High Performance Microprocessor Emulation for Software Validation Facilities and Operational Simulators

Dr. Mattias Holm <maho@terma.com>
Boot Firmware

Computer Architecture

Compiler Technology

Interpreters

Operating Systems
Emulation

- Instruction level simulation of a CPU
- Executes the emulated target's instruction set in a virtual machine running on a host.
- Simulates memory and MMU
- Simulates peripheral devices and/or provides a way to integrate external devices
- Performance measured in MIPS (millions of emulated instructions per second)
void emulate(cpu_t *cpu, mem_t *mem) {
    while (1) {
        if (setjmp(cpu->jmpbuf) == 0) {
            while (cpu->running) {
                instr = fetch(mem, cpu->pc);
                semfunc = decode(instr);
                semfunc(cpu, mem, instr);
            }
        } else {
            handleTrap();
            if (!cpu->running) return;
        }
    }
}

instfunc_t decode(u32 inst) {
    switch (((inst >> 30) & 3) {
    case 0:
        switch (...) {...}
        break;
    case 1:
        switch (...) {...}
        break;
    case 2:
        switch (...) {...}
        break;
    case 3:
        return callInstFunc;
    } return illegalInstFunc;
}
Methods for Instruction Set Simulation

- **Interpretation** (~120 MIPS theoretical on 3.5 GHz x86-64)
  - Decode-dispatch or threaded
  - Indirect / direct
  - Pre-decoded

- **Binary translation** (~350 MIPS on 3.5 GHz x86-64)
  - Dynamic
  - Static
  - Basic-block chaining
  - Return stack shadowing
Emulation: Parts of an Emulator (1/3)

- **Instruction decoder (several of them)**
  - One for the interpreter
  - One for the binary translator
  - One for the assembler living down the lane...

- **Instruction semantics**
  - One routine per instruction
    - May be in variants (e.g. arithmetic instructions with %g0 as destination)
  - Binary translator and interpreter need different types of instruction descriptions...
    - Write two... or rather not.
    - Write one and #ifdef yourself around issues
    - Write one and transform it to the relevant format
Emulation: Parts of an Emulator (2/3)

- **Memory System**
  - ROM and RAM
  - MMIO (see below)
  - MMU model

- **Device Models**
  - Memory Mapped I/O models
  - Bus connected models
  - Time triggered environmental models

- **User Interface**
  - Command Line Interface
  - GDB Interface
  - APIs
  - Others
Emulation: Parts of an Emulator (3/3)

- **Events**
  - Scheduled events (driven by CPU time)
  - Memory access events
  - Named notifications (e.g. raised trap, bus message sent etc)

- **Publication (in T-EMU)**
  - Checkpointing/breakpointing of simulation state

- **Other interfaces**
  - Interrupt raising etc
Emulation

- **Common Implementation Languages**
  - **Assembler**
    - Useful for interpreters
    - Can fine tune
    - Not portable
  - **C**
    - Usually not fast enough for interpretation (except when threading code...)
    - Can implement dynamic code generator reasonably efficiently
  - **Custom languages / DSLs**
    - Portable (depending on DSL compiler)
    - High performance
    - Easy to maintain but may need significant resources for in-house maintenance of the DSL.

- **T-EMU 2 uses the LLVM toolchain**
  - TableGen for instruction decoders
  - LLVM Assembler for instruction semantics (embedded in TableGen files)
T-EMU 2: TableGen CPU Descriptions

multiclass ri_inst_alu<bits<2> op, bits<6> op3, string asm, code sem> {
    def rr : fmt3_1<op, op3> {
        let AsmStr = asm # " {rs1:gpr}, {rs2:gpr}, {rd:gpr}";
        let Semantics = [{
            %r1 = call i32 @emu.getReg(%cpu_t* %cpu, i5 %rs1)
            %r2 = call i32 @emu.getReg(%cpu_t* %cpu, i5 %rs2)
        }] # sem # [{
            call void @emu.setReg(%cpu_t* %cpu, i5 %rd, i32 %res)
        }]
    }

    def ri : fmt3_2<op, op3> {
        let AsmStr = asm # " {rs1:gpr}, {simm13}, {rd:gpr}";
        ...
    }

    defm add : ri_inst_alu <0b10, 0b1010101, "add", [{
        %res = add i32 %r1, %r2
    }]>;
State of the Art

• **Binary translators**
  - OVPSim
  - Windriver Simics (~350 MIPS)
  - QEMU (partially GPL → no use in certain industries)

• **Interpretation (SPARC emulators)**
  - TSIM (~60 MIPS)
  - ESOC Emulator (65 MIPS no MMU, 25 MIPS with MMU)
  - T-EMU 2...

