class already added, skipping
[zip] java/util/VMTimeZone.class already added, skipping
[zip] java/util/concurrent/atomic/AtomicLong.class already added, skipping
[zip] sun/misc/Unsafe.class already added, skipping
[jar] Updating jar: /home/martin/src/riscv/git/JikesRVM/target/BaseBaseNoGC_riscv64-linux/jksvm.jar

package-harmony:

package:

prepare-asm:

gen-asm:
      [zip] Updating zip: /home/martin/src/riscv/git/JikesRVM/target/BaseBaseNoGC_riscv64-linux/rvmrt.jar

gen-jni-headers:

gen-interface:

check-git-present:

extract-git-version:

get-git-version:

gen-runbootimage:
```

```
JIKES_PATH=/[…]/BaseBaseNoGC_riscv64-linux
sudo mount -t ext2 -o loop,rw jikes.rootfs.ext2 mount
pushd mount
sudo rm -r home/root/BaseBaseMarkSweep_riscv64-linux
rm $JIKES_PATH/BootImageWriterOutput.txt
sudo cp -R $JIKES_PATH home/root/BaseBaseMarkSweep_riscv64-linux
popd
sudo umount mount
```

(2) Boots the JVM in QEMU with an inittab that runs the JVM, copies the result into a file and shuts down the machine

/etc/inittab

```
[…]
l6:6:wait:/etc/init.d/rc 6
z6:6:respawn:/sbin/sulogin
H0:12345:respawn:/bin/bash -c "/home/root/test.sh”
```

/home/root/test.sh

```
export LD_LIBRARY_PATH=/home/root
echo "====================== […] =======================”
echo "-- JIKES RVM TEST ENVIRONMENT --”
echo "====================== […] =======================”
cd /home/root/BaseBaseMarkSweep_riscv64-linux/
./rvm -X:verbose -X:verboseBoot=10 HelloWorld 2>&1 | ../spike-dasm
echo "====================== […] =======================”
halt
bash
```

```
hwclock: can't open '/dev/misc/rtc': No such file or directory
Tue May 31 14:23:28 UTC 2016
hwclock: can't open '/dev/misc/rtc': No such file or directory
INIT: Entering runlevel: 1
Unmounting remote filesystems...
=====================================================================
-- JIKES RVM TEST ENVIRONMENT --
=====================================================================
/home/root/BaseBaseMarkSweep_riscv64-linux/tests /home/root/BaseBaseMarkSweep_riscv64-linux
Running test.org.jikesrvm.basic.java.lang.TestMath...
[    6.340000] random: nonblocking pool is initialized
Math.floor(1.6) Expected: 1.0 Actual: 1.0
Math.floor(1.5) Expected: 1.0 Actual: 1.0
Math.floor(1.4) Expected: 1.0 Actual: 1.0
Math.floor(1.0) Expected: 1.0 Actual: 1.0
Math.floor(-2.0) Expected: -2.0 Actual: -2.0
Math.floor(-1.6) Expected: -2.0 Actual: -2.0
Math.floor(-1.5) Expected: -2.0 Actual: -2.0
Math.floor(-1.4) Expected: -2.0 Actual: -2.0
Math.floor(NaN) Expected: NaN Actual: NaN
Math.floor(-Infinity) Expected: -Infinity Actual: -Infinity
Math.floor(Infinity) Expected: Infinity Actual: Infinity
Math.floor(0.0) Expected: 0.0 Actual: 0.0
Math.floor(0.0) Expected: 0.0 Actual: 0.0
Math.ceil(1.6) Expected: 2.0 Actual: 2.0
Math.ceil(1.5) Expected: 2.0 Actual: 2.0
Math.ceil(1.4) Expected: 2.0 Actual: 2.0
Math.ceil(1.0) Expected: 0.0 Actual: 1.0
Math.ceil(-2.0) Expected: -2.0 Actual: -2.0
Math.ceil(-1.6) Expected: -1.0 Actual: -1.0
Math.ceil(-1.5) Expected: -1.0 Actual: -1.0
Math.ceil(-1.4) Expected: -1.0 Actual: -1.0
Math.ceil(NaN) Expected: NaN Actual: NaN
Math.ceil(-Infinity) Expected: -Infinity Actual: -Infinity
Math.ceil(Infinity) Expected: Infinity Actual: Infinity
Math.ceil(0.0) Expected: 0.0 Actual: 0.0
Math.ceil(0.0) Expected: 0.0 Actual: 0.0
Math.sqrt(4.0) Expected: 2.0 Actual: 2.0
Math.sqrt(Infinity) Expected: Infinity Actual: Infinity
Math.sqrt(NaN) Expected: NaN Actual: NaN
Math.sqrt(-Infinity) Expected: NaN Actual: NaN
Math.sqrt(-1.0) Expected: NaN Actual: NaN
Math.sqrt(-0.0) Expected: -0.0 Actual: -0.0
Math.sqrt(0.0) Expected: 0.0 Actual: 0.0
/home/root/BaseBaseMarkSweep_riscv64-linux
INIT: ===================================================================
Switching to runlevel: 0
INIT: Sending processes the TERM signal
hwclock: can't open '/dev/misc/rtc': No such file or directory
Stopping syslogd/klogd: no syslogd found; none killed
Deconfiguring network interfaces... ifdown: interface lo not configured
ifdown: interface eth0 not configured
done.
Sending all processes the TERM signal...
Deactivating swap...
Unmounting local filesystems...
[   14.510000] reboot: Power down
TEST PASSED
```

Lesson #3: Generate the Assembler

Don't write the assembler by hand, use **riscv-opcodes** and generate it!

