

ISPASS 2007

PTLsim: A Cycle Accurate Full System
x86-64 Microarchitectural Simulator

Matt T. Yourst

`yourst@cs.binghamton.edu`

Thursday, April 26, 2007

Agenda

- Introduction
- Simulation Requirements for the x86-64 Architecture
- PTLsim Design and Implementation
- Experimental Demonstration

Introducing *PTLsim*

PTLsim is a cycle accurate simulator of the **x86-64** instruction set:

- Models a microarchitecture very similar Intel **Pentium 4** and AMD **Athlon 64** chips (both 32-bit and 64-bit)
- Detailed models of an **out of order superscalar processor** pipeline, including issue queues, caches, etc.
- **Extremely Detailed**: RTL level detail of many core structures
- **PTLsim Classic**: single Linux process in **userspace**
- **PTLsim/X**: **full system**, multiprocessor SMP / SMT, device modeling
- **The only uop-level cycle accurate x86 simulator** available to the public as open source
- 250-page **book** available from www.ptlsim.org

x86 Instruction Set

Why use the world's most widely used ISA?

- Many **hardware implementations** (Intel, AMD, Transmeta, Via, etc)
 - Allows direct performance comparisons against commercial microprocessors (we'll demo this)
- **Industry standard compilers** and tools: pre-built software ready to run
- **Native mode co-simulation:** PTLsim runs on same ISA it's simulating:
 - Switch between real x86 host processor and simulation model at any time
 - **Extremely fast:** only use simulated CPU on regions of interest - everything else is native
 - **Self debugging and continuous validation:** can compare with real x86 host chip at any time

x86 Instruction Set

Why use the world's most widely used ISA?

- **The Competition:** traditional academic research tools only support **Alpha** or **MIPS** ISAs:
 - **Discontinued hardware:** co-simulation impossible
 - **Outdated and unmaintained compilers:** many benchmarks will not run anymore
 - **Little commercial relevance:** key benchmarks will not run on these discontinued ISAs

Simulation Requirements for x86 Instruction Set

Challenges and Solutions:

- Cycle accurate simulation of x86 is **fundamentally different than for RISC** architectures
- x86 instructions **decoded into uops** (micro operations), with extensive **microcode** to handle complex instructions
- x86 instructions are **atomic**: all uops in an instruction must commit *atomically* (all at once) or none may commit
- **ALU semantics**: condition codes and size merging
- **Unaligned** load / store fixups in hardware

Simulation Requirements for x86 Instruction Set

Challenges and Solutions:

- **Complex memory model** with hardware TLB refill
- **Self modifying code (SMC)** automatically handled by hardware
- **Interlocked atomic instructions** with complex semantics
- Legacy **x87 Floating Point**
- **Legacy features:** segmentation, 16-bit code, etc.

Micro-Operations (*uops*) Example

Consider the following x86 instruction, in assembly language:

```
addw [%rsp + %r13*8 + 123],%rax
```

This adds register `rax` to the 2-byte (“w” = 16-bit) word at the address formed by adding together register `rsp`, 8 times `r13` and the immediate 123.

The **decode stage** in the PTLsim pipeline breaks this down into **four uops**:

```
adda %t1 = %rsp,%r13*8    Generate address
ldw  %t2 = [%t1 + 123]    Load old value
addw %t2 = %t2,%rax       Do addition
stw  [%t1 + 123],%t2     Store new value
```

The `%t1` and `%t2` registers are temporary (not visible outside the processor core)

PTLsim Core Models

Several plug-in cores:

- **Sequential** (in-order functional modeling)
- **OOO** (simple out of order),
- **SMT** (out of order with simultaneous multithreading)

Out of Order Core

OOO/SMT core microarchitecture very similar to Intel Pentium 4, AMD K8 and Intel Core 2:

- **Reorder Buffer (ROB)** based core
- Supports very complex configurations of functional units, clusters, bypass paths, issue queues and register files
- **Speculative execution** with replays and load/store reordering
- **RTL-level details** for key pipeline structures (issue queues, cache state machine, etc.)

Out of Order Core

OOO/SMT core microarchitecture very similar to Intel Pentium 4, AMD K8 and Intel Core 2:

- **Cache hierarchy** model supports multi-level caches and TLBs, latency and bandwidth modeling and more
- Full **simultaneous multithreading (SMT)** support in conjunction with multi-core configurations
- Cache hierarchy model supports multi-level caches and TLBs, latency and bandwidth modeling and more

Native Mode Co-Simulation

- **Dynamically switch virtual machine** between physical x86 CPUs (Intel, AMD) and PTLsim software-based models
- **Extremely fast:** only use simulated CPU on regions of interest - everything else is native
 - Avoid “warmup periods” required by traditional tools
- **Self debugging with continuous validation:**
 - Compare with real x86 host chip at any time
 - Isolate problems via binary search for divergence point

