Hardware accelerated Java

Guennadi Liakhovetski, Hua Xueliang

(Contact: jamvm-general@lists.sourceforge.net)

February 16, 2009

Hardware overview

- Hardware platforms
- AVR32: JEM principles

Software

- JVM vs. JRM
- Interpreter design
- Current state of development
- Next steps

Hardware platforms

- ARM: Jazelle technology on ARM7EJ-S, ARM926EJ-S, ARM1026EJ-S, ARM1136J-S, ARM1176J-S... Documentation not available publicly.
- Atmel: Java Extension Module (JEM) on AVR32 SoC. Documentation openly available.

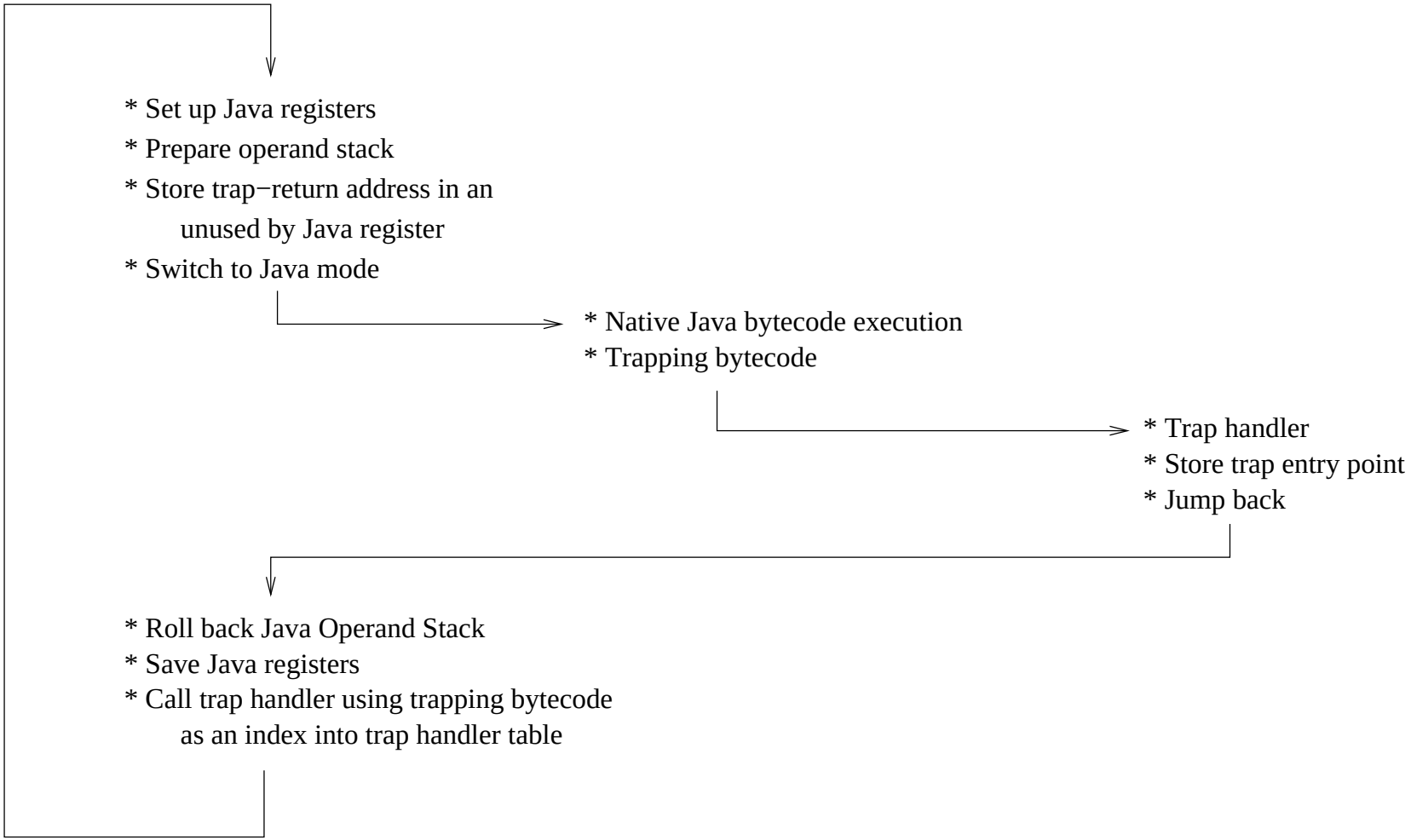
AVR32: JEM principles

- JEM provides AVR32 registers and assembly instructions to access JEM data and enter JEM mode.
- Java bytecodes are executed natively by a hardware module.
- Unsupported instructions and exceptions trap into native AVR32 software:
  - Java exceptions, stack over- and underflow.
  - More than 80 bytecodes always trap: long, FP, double, object manipulation (new, getfield, putfield), invoke / return.
  - Some Java bytecodes trap conditionally.
- Execution mode: trap-table entries only configurable from supervisor mode, the rest is also accessible from the user mode. Documentation unclear about trap entry context.
