Towards a Smalltalk VM for the 21st Century

Boris Shingarov
www.shingarov.com

Challenges of Shifting Reality

Computing is changing

- What we run on Smalltalk
- What we run Smalltalk on

What we run on Smalltalk

Complex workload on Smalltalk runtime

- Complexity of today's average VM bug
  - parallelism, race conditions, complex optimiations
- "High-level debuggers" do not work
  - this is not a C application
- In-band solutions (e.g. DTrace-like) partially help

What is "Performance"?

- Traditional: "Let's make Smalltalk fast"
  - very fast PICs, JIT using ILP hardware features
- We no longer fit the traditional model
- Example: Big Data applications challenge the object graph model
  - Packed objects required for less FFI marshalling overhead and improving cache performance
- New generation of hardware dictating radically new performance metrics
  - What we run Smalltalk on

Old vs New "Conventional Wisdom"

(after Dave Patterson, ACM President)

- Design Cost Wall
- Software Legacy Wall
- Power Wall
- Memory Wall
- ILP Wall

End of Uniprocessor Era

Old vs New "Conventional Wisdom"

Demonstrate new H/W ideas by building chips	No researchers can build believable prototypes
H/W hard to change, S/W flexible	H/W flexible, S/W hard to change
Power is free, transistors expensive	Can put more transistors on chip than can afford to turn on
Multiply slow, memory access fast	1 RAM access ≈ 200 clocks
Increasing ILP: RISC, compilers, out-of-order, VLIW, speculation etc	Diminishing returns on more ILP H/W

VM Observation: In-band vs out-of-band

- VM "communicates" with the processor via the Processor Architecture
  - VM -> Processor: instruction stream
  - Processor -> VM: flags/branching, interrupts
- In-band observation agents are inherently limited in scope and access, and destructive to machine state
  - stopping at breakpoint destroys the state of memory hierarchy
  - (no access to cache details anyway)

VM Observation: In-band vs out-of-band

- Out-of-band: VM introspection channels outside of Processor Architecture
  - Invisible to both VM and Processor
  - Varying levels of fidelity
- Out-of-band examples:
  - Processor functional simulation
  - Hardware-level processor modeling/simulation
    - Instrumented FPGA models
    - Hardware simulation in software

Full-system simulation

- Simics
- GEMS
- M5
- GEM5
- OVPsim

Characteristics of FSS

- Timing Abstractions, levels of accuracy (Software Timed (e.g., QEMU, IBM CECsim), Loosely Timed, Approximately Timed as in TLM-2.0); Temporal Decoupling
- Observability — full recording of simulation makes possible arbitrarily complex analysis of interaction between any parts of the systems (e.g. signals not hidden on an internal bus of a SoC); stopped time, time warping
- Checkpointing (persisting full state of simulation), useful for optimizing workflow, communication between teams (e.g. to reproduce a bug), switch between levels of simulation detail
- Dynamic Reconfiguration
- Repeatability
- Reverse execution
- Intelligent OS awareness

Modules

Simulators are modular and expose an open set of APIs.

- devices, memory, systems, processors
- even the foundation of simulation — the time model
- modules allow full awareness of software running on the simulated system
    - OS awareness (example: Linux process tracker)
    - full symbolic debugging (C)
    - enables full awareness of Smalltalk

Demo: Emitting a magic instruction in JIT

- A "Magic Instruction" causes simulation breakpoint
- Example of Program-Simulator signalization
- Simics Magic Instructions on different architectures:

Target	**Magic instruction**	
x86	xchg %bx, %bx	*encoding: 66 87 DB*
ARM	orreq rn, rn, rn	*0 <= n < 15*
PowerPC 32-bit	mr n, n	*0 <= n < 32*
PowerPC 64-bit	fmr n, n	*0 <= n < 32*
SPARC	sethi n, %g0	*1 <= n < 0x400000*

Modify the JIT translator

- CogIA32Compiler>concretizeMagic
- CogIA32Compiler>dispatchConcretize
- CogRTLOpcodes>>initialize

  Add "Magic" to the end and send *#initialize*. Now our abstract RTL has the magic instruction.

- Cogit>>Magic

```
Cogit>>Magic
   <inline:true>
   <returnTypeC:#'AbstractInstruction*'>
   ^self gen: Magic
```

Use the instruction somewhere...

Why not in, say, #genGetClassFormatOfNonInt:into:scratchReg:?

NB: What we are doing is adding to the code emission code, but the actual magic instruction will be part of the emitted N-code, so the break will NOT happen in #genGetClassFormatOfNonInt:into:scratchReg:.

Try simulating it...

- Regenerate VM sources
- Compile the VM
- Run under simulation
  - ./NBCog --nodisplay simple.image eval '2+3'
- Now with *magic-break-enable*

Let's look around

simics> pregs

32-bit legacy protected mode
eax = 0x00000001, ax = 0x0001, ah = 0x00, al = 0x01
ecx = 0x00000006, cx = 0x0006, ch = 0x00, cl = 0x06
edx = 0x944f9540, dx = 0x9540, dh = 0x95, dl = 0x40
ebx = 0x00000004, bx = 0x0004, bh = 0x00, bl = 0x04
esp = 0xbf869670, sp = 0x9670
ebp = 0xbf869680, bp = 0x9680
esi = 0x00000003, si = 0x0003
edi = 0x00000003, di = 0x0003

eip = 0x93cff260, linear = 0x93cff260

eflags = 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 = 0x00000202

receiver in %EDX

Let's look around (cont.)