Xen Intro

- **Xen:** x86 virtual machine monitor (a.k.a. hypervisor)
- Domain 0 runs standard Linux and provides physical device drivers
- **Para-virtualization:** kernel knows about hypervisor and cooperates
 - Xen guest support now in Linux, Solaris, etc.
 - Very high performance (indistinguishable from native)
- **Hardware Virtual Machine:**
 - x86 extensions allow Xen to run unmodified OSs (i.e. Windows)
 - Slower but more compatible

PTLsim and Xen

Advantages:

- Para-virtualized environment easier to work with than legacy x86 (16-bit, etc.)
- Virtual devices can be modified for detailed performance modeling

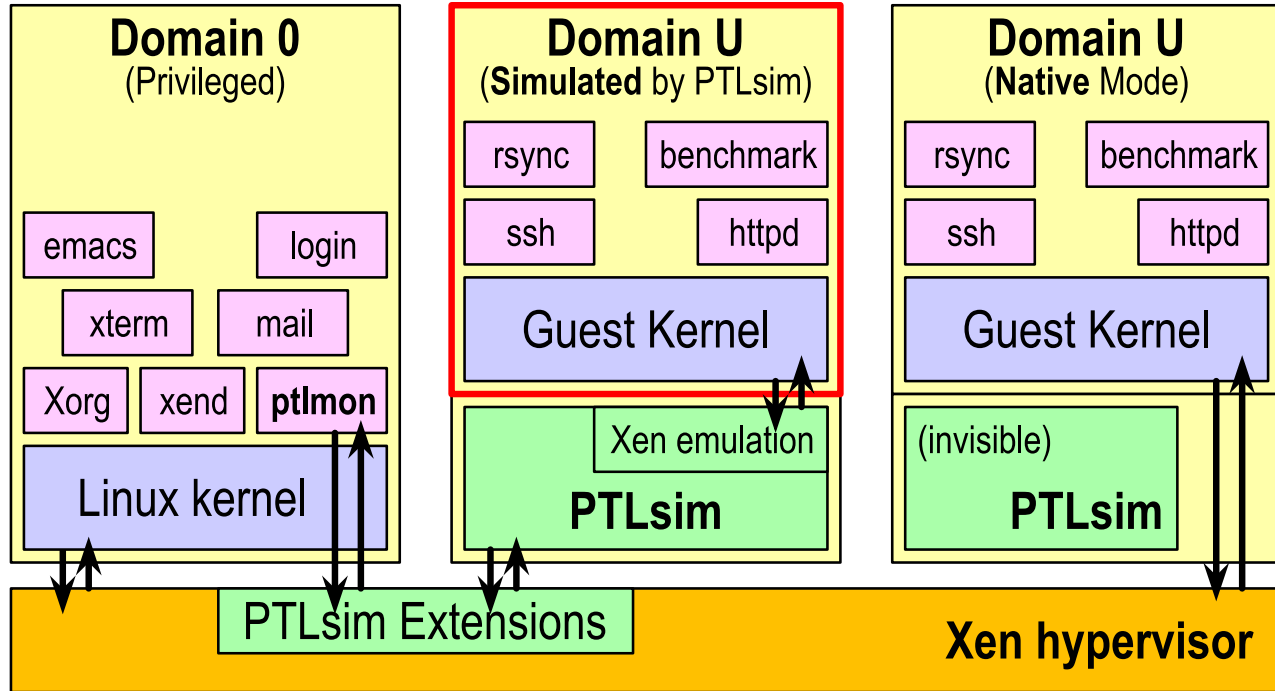
Disadvantages:

- Extra effort to set up and configure virtual machines (disk images, etc)
- Host machine must run Xen hypervisor and special Linux kernel in domain 0
- We are considering a move to KVM in place of Xen

PTLsim and Xen: Architecture

- **Xen hypervisor** modifications:
 - context switching between physical host CPUs and PTLsim virtual CPUs
 - interrupt control, time virtualization, trigger instruction intercepts
- **PTLsim core**:
 - **Runs on the (virtualized) bare hardware** inside the target domain's memory space
 - PTLsim does its own memory management, interrupt handling, etc. (NOT a Linux process!)
 - Isolated from both domain 0 and hypervisor (for easy debugging)
- **PTLsim monitor** process: Linux process in domain 0
 - Connects PTLsim to the outside world (syscall forwarding)

PTLsim and Xen: Architecture



Native Mode and Full System PTLsim

- Special x86 instruction (opcode 0x0f37) inside virtual machine sends commands to PTLsim hypervisor
- Switch between native mode and various simulation cores
- Callable from shell scripts via **ptlctl** program or via ptlcalls.o library
- PTLsim is fully scriptable from both inside and outside the domain

The Nature of Time

- **Simulation causes “time dilation”**: simulator may run at 1 MHz, yet attached (virtual) devices still run at realtime
- Interactions (interrupts, etc) with virtual devices **must be slowed down** to match simulation:
 - Interrupt livelock if too fast
 - TCP timeouts if too slow
- **Implementation**: record stream of interrupts and DMAs to and from devices
 - **Replay this trace**, with event delivery keyed to simulation cycle counter
- PTLsim leverages **Xen checkpoints**

Experimental Evaluation

How Accurate is PTLsim?