- H-bit - access to objects via handles, 4 bits available for, e.g., mark-sweep, write barrier - trap when storing an object reference in an object field or an array (MC2 GC).

JVM vs. JRM

- Young but intensively developing Linux port to AVR32 CPUs.
- JVM choice on AVR32 - JamVM unofficial port existed.
- Linux kernel modifications: Java trap table base address, Java context save / restore.
- Simple test program has been written to clarify JEM entry / trap execution.
- New interpreter had to be written for JamVM.

Interpreter design



Current state of development

- First Open-Source hardware accelerated JVM implementation.
- First milestone: a "Hello, World!" Java class executes successfully on JEM.
- kahLua lua interpreter made to run on JEM.
- Code available in a git repository under http://repo.or.cz/w/jamvm-avr32-jem.git.
- First release jem-0.1.
- About 40 traps implemented.

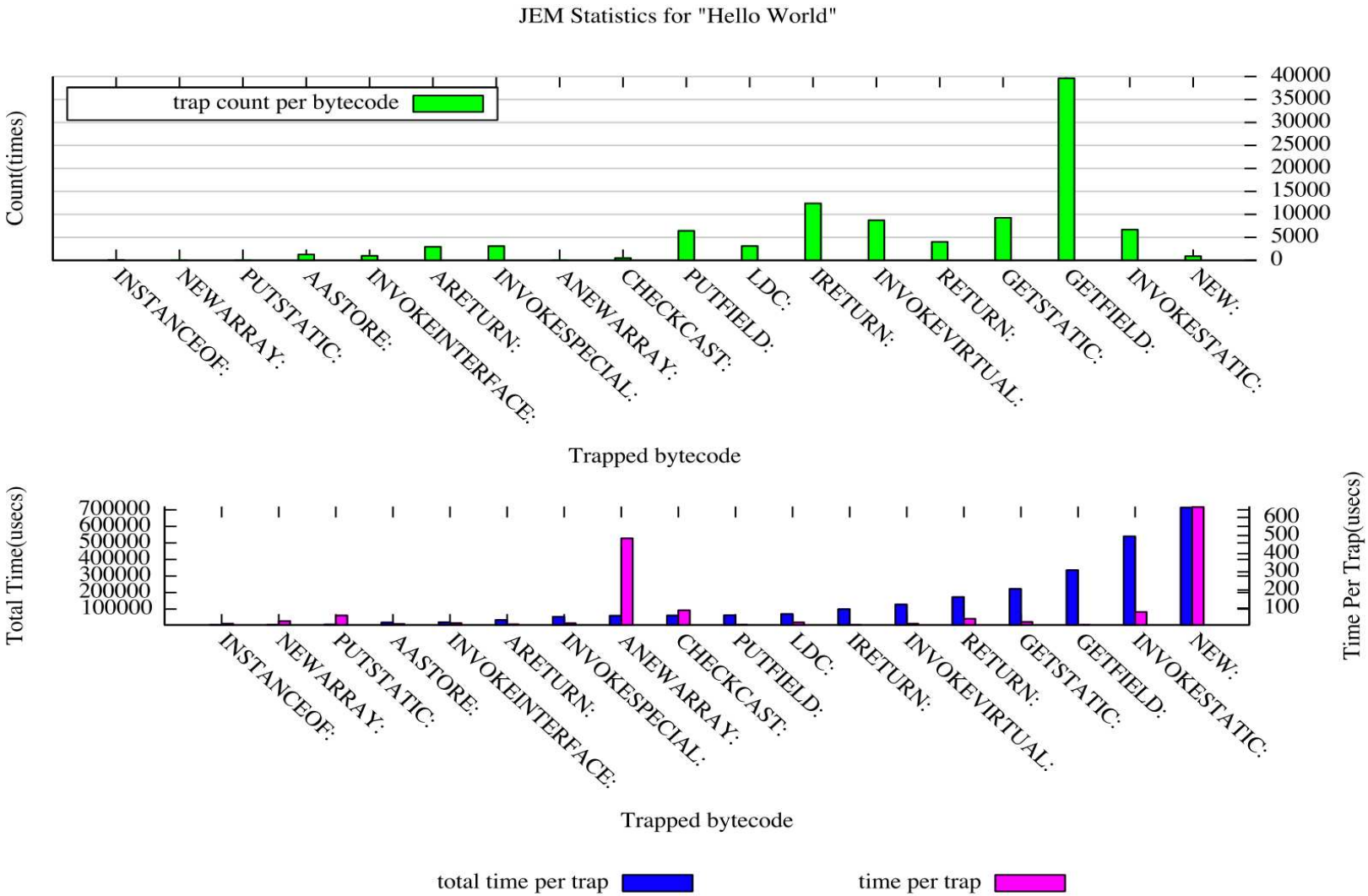
Timing "Hello, World!"

```
~ > time jamvm -cp . helloworld
Hello, World!
real    0m 2.90s
user    0m 1.98s
sys     0m 0.87s
```

