ProtoFlex: FPGA-Accelerated Full-System MP Simulation

Eric S. Chung, Eriko Nurvitadhi, James C. Hoe, Babak Falsafi, Ken Mai

Our work in this area has been supported in part by NSF, IBM, Intel, and Xilinx.

Tutorial, June 2007
Multiprocessor Emulation

• Need fast MP emulators to study future generations HW and SW

  state-of-the-art simulation is ~100·N slowdown

• Hardware concurrency of FPGA emulation can scale up multiprocessor simulation speed

• BUT, we also want full-system fidelity (OS and I/O support)

How to build a 1000-node fully detailed MP system? FPGA or not?
How to get full-system fidelity without building the full system?
Combining Simulators & FPGAs

• Simulators already provide full-system behaviors
  ⇒ why not just simulate infrequent behaviors (e.g., I/O devices)?

• Advantages
  – avoid implementing infrequent behaviors
    ⇒ simplify full-system emulator development
  – low impact on scalability and performance acceleration
Hybrid Full-System Emulation

- 3 ways to map target component to hybrid emulation host
  - Emulation-only
  - Simulation-only
  - Transplantable

- CPUs can fallback to SW by “transplanting” between hosts
  - Only common-case instructions/behaviors implemented in FPGA
  - Remaining behaviors relegated to SW (includes many of the most complex ones)

Transplants reduce full-system design effort
It Really Works

Xilinx XUP Virtex-II Pro 30

Our SPARCv9 core

Embedded PowerPC

DDR memory

Virtutech Simics (commercial simulator)

Simics UltraSPARC

Simulated target devices

Transplant & message interface

Ethernet

“SUN 3800 Server”
(1x UltraSPARC III, Solaris 8)

developed in 6 months
It Really Works

Xilinx XUP Virtex-II Pro 30

Our SPARCV9 core

Embedded PowerPC

Virtutech Simics (commercial simulator)

Simics UltraSPARC

Simulated target devices

Transplant & message

BlueSPARC specs:

• 7k lines Bluespec HDL (compiles to Verilog)
• UltraSPARC III ISA
• Validated against Simics w/ real apps (e.g., Solaris 8, SPEC2000, DB2, Oracle, etc.)
• 41% all instr groups implemented + MMU
• 8kB I/D direct-mapped caches
• multi-cycle func model ($CPI_{ideal} = 5 @ 100MHz$)
• 16K LUTs (50% of XUP Virtex-II Pro 30)
Performance Reality Check

• The latest BlueSPARC core = 100 MIPS raw
• Transplant overhead = 10 millisec or 1 million cycles

\[ \text{MIPS}_{\text{effective}} = \frac{\text{MIPS}_{\text{raw}}}{(1 + \text{MIPS}_{\text{raw}} \cdot \text{rate}_{\text{xplant-per-million}} \cdot 0.010)} \]

Even if just 1 in 1 million instructions requires transplanting, effective MIPS goes down to 50 MIPS

• Big Problem
  – decreasing transplant rate any further would require the escalating effort of implementing many “rare” and difficult instructions
  – increasing MIPS_{raw} increases the discounting factor such that 200 MIPS_{raw} would yield only 66 MIPS_{effective}

Can’t fight against diminishing return!
Reducing transplant overhead

- Sanity check: transplants are expensive (over ethernet ~ 10ms)
  - given CPI = 1 @ 100 Mhz (100 MIPS), 1 transplant per 1 million instructions increases CPI to 2 (50 MIPS)
- Recall lessons in hierarchical cache design ...
- Micro-transplants
  - Run “simulator kernel” on nearby embedded hard core (e.g., PowerPC on Xilinx)
  - Carries out unimpl instructions
  - Only I/O goes to full-sys simulator

\[
\text{CPI}_{\text{effective}} = 2 \quad \text{CPI}_{\text{effective}} = 1.1
\]

- FPGA fabric
  - coverage=99.9999%
  - CPI\text{raw} = 1

- Embedded PPC ISAsim
  - coverage=99.99999%
  - CPI=1,000

- full-system simulator
  - coverage=100%
  - CPI\text{tplant} = 1,000,000
How to build a 1K-node MP emulator, without building 1024 nodes?
How fast do you need to simulate?

**User Expectations:**

- 10-100x slowdown is OK for interactive use (ex: simics)
- 1k-10k slowdown OK for functional design exploration (ex: cache-sim)
Different ways to simulate 1K cores

• Even for a 1K-node MP, only need 1000 to 10,000 MIPS (in aggregate) to do useful work

• The naïve approach
  – build a fast ISA core (estimate 100 MIPS per core)
  – physically replicate the core 1000 times

  ⇒ 10x to 100x faster than it needs to be
  ⇒ Why spend effort on performance I don’t need

• The better approach—think in terms of MIPS
  – build a 100-MIPS ISA core with a interleaved pipeline that can support multiple contexts
  – interleave 100 contexts per core to emulate a 1K-node system with just 10 physical cores

  ⇒ the parameters and the effort required can be tuned to make the emulator just fast enough and not more
PROTOFLEX\textsuperscript{MP}

- Build a 1000-MIPS simulator from 10s of FPGAs
  - maximize throughput per emulation engine to be shared by multiple interleaved contexts
  - multiplex a large number of emulated contexts onto a few emulation engines

*Base the number of emulation engines you need on how much performance you need, and not on how many nodes you are emulating*

N-way target system

P-way FPGA emulation engines, P<<N
Interleaved Emulation Engine

• Each emulation engine is essentially an interleaved multithreading datapath
  – has simpler pipeline without forwarding or interlock
  – can have deeper pipelines for higher frequency
  – can help hide the latency of memory and transplant

⇒ it is actually easier to optimize this pipeline
⇒ get cache coherence “free” within the same engine

• Issues to work out
  – How to manage a very large number of core contexts? Can we dynamically “page” clusters of contexts in and out of the core?
  – How to “fake” memory capacity? How much DRAM do you need to emulate a 1000-node system?
  – Lots of interesting problems left
Work in progress

Bluesparc\textsuperscript{MP} engine design:
- Interleaved, single-issue, 9 stages w/ fair scheduling
- 16 UltraSPARC III (64-bit) contexts
- 100 MIPS @ 100 MHz
- Written in Bluespec HDL: “looks” like simulator—can be modified/traced

Status report:
- Started design from scratch in March
- 36 of 46 planned instr groups done
- Plan to run full-sys commercial apps
- Synth estimates: 25kLUTs on V2P70 (35%), 218 BRAMs (66%)
- We are starting board tests on BEE2
What about performance simulation?

• ProtoFlex facilitates performance simulation via simulation sampling [Wenisch et. al IEEE Micro, Aug 2006]
  – estimate accurate performance measurements by sampling only many small segments of execution

Execution

- Functional Warming
- Detailed Warmup
- Measurement

- the amount you sample is so small, the speed of the timing simulator is inconsequential
- the bottleneck really is in architectural-level simulation
  • to generate the architectural state at the start of the sampled sections
  • to maintain the microarchitcture structures with long transients (L2 cache) in between sampled sections
Summary

- Technology to build a large-scale full-system multicore/multiprocessor simulator
  - Use hybrid transplantation to avoid a full-system construction effort
  - Use interleaved emulation cores to reduce physical system size and complexity