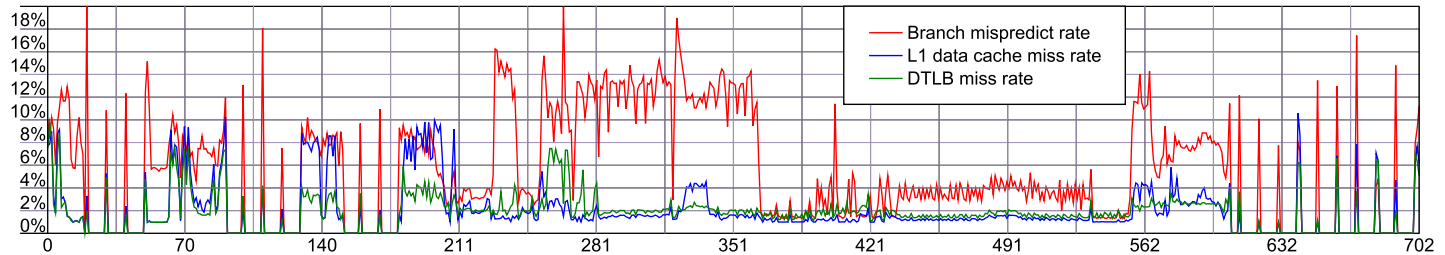
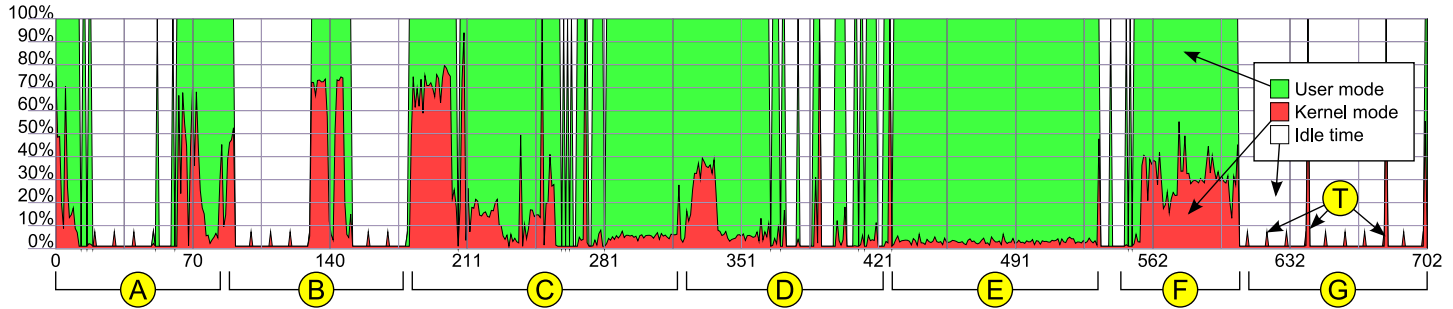
- **Benchmark selected:** `rsync` client/server over SSH and TCP/IP
 - SPEC is irrelevant here: does not stress kernel enough
- **Reference system:** AMD Athlon 64 2.2 GHz (a.k.a. “K8” microarchitecture)
- PTLsim configured with a K8-like model for the core and caches
- **Native run:** record K8 hardware performance counters
- **Simulation run:** record PTLsim performance counters
- **End result:** PTLsim within **5%** of real K8 on all key metrics

Results

Trial	Native K8	PTLsim	%Diff
Cycles	1,482,035K	1,545,810K	+4.30%
x86 Insns Committed	990,360K	1,005,795K	+1.55%
uops	1,097,012K	1,436,979K	+30.99%
L1 D-cache Misses	6,118K	6,564K	+7.28%
L1 D-cache Accesses	414,285K	418,072K	+0.91%
L1 Misses as %	1.48%	1.57%	+0.9%
Total Branches	138,062K	135,857K	-1.60%
Mispredicted Branches	5,727K	5,392K	-5.84%
Mispredicted %	4.15%	3.97%	-0.18%
DTLB Misses	1,593K	3,895K	144%
DTLB Miss Rate %	0.38%	0.93%	245%

Results

PTLsim includes the PTLstats program that automatically generated these graphs:



Conclusion

- Detailed model of a modern out of order x86 processor
- Full system and multi-processor / SMT support in conjunction with Xen
- Highly relevant and accurate for real industrial-grade performance modeling
 - PTLsim's AMD K8 model is within 5% of the real chip
- Available from www.ptlsim.org