Welcome to PTLSim

PTLSim User's Guide Presentation

Prolog Based Transformation Language

HISTORY

PTLsim was designed and developed by Matt T. Yourst with its beginnings dating back to 2001. The main PTLsim code base, including the out of order processor model, has been in active development since 2003 and has been used extensively by the processor design research group at the State University of New York at Binghamton in addition to hundreds of major universities, industry research labs and several well known microprocessor vendors.

PTLsim is not related to other legacy simulators. It is our hope that PTLsim will help microprocessor researchers move to a contemporary and widely used instruction set (x86 and x86-64) with readily available hardware implementations. This will provide a new option for researchers stuck with simulation tools supporting only the Alpha or MIPS based instruction sets, both of which have since been discontinued on real commercially available hardware (making co-simulation impossible) with an uncertain future in up to date compiler toolchains.

INTRODUCING PTLSIM

PTLsim is a state of the art cycle accurate microprocessor simulator and virtual machine for the x86 and x86-64 instruction sets. PTLsim models a modern superscalar out of order x86-64 compatible processor core at a configurable level of detail ranging from full-speed native execution on the host CPU all the way down to RTL level models of all key pipeline structures. PTLsim supports the full x86-64 instruction set of the Pentium 4+, Athlon 64 and similar machines with all extensions (x86-64, SSE/SSE2/SSE3, MMX, x87). It is currently the only tool available to the public to support true cycle accurate modelling of real x86 microarchitectures.

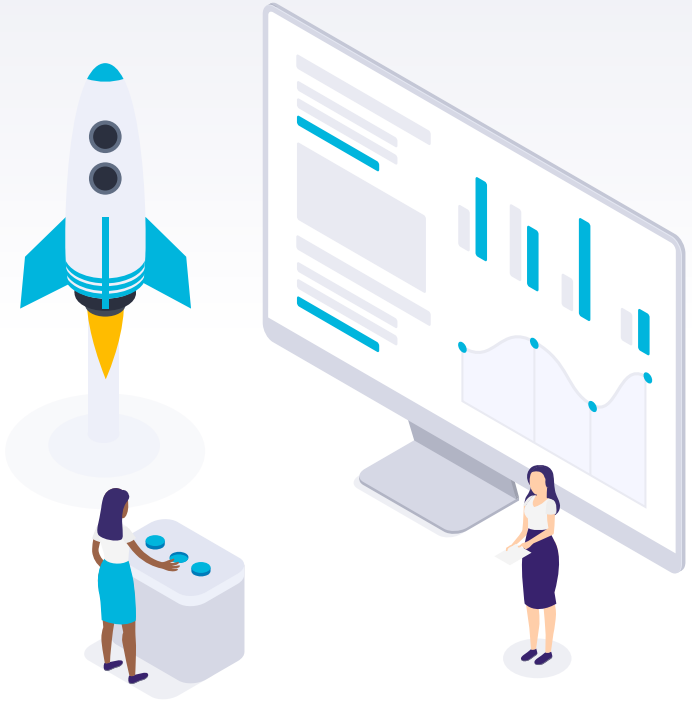
PTLsim is very different from most cycle accurate simulators. Because it runs directly on the same platform it is simulating (an x86 or x86-64 machine, typically running Linux), it is able to switch in and out of full out of order simulation mode and native x86 or x86-64 mode at any time completely transparent to the running user code.

Compared to competing simulators, PTLsim provides extremely high performance even when running in full cycle accurate out of order simulation mode. Through extensive tuning, cache profiling 10 and the use of x86 specific accelerated vector operations and instructions, PTLsim significantly cuts simulation time compared to traditional research simulators. Even with its optimized core, PTLsim still allows a significant amount of flexibility for easy experimentation through the use of optimized C++ template classes and libraries suited to synchronous logic design.

1

PTLsim Architecture

PTLsim Code Base Overview



Designing a cycle accurate model of a modern x86-64 microarchitecture is fundamentally different than building a simulator for a RISC architecture.

The internal µop instruction set used by PTLsim has many key differences from RISC instructions, since it is intended to efficiently support the nuances of x86 instructions while still being implementable in hardware. The uops used by PTLsim are intended to be quite similar to the functionality and encoding of uops in the Intel Pentium 4, Core 2 and AMD K8 processors.

PTLsim is written in **C++** with extensive use of x86 and x86-64 inline assembly code. It must be compiled with gcc on a Linux 2.6 based x86 or x86-64 machine. The C++ variant used by PTLsim is known as Embedded C++. Essentially, we only use the features found in C, but add templates, classes and operator overloading including other C++ features such as hidden side effects in constructors, exception handling, RTTI, multiple inheritance and virtual methods (in most cases).