simics> x l:0x944f9540 4

l:0x944f9540 1d04 9a12 *<- receiver's object header*

class oop in header word 2, offset -4:

simics> x l:0x944f953C 4

l:0x944f9530 813e 1194 *<- class oop*

A rudimentary Smalltalk module

Look at the object memory using Simics Python API

```
def print_class_of_oop(oop):
    if ((oop & 1)==1):
       print "SmallInteger"
    else:
       headerType = smalltalk_headerType(oop)
       if (headerType==3):
          print "...looks like compact class..."
       else:
          word2 = read_virt_value(oop-4, 4)
          classOop = word2&0xFFFFFFFC
          print "class oop: ", hex(classOop)
          classNameOop = read_virt_value(classOop+32, 4)
          print "class name oop: ", hex(classNameOop)
          str=""
          for offset in range(smalltalk_objByteSize(classNameOop)):
              str += "%c" % read_virt_value(classNameOop + 4 + offset, 1)
          print str
```

Make it into a command...

```
new_command("print-class-of-oop", print_class_of_oop,
            [arg(int_t, "oop")],
            type = "Debugging",
            see_also = [],
            short = "describe an oop",
            doc = """
Print the class of oop.""")
```

Try printing classes of some OOPs...

```
simics> print-class-of-oop %edx

class oop:  0x94113E80L
class name oop:  0x93F401CCL
WeakAnnouncementSubscription

simics> print-class-of-oop 0x93F401CC

class oop:  0x940FB4B4L
class name oop:  0x93E94140L
ByteSymbol
```

Closer to practice

- Debugging a SIGSEGV
- Put simulation breakpoint in SEGV handler
- Because everything is recorded, we can step back to the source of the bug
- Can solve bugs that are hopelessly complex for in-band approach

Digging deeper — MAI mode

- Vanilla Simics operates at instruction level
- Micro Architectural Interface allows detailed modeling of processor pipelines, out-of-order-execution and timing of cache hierarchies
- Simulation speed / fidelity tradeoff
- Save checkpoint state and simulate detail of only the interesting pieces

Digging even deeper

- GEM5
  - www.m5sim.org
- detailed full-system and microarchitectural models
- Ruby memory hierarchy system
- Cache coherence modeling
- Opal (aka TFSim) — SPARCv9 out-of-order processor
- AtomicSimple / TimingSimple / InOrder / O3CPU processor models

Can we go even further?

- looking at what's happening inside the processor in even more detail
- Hardware to exeriment with processors is within reach
- Open-source hardware IP has matured: Practically important processors and SoCs in production

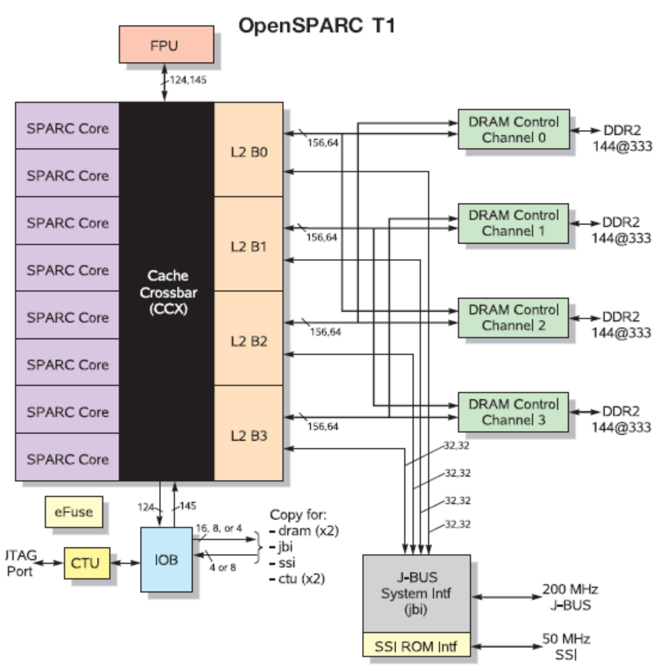
Some interesting processors

- OpenSPARC
  - implements SPARCv9 (64-bit) ISA
  - T1: 8 cores x 4 threads UltraSPARC released as open-source Verilog
- LEON3
  - SPARCv8 (32-bit) ISA
  - Used by major aerospace projects (ISS...)
- Storm ARM processor and SoC
- ATLAS
- Amber (ARM ISA compatible)

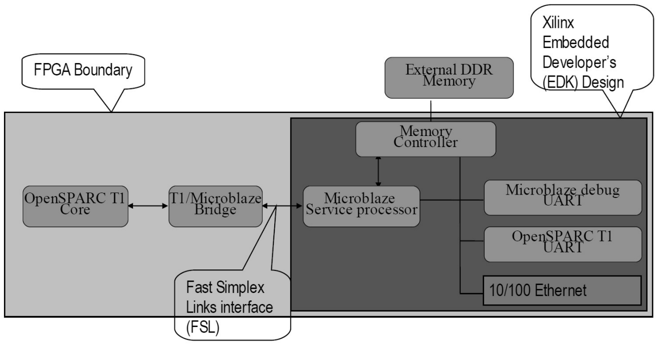
Smalltalk instrumentation

- Out-of-band introspection of OpenSPARC on Xilinx Virtex-5
- Debug interface based on the MicroBlaze service processor (same core running the CCX)
- Physical communication over JTAG

Basic T1



Implementation on Virtex-5



Even more challenges

Speed vs Power

- Power-Optimized JIT
  - Instruction selection, intruction scheduling etc.
- explicit power mamagement
  - voltage scaling
- $W = U^2 / R$
- You don't want to run as fast as you can
  - missing integration mechanism to tell

References

- G.Wright et al.: *Introspection of a Java Virtual Machine under Simulation.* SMLI TR-2006-159, Sun Microsystems, 2006.
- R.Leupers, O.Temam (eds.): *Processor and System-on-Chip Simulation.* Springer, 2010.