FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators

Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil Patil, William Reinhart, D. Eric Johnson, Jebediah Keefe, Hari Angepat

University of Texas at Austin
Electrical and Computer Engineering

DOE Career, NSF, SRC, Intel, IBM, Freescale, Xilinx
First, Some Terminology

- **Host**: the system on which a simulator runs
  - Dell 390 with a single 1.8GHz Core 2 Duo and 4GB of RAM
  - A Xilinx FPGA board
- **Target**: the system being modeled
  - Alpha 21264 processor
  - Dell 390 with a single 1.8GHz Core 2 Duo and 4GB of RAM

Target
- Alpha 21264

Simulator
- Simplescalar (sim-alpha)

Host
- Your desktop
FAST Computer System Simulator Goals

- **Fast**: as fast as possible
  - ~ 2-3 orders of magnitude slower than target
  - Fast enough to run real datasets to completion
  - Useful to software developers (performance tuning)?
- **Timely**: Available during architectural phase
- **Accurate**: produce cycle-accurate numbers
- **Complete**: run unmodified operating systems, applications, peripherals…
- **Transparent**: full visibility, no performance hit
- **Relatively Inexpensive**
- **Flexible**: quick changes
- **Complex Targets**: x86, PowerPC, OoO
FAST Prototype in Real Time
Speculative Functional/Timing Partitioning

- Proven partitioning (FastSim)
  - FM executes instructions, pushes instruction trace to TM
  - If functional path != timing path, TM forces FM to rollback
    - Eg.: branch mis-speculation, resolve
  - Clean inst trace/rollback interface

Functional Model
  - (ISA + peripherals)

Timing Model
  - (Micro-architecture)
What Is A FAST Functional Model?

- **Requirements**
  - Fast
  - Full System
  - Generates instruction trace
  - Supports rollback

- **Hardware functional models (very fast)**
  - Real processor doesn’t support trace/rollback
  - FPGA implementation difficult to make complete
    - x86, boots Windows?

- **Software functional models exist today**
  - Bochs, QEMU, Simics, SimNow, SimOS, etc.
  - Relatively fast, full system
    - *Run on fastest hardware we know about to execute an ISA*
  - Can be modified to generate trace/support rollback
What is a FAST Timing Model?

Stats gathering in hardware => no performance impact

Step 1: Improving Performance via Parallelization

- Parallel slowdown due to communication?
  - FM runs ahead, speculatively, round-trip communication infrequent
  - Round-trip communication only when (functional path ≠ timing path)
- Microprocessors have same problem
  - Multiple issue, deep pipelines only work if predicted path is correct
    - FM like perfect front end of processor, real uArch (TM) slows it down
    - The better the target micro-architecture, the faster the simulator
Step 2: Parallelizing Timing Model

- Software timing model is bottleneck. Parallelize?
  - Difficult to parallelize in software (very tight dependencies)
  - Practical limitation of number of processors that can communicate quickly
- Hardware-based (FPGA) timing model
  - Parallelizes nicely in hardware
  - TM very simple since does not implement functionality
  - Latency tolerant, infrequent round-trips
Prototype Overview

- Software functional model
  - Eventually hardware functional model, but software sim exists
- FPGA-based timing model written in Bluespec
  - Complex OoO micro-architecture fits in a single FPGA
- DRC or XUP

Diagram:
- Functional Model
  - Software
- Timing Model
  - Bluespec HDL
- PowerPC 405
- Xilinx FPGA
- DRC Computer
Current Prototype Functional Model

• Derived from QEMU
  • Fast (JIT), boots Linux, Windows
  • Supports x86, x86-64, PowerPC, Sparc, ARM, MIPS, …

• Prototype currently supports x86
  • Added tracing, rollback (implemented with checkpoint)
    • Including I/O (keyboard, mouse, video)
  • Hosts
    • x86 machines
    • PowerPC inside of an FPGA

• PowerPC target by January
  • about 1 month to port
Modular Timing Models: Modules + Connectors

- Modules model timing functionality
  - E.g., rename, caches, etc.
  - Built hierarchically for extensibility
    - CAM, FIFOs, arbiters, etc.
    - Branch predictors, Caches, TLBs, Schedulers, ALUs
    - Fetch, Decode, Rename, RS, ROB
  - Many are essentially wires (e.g., ALU)
  - Often written to execute one operation
    - E.g., Rename, Cache
    - Executed multiple times per target cycle for wider processor, higher associativity
    - Simplifies implementation, tradeoff time for space

- Connectors connect modules
  - Abstract timing from modules
    - Throughput (input, output), delay, maxTransactions
  - Stats and tracing
Microcode Compiler

- Intention
  - Automate generation of new ISA instructions
  - Automatically retarget new micro-architectures
  - Necessary for x86
- Uses the LLVM Compiler Infrastructure developed at UIUC
  - http://www.llvm.org
- Compile the Bochs CPU model
  - Bochs is another portable x86 full-system emulator.
  - Backend retargeted to micro-op ISA
- Generates microcode that “runs” on the timing model

- Over 99% dynamic inst coverage for most INT benchmarks
  - floating point instructions not yet supported
- Average 1.27 uOps per handled dynamic x86 instruction
Current Simulator Performance on DRC

Includes Operating System Code

- gshare
- BP 97%
- BP 100%

MIPS

Linux Windows XP 164.gzip 175.vpr 176.gcc 181.mcf 186.crafty 197.parser 252.eon 253.perfbrmk 254.gap 255.vortex 256.lzzip2 300.twolf amean
Performance Details

- Timing model is current bottleneck
  - 100MHz host cycle (not pushing timing)
  - Currently taking ~30 host (FPGA) cycles per target cycle, max about 54 cycles (currently max latency defines target clock)
  - BP is a simple gshare predictor

- Functional model
  - Unoptimized modified QEMU
  - With perfect BP, immediate return from TM, 5.4MIPS

- FM/TM communication
  - 469ns blocking read from Opteron on DRC (has gotten better)
    - Poll every other basic block
  - 13ns/word for burst write
Some Related Work (there is a lot)

- **Software**
  - Functional/timing partitioned
    - Asim, current M5, Timing-First, Opal all timing model driven
      - Timing model tells functional model what to do and when to do it
    - FastSim (Schnarr, et al, ASPLOS 98)
      - Functional/timing, rollback when functional path != timing path
      - But, instrumented binaries, not parallelized, no hardware
  
- **Hardware**
  - HASim: Hardware ASim (Emer, et. al)
    - Timing-first
    - Seven points of communication between FM & TM
      - Requires infinitely renamed out-of-order FM
    - Current supports a simplified MIPS ISA
Conclusions/Future Work

- It works
- Current FAST simulator prototype
  - 1.2MIPS (unoptimized), about 1000 times slower than target
  - Timely: during architecture phase
  - Complete: runs Windows, Linux
  - Transparent: extensive, hardware-based stats
  - Relatively inexpensive, easy to build and extend
- (Some) future work
  - Optimize
    - 5MIPS soon, 10MIPS-20MIPS later (hardware FM using uCode?)
  - More realistic timing model & calibration
  - Tattler: automatic bottleneck detection
  - CMP/SMP targets