PTLsim uses its microcode to build stack frames, access interrupt descriptor tables (IDT), switch to kernel mode and redirect the processor to the exception handler entry point.

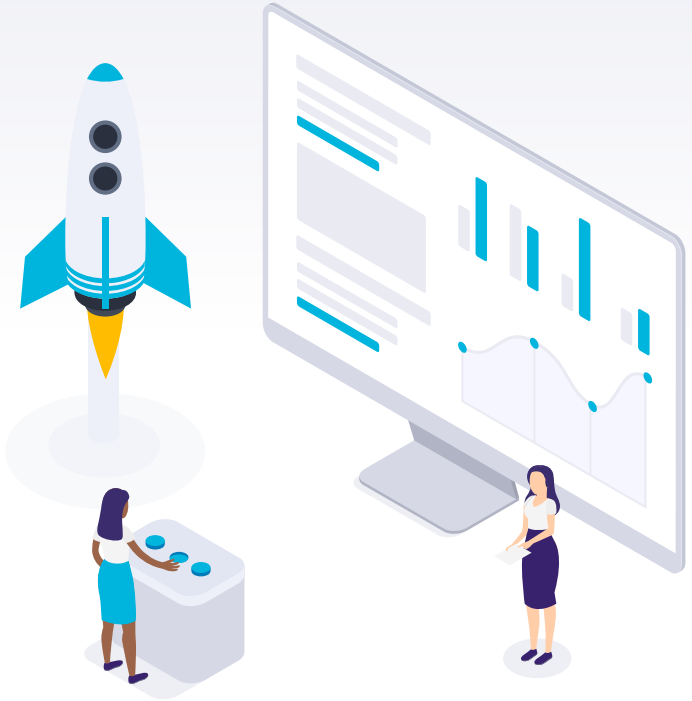
PTLsim must provide virtual hardware models of the real time clock, interrupt controller and other devices, and these models all must be cycle accurate and fully deterministic for debugging purposes.

PTLsim also supports modelling of multi-processor or simultaneous multithreading (SMT) machines.

1

PTLsim Core Model

PTLsim Architecture Overview



PTLsim includes a variety of core models, including the main superscalar out of order core, an SMT (simultaneously multi-threaded) version of that core, and an in-order sequential core used for rapid testing and microcode debugging.

The default core model is a modern superscalar out of order design, based on a combination of features from the Intel Pentium 4, AMD K8 and Intel Core 2, but also incorporates some ideas from IBM Power4/Power5 and Alpha EV8.

Firstly, PTLsim includes a very powerful feature absent from other cycle accurate simulators: it can dynamically switch the target virtual machine (or in userspace PTLsim, the target process) between the host system's real physical CPUs and one of PTLSim's simulated core models (such as the out of order core, the SMT core or the sequential core).

Secondly, native mode effectively makes PTLsim self debugging. It is possible, on an instruction by instruction basis, to determine where the architectural state produced by PTLSim's model begins to diverge from the state produced by the native x86 host processor.

Thirdly, statistical sampled simulation can be used very effectively: for instance, using PTLSim's scripting language, the user can request that the out of order simulation core be used for 100 million instruction spans out of every billion real instructions, with the majority of the time spent in native mode.

Common Libraries and Logic Design APIs

PTLsim includes a number of powerful C++ templates, macros and functions not found anywhere else. Under this section we have:

General Purpose Macros:

The file globals.h contains a wide range of very useful definitions, functions and macros we have accumulated over the years, including :

- Basic data types used throughout PTLsim (e.g. W64 for 64-bit words, Waddr for words the same size as pointers, and so on)
- Type safe C++ template based functions, including min, max, abs, mux, etc.

Super Standard Template Library (SuperSTL):

The Super Standard Template Library (SuperSTL) is an internal C++ library we use internally in lieu of the normal C++ STL for various technical and preferential reasons.

Multithreading and Multiprocessor Support

PTLsim was designed from the ground up to support multiple VCPUs per domain, thereby allowing efficient SMP, multi-core and multi-threaded core models. The active core model decides how to distribute the VCPUs in a domain onto individual cores and threads within cores.

The Context structure in PTLsim is central to multi-processor support. Each VCPU has one Context structure encapsulating all information about that VCPU, including its architectural registers, x86 machine state registers (MSRs), page tables and internal PTLsim state. As the simulator commits instructions from a given hardware thread or core, it updates the architectural state in that core's Context structure. Similarly, the Context structure is available to all microcode functions and other PTLsim subsystems.