```python
def make_jikesrvm():
  for name in namelist:
    args = arguments[name]

    funargs = []
    substargs = []

    register_order = ['rd', 'rs1', 'rs2', 'rs3']

    # loads/stores have different order of registers
    if name in ['lb', 'lh', 'lw', 'ld', 'flw', 'fld', 'sb', 'sh', 'sw', 'sd', 'fsw', 'fsd']:
      register_order = ['rd', 'rs2', 'rs1']

    fcvt_float2int = []
    fcvt_int2float = []
    fcmp = []

    for f in ['d', 's']:
      for i in ['l', 'w', 'lu', 'wu']:
        fcvt_float2int.append('fcvt.%s.%s' % (i,f))
        fcvt_int2float.append('fcvt.%s.%s' % (f,i))

      for instr in ['fle', 'flt', 'feq']:
        fcmp.append('%s.%s' % (instr, f))

    for r in register_order:
      if r in args:
        if name.startswith('f'):
                if name in ['flw', 'fld', 'fsw', 'fsd'] and r == 'rs1':
                        funargs.append('GPR ' + r)
                elif name in fcvt_float2int and r == 'rd':
                        funargs.append('GPR ' + r)
                elif name in fcvt_int2float and r == 'rs1':
                        funargs.append('GPR ' + r)
                elif name in fcmp and r == 'rd':
                        funargs.append('GPR ' + r)
                elif name in ['frflags', 'fsflags', 'fsflagsi']:
                        funargs.append('GPR ' + r)
                else:
                        funargs.append('FPR ' + r)
        else:
                funargs.append('GPR ' + r)

        substargs.append('%s.value() << %d' % (r, arglut[r][1]))
        args.remove(r)
```

```java
public final void emitLH(GPR rd, GPR rs1, int imm12) {
  int mi = 0x1003 | rd.value() << 7 | rs1.value() << 15 | make_imm12(imm12);
  mIP++;
  mc.addInstruction(mi);
}

public final void emitLW(GPR rd, GPR rs1, int imm12) {
  int mi = 0x2003 | rd.value() << 7 | rs1.value() << 15 | make_imm12(imm12);
  mIP++;
  mc.addInstruction(mi);
}

public final void emitLD(GPR rd, GPR rs1, int imm12) {
  int mi = 0x3003 | rd.value() << 7 | rs1.value() << 15 | make_imm12(imm12);
  mIP++;
  mc.addInstruction(mi);
}

public final void emitLBU(GPR rd, GPR rs1, int imm12) {
  int mi = 0x4003 | rd.value() << 7 | rs1.value() << 15 | make_imm12(imm12);
  mIP++;
  mc.addInstruction(mi);
}

public final void emitLHU(GPR rd, GPR rs1, int imm12) {
  int mi = 0x5003 | rd.value() << 7 | rs1.value() << 15 | make_imm12(imm12);
  mIP++;
  mc.addInstruction(mi);
}

public final void emitLWU(GPR rd, GPR rs1, int imm12) {
  int mi = 0x6003 | rd.value() << 7 | rs1.value() << 15 | make_imm12(imm12);
  mIP++;
  mc.addInstruction(mi);
}

public final void emitSB(GPR rs2, GPR rs1, int imm12) {
  int mi = 0x23 | rs2.value() << 20 | rs1.value() << 15 | make_imm12hi(imm12);
  mIP++;
  mc.addInstruction(mi);
}

public final void emitSH(GPR rs2, GPR rs1, int imm12) {
  int mi = 0x1023 | rs2.value() << 20 | rs1.value() << 15 | make_imm12hi(imm12);
  mIP++;
  mc.addInstruction(mi);
}
```

Lesson #4: Assertions everywhere!

Add assertions to check that constants fit into the immediate, loaded values are correct, etc.