• **Others**
  - Countless of game console emulators etc
T-EMU 2: The Terma Emulator

**T-EMU 1:**
- Derivation of ESOC Emulator Suite 1.11
- Formed the baseline for the work on ESOC Emulator Suite 2.0
- Written in EMMA: The Extensible Meta-Macro Assembler (embedded assembler, using Ada as host language)
- Emulates
  - MIL-STD-1750A/B
  - SPARCv8 (ERC32, LEON2, LEON3)

**T-EMU 2:**
- Complete rewrite
- Using modern C++11 and LLVM
- LLVM compiler tools are used extensively
- Interpreted, but ready to upgrade with binary translation capabilities
- Significant work spent on defining a device modelling APIs
  - Can easily be wrapped for scripting languages (e.g. prototype your device model in Python) or SMP2 (an ESA standard for simulation models)
- Can emulate multi-core processors
- Emulates SPARCv8 (ERC32, LEON2, LEON3, LEON4)
T-EMU 2: Features and Models

- **Processors**
  - ERC32
  - LEON2
  - LEON3
  - LEON4
  - NOTE: SMP and multi-core processors can be emulated.

- **Buses**
  - AMBA
    - PNP supported for AMBA devices
  - Serial
  - MIL-STD-1553
  - Spacewire
  - Ethernet (in development)

- **Models**
  - On-Chip Devices
    - MEC (ERC32)
    - LEON2 on-chip devices
  - GRLIB:
    - AHBCTRL, AHBSTAT, AHBUART
    - APBCTRL, APBUART, FTMCCTRL
    - GPTIMER, IRQMP
    - Additional added as we go along.
T-EMU 2: The Terma Emulator

- **Library based design**
  - Easy to integrate in simulators
  - Public stable API is C (i.e. can integrate with just about anything).
  - Easy to use emulator framework as a simulator driver.

- **Command Line Interface**
  - Assisting with emulator and model development and integration
  - Embedded / on-board software development (e.g. unit tests)
Performance Measures

- Threaded interpreter
- Instruction implementations hand crafted in LLVM assembler (minimising instruction counts)
- Instruction decoders are machine optimised
- Address decoders are $O(1)$ and about 2 loads, 2 shifts, 2 masks and 1 or 2 branches plus one function call
  - Contrast to linear search of unsorted lists done in many simulators
- Fast path address cache lookups (soft TLB)
- Custom LLVM optimisers
  - Threading hot variables
  - Spill point optimisers
  - Others
- Every cycle counts!
Current Interpreted Emulator Performance

- 3.5 GHz x86-64
- ESOC Emu numbers are for the stock ESOC Emu configuration available to us, without MMU.
- TSIM numbers from http://www.gaisler.com/
- Anything above 50 MIPS is high performance for an interpreted emulator
T-EMU: General Future Directions

- Binary translation (>300 MIPS)
- Additional architectures (ARM, PowerPC, MIPS etc)
- Direct support for more ways for device modelling:
  - SMP2
  - HDLs: System-C, VHDL, Verilog etc
  - Custom domain specific language (code size reductions of over 60% in some cases for a prototype)
- Bigger model library:
  - Provide models for all common spacecraft processors and peripherals
- Reverse Execution!!!
- Source level debugging
- And a lot more, roadmap will keep us busy for years.
Emulation in simulation

- **Software Validation Facilities and operational simulators.**
- **Approaches**
  - Systems approach: Emulator embedded in sim framework as “just another model”. Many events (e.g. bus transactions) will be scheduled by the emulator...
    - Two schedulers (sim framework and the emulator model)
    - Difficult (but not impossible) to achieve full determinism
    - Domain mismatch: emulator run event is not a discrete event, it is an interval event
  - Software centric: Embed simulation models in emulator, requires emulator with support for scheduling, scripting, model publication etc.
    - One scheduler
    - Control of on-board software down to the cycle
    - Fully deterministic = reversible

- Many modelling frameworks have been designed by systems people without input from emulator designers.
- Emulator often key in performance of simulator.
- Emulators are simulators.
  - Sims and emulator communities should talk more.
Questions?

Flyers available!

http://t-emu.terma.com/

PoCs:
- Technical: Dr. Mattias Holm <maho@terma.com>
- Sales: Roger M. Patrick <rmp@terma.com>
Further Reading

- Dynamically Translating x86 to LLVM using QEMU: Vitaly Chipounov and George Candea, 2010
- Using the LLVM Compiler Infrastructure for Optimised Asynchronous Dynamic Translation in QEMU, Andrew Jeffery, 2009
- LnQ: Building High Performance Dynamic Binary Translators with Existing Compiler Backends, Chun-Chen Hsu et.al, 2011