PTLSim's multi-core abilities are currently limited in that each core has a dedicated cache hierarchy (L1/L2/L3); the cycle accurate interconnect network and cache coherence logic are up to each user to provide. By default, PTLsim models an "instant visibility" cache coherence model: there are no delays on line movements between cores. However, the infrastructure is in place for MOESI-compatible cache coherence models to be easily plugged into the PTLsim code.

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%

Accuracy of PTLsim on multiple metrics compared to reference silicon (AMD Athlon 64 @ 2.2 GHz). Figures are in thousands.

Getting Started with PTLsim

- PTLsim can be built on both 64-bit x86-64 machines (AMD Athlon 64 / Opteron / Turion, Intel Pentium 4 with EM64T and Intel Core 2) as well as ordinary 32-bit x86 systems. In either case, the system must support SSE2 instructions; all modern CPUs made in the last few years (such as Pentium 4 and Athlon 64) support this, but older CPUs (Pentium III and earlier) specifically do not support PTLsim

- If built for x86-64, PTLsim will run both 64-bit and 32-bit programs automatically. If built on a 32-bit Linux distribution and compiler, PTLsim only supports ordinary x86 programs and will typically be slower than the 64-bit build even on 32-bit user programs.

- PTLsim runs on any recent Linux 2.6 based distribution.

- Steps for downloading PTLSim :

Download PTLsim from the web site (http://www.ptlsim.org/download.php). It is recommend starting with the “stable” version, since this contains all the files you need and can be updated later if desired.

• Unpack ptlsim-2006xxxx-rXXX.tar.gz to create the PTLSim directory. 47

• Run make.

– The Makefile will detect your platform and automatically compile the correct version of PTLsim (32-bit or 64-bit).

Running PTLsim:

PTLsim invocation is very simple: after compiling the simulator and making sure the PTLSim executable is in the path, simply run:

```
ptlsim full-path-to-executable arguments...
```

PTLsim reads configuration options for running various user programs by looking for a configuration file named `/home/username /.ptlsim/path/to/program/executablename.conf`. To set options for each program, you'll need to create a directory of the form /home/username /.ptlsim and make sub-directories under it corresponding to the full path to the program. For example, to configure /bin/ls you'll need to run `"mkdir /home/username /.ptlsim/bin"` and then edit `"/home/username /.ptlsim/bin/ls.conf"` with the appropriate options. For example, try putting the following in ls.conf as described:

```
-logfile ptlsim.log -loglevel 9 -stats ls.stats -stopinsns 10000
```

Then run:

```
ptlsim /bin/ls -la
```

PTLsim should display its system information banner, then the output of simulating the directory listing. With the options above, PTLsim will simulate `/bin/ls` starting at the first x86 instruction in the dynamic linker's entry point, run until 10000 x86 instructions have been committed, and will then switch back to native mode (i.e. the user code will run directly on the real processor) until the program exits. During this time, it will compile an extensive log of the state of every micro-operation executed by the processor and will save it to "ptlsim.log" in the current directory. It will also create "ls.stats", a binary file containing snapshots of PTLSim internal performance counters.

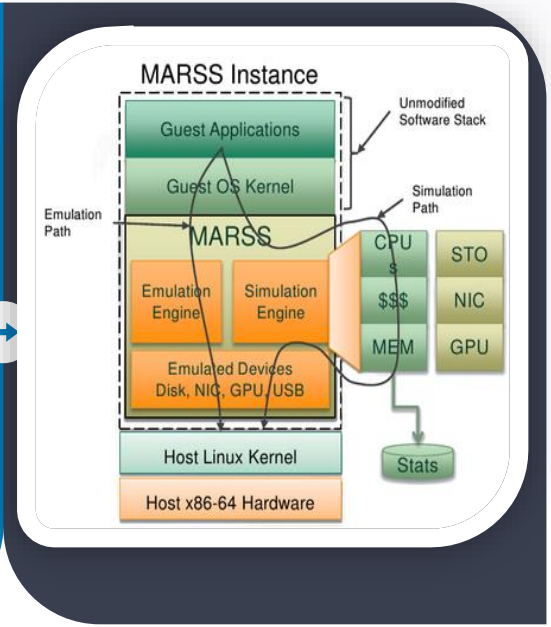
MARSSx86-Micro-ARchitectural and System Simulator for x86-based Systems

- MARSSx86 (MARSS for short) is a tool for cycle accurate full system simulation of the x86-64 architecture, specifically multicore implementations.
- Heterogenous Core Modelling
- Performance models for: Aggressive out of order core design along with multi threaded performance.