```
public static void patchCode(ArchitectureSpecific.CodeArray code, int indexa, int indexb) {
  if (VM.VerifyAssertions) VM._assert(VM.NOT_REACHED);
}

public final void check_fits_signed(int val, int bits) {
  if (VM.VerifyAssertions) {
    val = val >> bits - 1;
    VM._assert(val == 0 || val == -1);
  }
}

public final void check_fits_unsigned(int bits, int val) {
  if (VM.VerifyAssertions) {
    VM._assert(val >= 0 && (val >> bits) == 0);
  }
}

public final int mask(int val, int lo, int hi) { return ((val >> lo) & ((1 << (hi-lo+1)) - 1)); }

public final int make_imm20(int val) { check_fits_unsigned(20, val); return (val << 12); }
public final int make_imm12(int val) { check_fits_signed(12, val); return (val << 20); }
public final int make_jimm20(int val) { check_fits_signed(20, val);
  return (mask(val, 20, 20) << 31) | (mask(val, 1, 10) << 21) |
         (mask(val, 12, 19) << 12) | (mask(val, 11, 11) << 20);
}
public final int make_bimm12hi(int val) { check_fits_signed(12, val);
  return (mask(val, 12, 12) << 31) | (mask(val, 5, 10) << 25) |
         (mask(val, 1, 4) << 8) | (mask(val, 11, 11) << 7);
}
public final int make_imm12hi(int val) { check_fits_signed(12, val);
  return (mask(val, 5, 11) << 25) | (mask(val, 0, 4) << 7);
}
public final int make_shamt(int val) { check_fits_signed(5, val);
  return (val << 20);
}
public final int make_shamtw(int val) { check_fits_signed(20, val);
  return (val << 20);
}
```

```
            r19=(nil)
            r20=0x8
            r21=0x10
            r22=0x7fc
            r23=0x32f7d308
            r24=0x8000
            r25=0x1
            r26=0x1a
            r27=0x200002e9c8
            r28=0x31a71220
            r29=(nil)
            r30=(nil)
            r31=(nil)
JikesRVM: Unexpected hardware trap signal 0x4
UNKNOWN TRAPCODE             fp=0x32933a80
             tr=0x32f7d308
             ip=0x32933b20
          instr=0x32f7d308
```

Before assertions can be caught, let them emit an instruction that causes a SEGFAULT and dump state in the fault handler

e.g., LW X0, 4(X0)

Lesson #5: Debug Traces

Generate traces of bytecodes and machine code.
Can do this with software breakpoints.

```
---- 0x3541ff7c (Opcode: getstatic, Stack: 72)
|-> 0x32fcecd0 [72] ->  0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88 0x532f7dda8 0x2ca0 0x32f7ede0
  0x3541ff8c        lui     t0, 0x1
  0x3541ff90        xori    t0, t0, 368
  0x3541ff94        add     t0, t0, gp
  0x3541ff98        ld      t0, 0(t0)
  0x3541ff9c        sd      t0, 64(sp)
---- 0x3541ffa0 (Opcode: iconst_0, Stack: 64)
|-> 0x32fcecd0 [64] ->  0x330046c8 0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88 0x532f7dda8 0x2ca0
  0x3541ffb0        li      t0, 0
  0x3541ffb4        sw      t0, 60(sp)
---- 0x3541ffb8 (Opcode: iconst_1, Stack: 56)
|-> 0x32fcecd0 [56] ->  0x0 0x330046c8 0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88 0x532f7dda8
  0x3541ffc8        li      t0, 1
  0x3541ffcc        sw      t0, 52(sp)
---- 0x3541ffd0 (Opcode: iastore, Stack: 48)
|-> 0x32fcecd0 [48] ->  0x100000000 0x0 0x330046c8 0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88
  0x3541ffe0        lw      t2, 52(sp)
  0x3541ffe4        lw      t1, 60(sp)
  0x3541ffe8        ld      t0, 64(sp)
  0x3541ffec        lw      t3, 4088(t0)
  0x3541fff0        bltu    t1, t3, pc + 8
  0x3541fff4        lb      zero, 1(zero)
  0x3541fff8        slli    t1, t1, 2
  0x3541fffc        add     t1, t1, t0
  0x35420000        sw      t2, 0(t1)
---- 0x35420004 (Opcode: getstatic, Stack: 72)
|-> 0x32fcecd0 [72] ->  0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88 0x532f7dda8 0x2ca0 0x32f7ede0
  0x35420014        lui     t0, 0x1
  0x35420018        xori    t0, t0, 368
  0x3542001c        add     t0, t0, gp
  0x35420020        ld      t0, 0(t0)
  0x35420024        sd      t0, 64(sp)
---- 0x35420028 (Opcode: iconst_1, Stack: 64)
|-> 0x32fcecd0 [64] ->  0x330046c8 0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88 0x532f7dda8 0x2ca0
  0x35420038        li      t0, 1
  0x3542003c        sw      t0, 60(sp)
---- 0x35420040 (Opcode: iconst_1, Stack: 56)
|-> 0x32fcecd0 [56] ->  0x100000000 0x330046c8 0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88 0x532f7dda8
  0x35420050        li      t0, 1
  0x35420054        sw      t0, 52(sp)
---- 0x35420058 (Opcode: iastore, Stack: 48)
|-> 0x32fcecd0 [48] ->  0x100000000 0x100000000 0x330046c8 0x0 0x0 0x32f7dda8 0x200002e9a8 0x32fced88 0x3541d7e8 0xb5b 0x0 0x32fced88
  0x35420068        lw      t2, 52(sp)
  0x3542006c        lw      t1, 60(sp)
  0x35420070        ld      t0, 64(sp)
  0x35420074        lw      t3, 4088(t0)
  0x35420078        bltu    t1, t3, pc + 8
  0x3542007c        lb      zero, 1(zero)
  0x35420080        slli    t1, t1, 2
  0x35420084        add     t1, t1, t0
  0x35420088        sw      t2, 0(t1)
```

At the start of every Bytecode, emit a text sequence:

```
LD X0, 1024(X0) # SEGFAULT
(Number of instructions)
(Opcode)
(Stack Offset)
```

Print out instructions as DASM(0x12345678) **and pipe the output through spike-dasm**

Lesson #6: Implement JIT Incrementally

- Switch off optional features: No GC, no biased locking, one thread, avoid floating point, etc.
- Put a failing assertion for everything you skip.

Find out what the first bytecode ever executed by the JVM is. Write all your initial tests there.

(For Hotspot, it's the static initializer of the java.lang.String class, for Jikes it's VM.boot())

```
@UnpreemptibleNoWarn("No point threading until threading is booted")
@Entrypoint
public static void boot() {
  // TESTING CODE: Arrays.
  /*int N = 10;
  int[] a = new int[N];
  a[0] = 1;
  a[1] = 1;
  for (int i = 2; i < N; i++) {
    a[i] = a[i-1] + a[i-2];
  }
  VM.sysWriteln(a[N-1]);*/

  // TESTING CODE: Output and static function calls.
  /*sysCall.sysConsoleWriteInteger(42, 0);
  sysCall.sysConsoleWriteChar('\n');
  sysCall.sysExit(0);*/
  //sysCall.sysNumProcessors();

  /*
  VM.runningVM = true;
  test();
  sysCall.sysExit(0);

  return;
  */
```

Lesson #7: Calling Convention

RVG is a good starting point (need to intermix C and JVM frames for native calls). Be careful with registers used by glibc (e.g., TP/x4, GP/x3).

```
GPR[] LOCAL_GPRS = { GPR.X9, GPR.X18, GPR.X19, GPR.X20, GPR.X21, GPR.X22, GPR.X23, GPR.X24, GPR.X25, GPR.X26, GPR.X27 };
int NUM_LOCAL_GPRS = LOCAL_GPRS.length;

FPR[] LOCAL_FPRS = { FPR.F8, FPR.F9, FPR.F18, FPR.F19, FPR.F20, FPR.F21, FPR.F22, FPR.F23, FPR.F24, FPR.F25, FPR.F26, FPR.F27 };
int NUM_LOCAL_FPRS = LOCAL_FPRS.length;

// Treat SP and FP special (even though nonvolatile in theory)
GPR[] NONVOLATILE_GPRS = { GPR.X9, GPR.X18, GPR.X19, GPR.X20, GPR.X21, GPR.X22, GPR.X23, GPR.X24, GPR.X25, GPR.X26, GPR.X27 };
FPR[] NONVOLATILE_FPRS = { FPR.F8, FPR.F9, FPR.F18, FPR.F19, FPR.F20, FPR.F21, FPR.F22, FPR.F23, FPR.F24, FPR.F25, FPR.F26, FPR.F27 };

int NUM_NONVOLATILE_GPRS = NONVOLATILE_GPRS.length;
int NUM_NONVOLATILE_FPRS = NONVOLATILE_FPRS.length;

GPR[] PARAMETER_GPRS = { GPR.X10, GPR.X11, GPR.X12, GPR.X13, GPR.X14, GPR.X15, GPR.X16, GPR.X17 };
int NUM_PARAMETER_GPRS = PARAMETER_GPRS.length;

FPR[] PARAMETER_FPRS = { FPR.F10, FPR.F11, FPR.F12, FPR.F13, FPR.F14, FPR.F15, FPR.F16, FPR.F17 };
int NUM_PARAMETER_FPRS = PARAMETER_FPRS.length;

GPR[] OS_PARAMETER_GPRS = { GPR.X10, GPR.X11, GPR.X12, GPR.X13, GPR.X14, GPR.X15, GPR.X16, GPR.X17 };
int NUM_OS_PARAMETER_GPRS = OS_PARAMETER_GPRS.length;

FPR[] OS_PARAMETER_FPRS = { FPR.F10, FPR.F11, FPR.F12, FPR.F13, FPR.F14, FPR.F15, FPR.F16, FPR.F17 };
int NUM_OS_PARAMETER_FPRS = OS_PARAMETER_FPRS.length;
```

Lesson #8: Exception Delivery

Get exceptions to work as quickly as possible!