Statistics

A total of 103473 Traps:

INSTANCEOF:	0xc1:	299 times,	4333 usec
NEWARRAY:	0xbc:	203 times,	5677 usec
PUTSTATIC:	0xb3:	121 times,	7119 usec
AASTORE:	0x53:	1471 times,	18489 usec
INVOKEINTERFACE:	0xb9:	1185 times,	20385 usec
ARETURN:	0xb0:	3124 times,	33753 usec
INVOKESPECIAL:	0xb7:	3273 times,	53673 usec
ANEWARRAY:	0xbd:	123 times,	59569 usec
CHECKCAST:	0xc0:	693 times,	60761 usec
PUTFIELD:	0xb5:	6587 times,	62690 usec
LDC:	0x12:	3290 times,	69831 usec
IRETURN:	0xac:	12506 times,	98972 usec
INVOKEVIRTUAL:	0xb6:	8861 times,	127220 usec
RETURN:	0xb1:	4204 times,	172445 usec
GETSTATIC:	0xb2:	9411 times,	221579 usec
GETFIELD:	0xb4:	39617 times,	335116 usec
INVOKESTATIC:	0xb8:	6837 times,	539617 usec
NEW:	0xbb:	1087 times,	713657 usec


JEM Statistics for "Hello World"
trap count per bytecode
Count(times)
40000
35000
30000
25000
20000
15000
10000
5000
0
INSTANCEOF:
NEWARRAY:
PUTSTATIC:
AASTORE:
INVOKEINTERFACE:
ARETURN:
INVOKESPECIAL:
ANEWARRAY:
CHECKCAST:
PUTFIELD:
LDC:
IRETURN:
INVOKEVIRTUAL:
RETURN:
GETSTATIC:
GETFIELD:
INVOKESTATIC:
NEW:
Trapped bytecode
Total Time(usecs)
700000
600000
500000
400000
300000
200000
100000
Time Per Trap(usecs)
600
500
400
300
200
100
Trapped bytecode
total time per trap
time per trap

Next steps

- Optimisations.
- Remaining traps.
- Multithreading, kernel support for context save/restore.
- Garbage collection.
- More test programs.
- H-bit.
- Clean up.