+

- Based on PTLSim (older x86 simulator from CAPS Group)
- Components used from PTLSim: Decoder, core components of Out-of-Order Datapath, SuperSTL and Logic libraries
- Fast models for coherent cache, memory system
- Several added optimizations for performance correctness and flexibility.

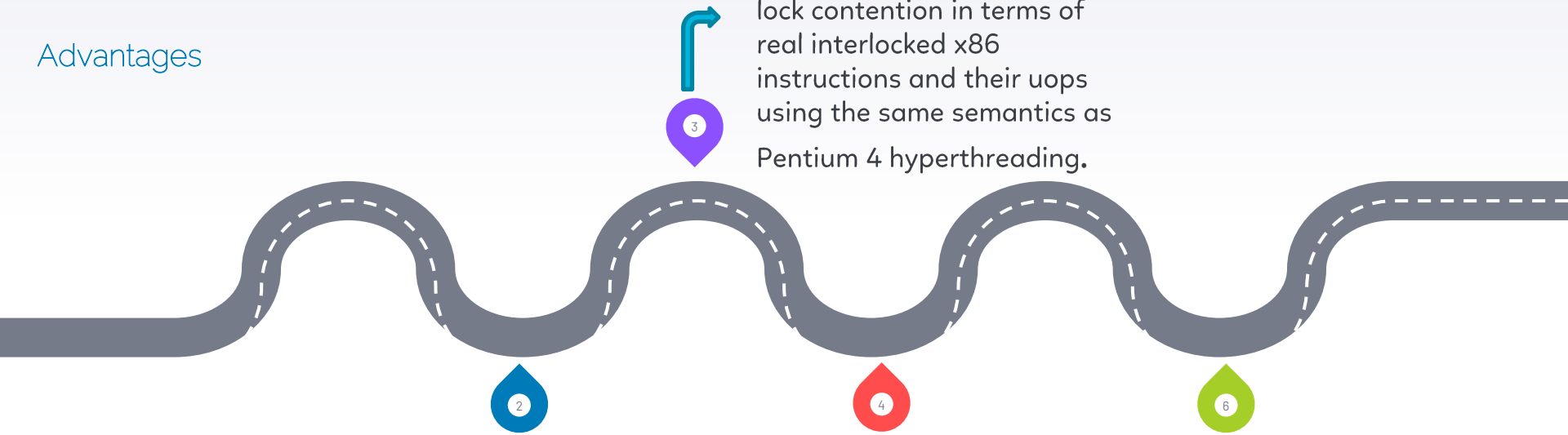
→



General Advantages and Disadvantages

Advantages

PTLsim faithfully models all lock contention in terms of real interlocked x86 instructions and their uops using the same semantics as Pentium 4 hyperthreading.



Disadvantages

The PTLSim typically cannot run true shared memory multithreaded programs involving lock contention, limiting their usefulness to simple single process benchmarks.

It is also unclear how accurate results can be obtained without simulating the kernel's thread scheduler and interrupt handling code.

The use of the simulator is limited till date.

Conclusion

In this presentation, we described how we extended PTLsim into a full system simulator by integrating it with the Xen hypervisor, and how we added multi-processor and multithreading support to PTLsim. We first described what it takes to perform cycle accurate modelling of a complete x86 machine at the uop (micro-operation) level, along with the challenges and requirements for effective full system multi-processor capable simulation. We also disclosed the PTLsim out of order core's features, and we described how native mode co-simulation integrates with PTLSim's core models. We detailed the internal architecture of full system PTLsim and how it interacts with the Xen hypervisor, as well as how we model the various system-level aspects of the x86 architecture. The challenges of accurately modelling the flow of time in a simulation environment were also discussed.

References

Main Design Analysis Paper

M. Yourst. *PTLsim User's Guide and Reference.* Technical report at http://www.ptlsim.org

Resources collected

P. Magnusson et al. *Simics: A Full System Simulation Platform.* IEEE Computer, Feb. 2002 (Vol 35 N 2), p50.

T. Austin et al. *Simple Scalar: An Infrastructure for Computer System Modelling.* IEEE Computer, February 2002

Resources collected

J. Veenstra, R. Fowler. *MINT: a front end for efficient simulation of shared-memory multiprocessors.* Proc. of Modeling, Analysis, and Simulation of Comp and Telecom Systems, 1994, p201.

Resources collected

That Ends My Presentation

Thanks a lot for your kind attention!!!!

Research and Development Done by

PRATTAY PAUL
CSE 2nd year
12200120007