```
Dumping stack for Thread #6
-- Stack --
(0x000000003a0eeb50 6564)   at [0x000000003a0eeb50, 0x0000000039858048] Lorg/jikesrvm/runtime/RuntimeEntrypoints; deliverHardwareException(
(0x000000003a0eec10 1)   at [0x000000003a0eec10, 0x0000000039858048] <hardware trap>
(0x000000003a0eec28 11977)   at [0x000000003a0eec28, 0x00000000398508c8] Ltest/org/jikesrvm/basic/core/nullchecks/TestNullChecks; invokeHas
(0x000000003a0eec70 11976)   at [0x000000003a0eec70, 0x0000000035379f9c] Ltest/org/jikesrvm/basic/core/nullchecks/TestNullChecks; main([Lja
(0x000000003a0eecb0 -1)   at [0x000000003a0eecb0, 0x00000000352c5464] <invisible method>
(0x000000003a0eecc8 8633)   at [0x000000003a0eecc8, 0x00000000352c5dc0] Lorg/jikesrvm/runtime/Reflection; outOfLineInvoke(Lorg/jikesrvm/cla
(0x000000003a0eee08 8632)   at [0x000000003a0eee08, 0x0000000003531ab6c] Lorg/jikesrvm/runtime/Reflection; invoke(Lorg/jikesrvm/classloader/
(0x000000003a0eee90 9776)   at [0x000000003a0eee90, 0x0000000003531a438] Lorg/jikesrvm/scheduler/MainThread; run()V at line 204
(0x000000003a0eef40 11369)   at [0x000000003a0eef40, 0x00000000353ecc68] Lorg/jikesrvm/scheduler/RVMThread; run()V at line 2731
(0x000000003a0eefb0 11458)   at [0x000000003a0eefb0, 0x000000200002b9f8] Lorg/jikesrvm/scheduler/RVMThread; startoff()V at line 2784
we have a thread = 0x0000000039010958
it's in state = 1
we have exception registers = 0x000000003900d988
Caught NPE while trying to invoke hashCode()
Successfully invoked hashCode() on object
Successfully invoked hashCode() on object
delivering hardware exception
```

Exceptions drop us back into bootloader. Implement exception delivery mechanism early for stack traces!

Trigger exceptions deliberately as part of runtime-checks (e.g., Divide-by-zero, range checks)

```
// Gen bounds check for array load/store bytecodes.
// Does implicit null check and array bounds check.
// Bounds check can always be implicit becuase array length is at negative offset from obj ptr.
// Kills T3
// on return: T0 => base, T1 => index.
private void genBoundsCheck() {
  // TODO: Implement
  popInt(T1);      // T1 is array index
  popAddr(T0);     // T0 is array ref
  asm.emitLWoffset(T3, T0, ObjectModel.getArrayLengthOffset());  // T2 is array length
  ForwardReference fr = asm.emitForwardUnsignedB(BranchCondition.GT, T3, T1);
  asm.emitTrapArrayIndex();
  fr.resolve(asm);
}
```

```
public final void emitTrapArrayIndex() {
    emitLB(ZERO, ZERO, TrapConstants.TRAP_ARRAY_INDEX);
}

public final void emitTrapCheckCast() {
    emitLB(ZERO, ZERO, TrapConstants.TRAP_CHECKCAST);
}

public final void emitTrapDivideByZero() {
    emitLB(ZERO, ZERO, TrapConstants.TRAP_DIVIDE_BY_ZERO);
}
```

Running "Hello World"

```
Booting
Setting up current RVMThread
Doing thread initialization
Setting up memory manager: bootrecord = 0x0000000031000018
Initializing baseline compiler options to defaults
Fetching command-line arguments
Early stage processing of command line
Collector processing rest of boot options
Initializing bootstrap class loader: ../jksvm.jar:../rvmrt.jar
Running various class initializers
running class initializer for java.util.WeakHashMap
invoking method < BootstrapCL, Ljava/util/WeakHashMap; >.<clinit> ()V
running class initializer for org.jikesrvm.classloader.Atom$InternedStrings
invoking method < BootstrapCL, Lorg/jikesrvm/classloader/Atom$InternedStrings; >.<clinit> ()V
running class initializer for gnu.classpath.SystemProperties
invoking method < BootstrapCL, Lgnu/classpath/SystemProperties; >.<clinit> ()V
running class initializer for java.lang.Throwable$StaticData
invoking method < BootstrapCL, Ljava/lang/Throwable$StaticData; >.<clinit> ()V
running class initializer for java.lang.Runtime
invoking method < BootstrapCL, Ljava/lang/Runtime; >.<clinit> ()V
running class initializer for java.lang.System
invoking method < BootstrapCL, Ljava/lang/System; >.<clinit> ()V
running class initializer for sun.misc.Unsafe
invoking method < BootstrapCL, Lsun/misc/Unsafe; >.<clinit> ()V
running class initializer for java.lang.Character
invoking method < BootstrapCL, Ljava/lang/Character; >.<clinit> ()V
running class initializer for org.jikesrvm.classloader.TypeReferenceVector
invoking method < BootstrapCL, Lorg/jikesrvm/classloader/TypeReferenceVector; >.<clinit> ()V
running class initializer for org.jikesrvm.classloader.MethodVector
invoking method < BootstrapCL, Lorg/jikesrvm/classloader/MethodVector; >.<clinit> ()V
running class initializer for org.jikesrvm.classloader.FieldVector
invoking method < BootstrapCL, Lorg/jikesrvm/classloader/FieldVector; >.<clinit> ()V
running class initializer for java.lang.ThreadGroup
invoking method < BootstrapCL, Ljava/lang/ThreadGroup; >.<clinit> ()V
running class initializer for java.lang.ThreadLocal
invoking method < BootstrapCL, Ljava/lang/ThreadLocal; >.<clinit> ()V
running class initializer for java.lang.ThreadLocalMap
invoking method < BootstrapCL, Ljava/lang/ThreadLocalMap; >.<clinit> ()V
running class initializer for java.security.VMAccessController
invoking method < BootstrapCL, Ljava/security/VMAccessController; >.<clinit> ()V
Booting Lock
Booting scheduler
Enabling GC
Setting up boot thread
Initializing JNI for boot thread
JNI initialized for boot thread
running class initializer for java.io.File
invoking method < BootstrapCL, Ljava/io/File; >.<clinit> ()V
running class initializer for java.lang.String
invoking method < BootstrapCL, Ljava/lang/String; >.<clinit> ()V
running class initializer for gnu.java.security.provider.DefaultPolicy
invoking method < BootstrapCL, Lgnu/java/security/provider/DefaultPolicy; >.<clinit> ()V
running class initializer for java.net.URL
invoking method < BootstrapCL, Ljava/net/URL; >.<clinit> ()V
running class initializer for java.net.URLClassLoader
invoking method < BootstrapCL, Ljava/net/URLClassLoader; >.<clinit> ()V
```

```
bash-4.3# ./rvm HelloWorld
Hello World
bash-4.3#
```

Booting JikesRVM requires:

- Loading and initializing **93 classes**
- Support for most Java bytecodes
- Dynamic linking
- Class loading
- JNI calls, including varargs, in both directions (Java -> C, C -> Java)
- Reflection (method invocation)
- Exception delivery across native and JVM stack frames
- Multithreading support
- Locking/Mutex support
- Handling unresolved classes (i.e., dynamic bridge support)
- …

PART III
Status & Next Steps

Status & Next Steps

```
=   * test.org.jikesrvm.basic.core.bytecode.TestResolveOnInvokeInterface [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestFieldAccess [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestClassInitializer [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestInstanceOf [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestArrayAccess [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestMiranda [FAILED]
=   * test.org.jikesrvm.basic.core.bytecode.TestArithmetic [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestResolveOnInstanceof [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestCompare [OK]
=   * test.org.jikesrvm.basic.core.bytecode.TestThrownException [OK]
=   * test.org.jikesrvm.basic.core.classloading.TestUTF8 [OK]
=   * test.org.jikesrvm.basic.core.classloading.TestClassLoading [OK]
=   * test.org.jikesrvm.basic.core.classloading.TestLoadingWithoutName [OK]
=   * test.org.jikesrvm.basic.core.nullchecks.TestNullChecks [OK]
=   * test.org.jikesrvm.basic.core.serialization.TestSerialization [FAILED]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileDoublesUnresolved [OK]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileLongsUnresolved [OK]
=   * test.org.jikesrvm.basic.core.threads.TestDispatch [OK]
=   * test.org.jikesrvm.basic.core.threads.TestInterruptedSleep2 [OK]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileDoubles [OK]
=   * test.org.jikesrvm.basic.core.threads.TestParallelHardwareTrap [OK]
=   * test.org.jikesrvm.basic.core.threads.TestShutdownHookAfterExit [OK]
=   * test.org.jikesrvm.basic.core.threads.TestTimedWait [OK]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileStaticLongsUnresolved [OK]
=   * test.org.jikesrvm.basic.core.threads.TestInterruptedSleep [OK]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileLongs [OK]
=   * test.org.jikesrvm.basic.core.threads.TestShutdownHook [OK]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileStaticLongs [OK]
=   * test.org.jikesrvm.basic.core.threads.TestBackEdgeGC [OK]
=   * test.org.jikesrvm.basic.core.threads.TestNotification [OK]
=   * test.org.jikesrvm.basic.core.threads.TestInterruptedWait [OK]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileStaticDoubles [OK]
=   * test.org.jikesrvm.basic.core.threads.TestVolatileStaticDoublesUnresolved [OK]
=   * test.org.jikesrvm.basic.core.threads.TestSuspend [OK]
=   * test.org.jikesrvm.basic.core.threads.TestTimeSlicing [OK]
=   * test.org.jikesrvm.basic.core.threads.TestInterruptAndSpin [OK]
=   * test.org.jikesrvm.basic.core.reflect.tClass [OK]
=   * test.org.jikesrvm.basic.core.reflect.TestFieldReflection [OK]
=   * test.org.jikesrvm.basic.core.reflect.tArray [OK]
=   * test.org.jikesrvm.basic.core.reflect.tNewInstance [OK]
=   * test.org.jikesrvm.basic.core.reflect.tInstance [OK]
=   * test.org.jikesrvm.basic.core.instrument.TestAgent [OK]
=   * test.org.jikesrvm.basic.bugs.R1722506 [OK]
=   * test.org.jikesrvm.basic.bugs.RVM_915 [OK]
=   * test.org.jikesrvm.basic.bugs.R1657236 [OK]
=   * test.org.jikesrvm.basic.bugs.R1644460 [OK]
=   * test.org.jikesrvm.basic.bugs.R1644449 [OK]
=   * test.org.jikesrvm.basic.bugs.RVM_550 [OK]
=   * test.org.jikesrvm.basic.bugs.RVM_703 [OK]
=   * test.org.jikesrvm.basic.bugs.RVM_708 [OK]
=   * test.org.jikesrvm.basic.bugs.RVM_853 [OK]
=   * test.org.jikesrvm.basic.bugs.R1644460_B [OK]
=   * test.org.jikesrvm.basic.bugs.RVM_914 [FAILED]
=   * test.org.jikesrvm.basic.java.nio.TestFileChannel [OK]
=   * test.org.jikesrvm.basic.java.lang.TestSystem [OK]
=   * test.org.jikesrvm.basic.java.lang.TestMath [OK]
= ==============================================================================
=
= 65 OUT OF 68 TESTS PASSED
=
```

Non-optimizing JIT compiler is mostly feature-complete:

- Currently only targets RV64G
- Passes 65 of 68 core tests

Next steps:

- Fix the last bugs/corner cases
- Run JikesRVM on actual hardware (RocketChip with BOOM)
- Port the optimizing compiler
- Integrate updates to support JDK 8 and OpenJDK class library
- Upstream the port (2017)

WHAT ARE WE DOING THIS FOR?

- JikesRVM enables new managed-language research
- We are using it to investigate hardware support for pause-free Garbage Collection

Grail Quest: A New Proposal for Hardware-assisted Garbage Collection

Martin Maas Krste Asanović John Kubiatowicz
University of California, Berkeley

ABSTRACT

Many big data systems are written in garbage-collected languages and GC has a substantial impact on throughput, responsiveness and predicability of these systems. However, despite decades of research, there is still no "Holy Grail" of GC: a collector with no measurable impact, even on real-time applications. Such a collector needs to achieve freedom from pauses, high GC throughput and good memory utilization, without slowing down application threads or using substantial amounts of compute resources.

In this paper, we propose a step towards this elusive goal by reviving the old idea of moving GC into hardware. We discuss the trends that make it the perfect time to revisit this approach and present the design of a hardware-assisted GC that aims to reconcile the conflicting goals. Our system is work in progress and we discuss design choices, trade-offs and open questions.

1. INTRODUCTION

A substantial portion of big data frameworks – and large-scale distributed workloads in general – are written in languages with Garbage Collection (GC), such as Java, Scala, Python or R. Due to its importance for a wide range of workloads, Garbage Collection has seen tremendous research efforts for over 50 years. Yet, we arguably still don't have what has been called the "Holy Grail" of GC [1]: a pause-free collector that achieves high memory utilization and high GC throughput (i.e., sustaining high allocation rates), preferably without a large resource cost for the application.

Many recent GC innovations have focused on the first three goals, and modern GCs can be made effectively pause-free at the cost of slowing down application threads and using a substantial amount of resources. Moreover, these approaches oftentimes ignore another factor that is very important in warehouse-scale computers: energy consumption. Previous work [2] has shown that GC can account for up to 25% of energy and 40% of execution time in common workloads (10% on average). Worse, as big data systems are processing ever larger heaps, these numbers will likely increase.

We believe that we can reconcile low pause times and energy efficiency by revisiting the old idea of moving GC into hardware. Our goal is to build a GC that simultaneously achieves high GC throughput, good memory utilization, pause times indistinguishable from LLC misses and energy efficiency. We build on an algorithm that performs well on the first three criteria but is resource-intensive [3]. Our key insight is that this algorithm can be made energy efficient by moving it into hardware, combined with some algorithmic changes.

We are not the first to propose hardware support for GC [3–7]. However, none of these schemes has been widely adopted. We believe that there are three reasons:

Garbage-collected languages are widely used, but they are rarely the only workload on a system. Systems designed for specific languages mostly lost out to general-purpose cores, partly due to Moore's law and economies of scale allowing these cores to quickly outperform the specialized ones. This is changing today, as the slow-down of Moore's law makes it more attractive to use chip area for accelerators to improve common workloads, such as garbage-collected applications.

Most garbage-collected workloads run on servers (note that there are exceptions, such as Android applications). Servers traditionally use commodity CPUs and the bar for adding hardware-support into such a chip is very high (take Hardware Transactional Memory as an example). However, this is changing: cloud hardware and rack-scale machines in general are expected to switch to custom SoCs, which could easily incorporate IP to improve GC performance and efficiency.

Many proposals were very invasive and would require re-architecting of the memory system or other components [5, 7, 8]. We believe an approach has to be relatively non-invasive to be adopted. The current trend to accelerators and processing near memory may make it easier to adopt similar techniques for GC without substantial modifications to the architecture.

We therefore think it is time to revisit hardware-assisted GC. In contrast to many previous schemes, we focus on making our design sufficiently non-invasive to incorporate into a server or mobile SoC. This requires isolating the GC logic into a small number of IP blocks and limiting changes outside these blocks to a minimum.

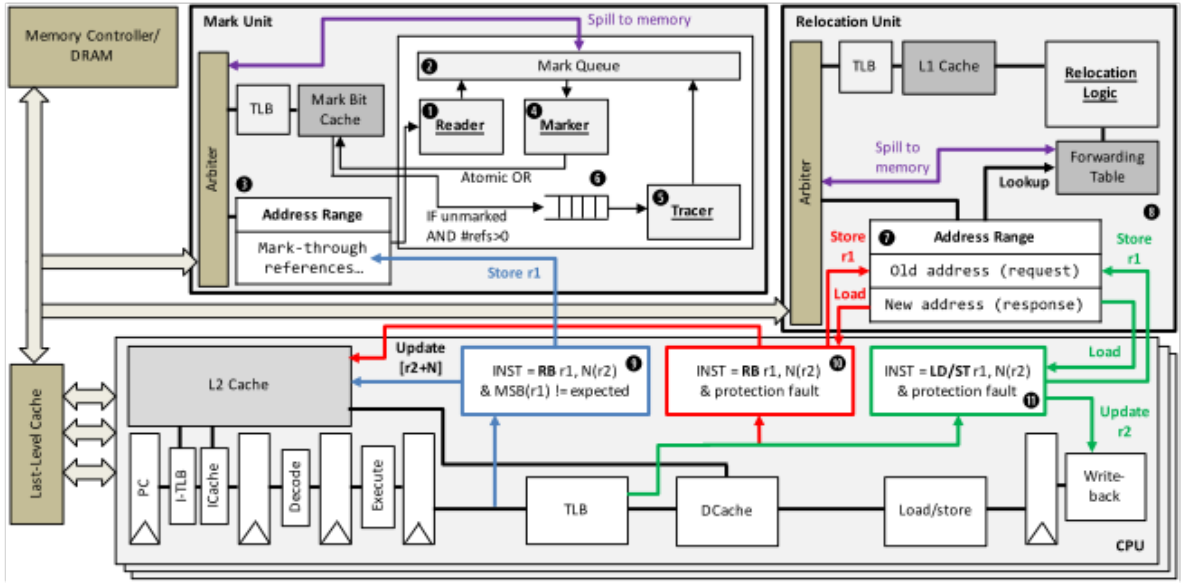
In this paper, we describe our proposed design. It exploits two insights: First, overheads of concurrent GCs stem from a large number of small but frequent slowdowns spread throughout the execution of the program. We move the culprits (primarily barriers) into hardware to alleviate their impact and allow out-of-order cores to speculate over them. Second, the most resource-intensive phases of a GC (marking and relocation) are a poor fit for general-purpose cores. We move them into accelerators close to DRAM, to save power and area.

2. BACKGROUND

An extensive body of work has been published on GC. Jones and Lins [9] provide a general introduction.

There are two fundamental GC strategies: tracing and reference counting. Tracing collectors start from a set of *roots* (such as static or stack variables), perform a

1



JikesRVM Port Runs Actual Programs…

```
==============================================================================
-- JIKES RVM TEST ENVIRONMENT --
==============================================================================
/home/root/BaseBaseMarkSweep_riscv64-linux/tests /home/root/BaseBaseMarkSweep_riscv64-linux
[    6.300000] random: nonblocking pool is initialized
Java Grande Forum Thread Benchmark Suite - Version 1.0 - Section 2 - Size A
Executing on 8 threads

Section2:SparseMatmult:Kernel:SizeA     54.618 (s)       3.6617966       (Iterations/s)
INIT: Switching to runlevel: 0/home/root/BaseBaseMarkSweep_riscv64-linux


==============================================================================
```

…but still has a few bugs

```
==============================================================================
-- JIKES RVM TEST ENVIRONMENT --
==============================================================================
/home/root/BaseBaseMarkSweep_riscv64-linux/tests /home/root/BaseBaseMarkSweep_riscv64-linux
[    6.580000] random: nonblocking pool is initialized
The no of threads has not been specified, defaulting to 1

Java Grande Forum Thread Benchmark Suite - Version 1.0 - Section 3 - Size A
Executing on 1 thread

Validation failed
Pixel checksum = 909613
Reference value = 2676692
Section3:RayTracer:Init:SizeA   0.229 (s)       279.47598       (Objects/s)
Section3:RayTracer:Run:SizeA    12.972 (s)      1734.5051       (Pixels/s)
Section3:RayTracer:Total:SizeA  13.363 (s)      0.0748335       (Solutions/s)
```

Please get in touch if you would like to get involved!

maas@eecs.berkeley.edu