SimTRaX: Simulation Infrastructure for Exploring Thousands of Cores

Konstantin Shkurko
University of Utah
kshkurko@cs.utah.edu

Tim Grant
University of Utah
tgrant@cs.utah.edu

Erik Brunvand
University of Utah
elb@cs.utah.edu

Daniel Kopta
University of Utah
dkopta@cs.utah.edu

Josef Spjut
NVIDIA
josef.spjut@gmail.com

Elena Vasiou
University of Utah
elvasiou@cs.utah.edu

Ian Mallett
University of Utah
imallett@cs.utah.edu

Cem Yuksel
University of Utah
cem@cemyuksel.com

ABSTRACT

SimTRaX is a simulation infrastructure for simultaneous exploration of highly parallel accelerator architectures and how applications map to them. The infrastructure targets both cycle-accurate and functional simulation of architectures with thousands of simple cores that may share expensive computation and memory resources. A modified LLVM backend used to compile C++ programs for the simulated architecture allows the user to create custom instructions that access proposed special-purpose hardware and to debug and profile the applications being executed. The simulator models a full memory hierarchy including registers, local scratchpad RAM, shared caches, external memory channels, and DRAM main memory, leveraging the USIMM DRAM simulator to provide accurate dynamic latencies and power usage. SimTRaX provides a powerful and flexible infrastructure for exploring a class of extremely parallel architectures for parallel applications that are not easily simulated using existing simulators.

CCS CONCEPTS

• **Computing methodologies** → **Simulation tools**; *Graphics processors*; • **Computer systems organization** → *Multiple instruction, multiple data*;

KEYWORDS

Architecture simulation; single program multiple data; LLVM

ACM Reference Format:
Konstantin Shkurko, Tim Grant, Erik Brunvand, Daniel Kopta, Josef Spjut, Elena Vasiou, Ian Mallett, and Cem Yuksel. 2018. SimTRaX: Simulation Infrastructure for Exploring Thousands of Cores. In *Proceedings of 2018 Great Lakes Symposium on VLSI (GLSVLSI '18)*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3194554.3194650

GLSVLSI '18, May 23–25, 2018, Chicago, IL, USA

1 INTRODUCTION

When designing application-specific accelerator architectures, cycle-accurate simulators are indispensable for rapid exploration of the potential design space. Unlike trace-based simulators that can quickly evaluate particular sub-systems under specific work-loads, cycle-accurate simulators capture the precise inner-workings of the entire system. Thus, they provide a testbed for software modifications to make the target application more suitable for the custom accelerator architecture.

This paper describes SimTRaX: a set of tools that resulted from exploring the design space of massively parallel accelerator architectures made up of simple in-order cores combined with shared compute and memory resources. Unlike most other simulators, SimTRaX is designed to simulate thousands of concurrent hardware threads with cycle-accuracy without introducing high-level approximations. SimTRaX provides ample performance to run applications to completion. We support the LLVM compiler [13] to provide simple software access to custom hardware features. We can use source-level debugging symbols provided by LLVM to debug and profile simulated architectures and applications on a wide variety of metrics. Furthermore, accurate simulation of the DRAM behavior produces reliable results especially for applications that process a large amount of data with unpredictable access patterns.

2 BACKGROUND

Although existing simulators enable experimentation within their own expected parameters, we have found that exploring a design space that is more massively parallel can benefit from a new simulation infrastructure. We find that these architectures, and the applications that map well to them, can often be categorized as single-program multiple-data (SPMD) processing. In this case all hardware processors execute the same program, but because they have their own PCs, can be at different points in that program depending on their own data and control flow[1]. This can leverage data

[1]Note that single-instruction multiple-thread (SIMT) processing as defined by NVIDIA [15] is often considered similar to single-program multiple-data (SPMD). However, SIMT support for divergent thread execution only tracks the divergence and still must mask off the results from diverged threads within a SIMT group. SPMD in our model allows threads to make individual progress while diverging.

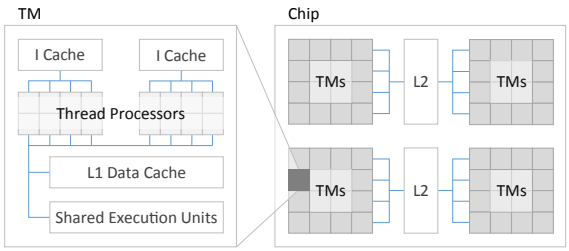


Figure 1: An example of a hierarchically organized parallel architecture [12] supported by SimTRaX.

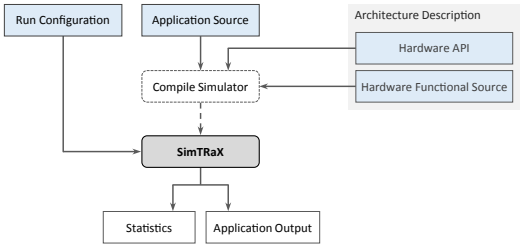


Figure 2: Functional mode work flow. *Application source* and *architecture description* are compiled into SimTRaX executable specific to application. Blue color highlights inputs.

parallelism and simultaneously sharing of large and complex functional units. Examples of such tiled architectures include TRaX [19], Rigel [10], Copernicus [8], STRaTA [11], and SGRT [14].

There are, of course, many existing simulators for exploring new computer architectures. CPU and multi-core simulators such as SimpleScalar [3], Simics [16], and gem5 [2] are designed to simulate up to tens of complex, typically out of order, thread processors with advanced features to accelerate individual thread latency. Simulators for massively parallel architectures like GPUs typically target the single-instruction multiple-data execution model. GPGPU-Sim [1] executes unmodified CUDA and OpenCL workloads on a simulated NVIDIA-like GPU architecture. However, keeping the simulator up to date presents an ongoing challenge.

Tiled architectures generally achieve their performance through massive parallelism via thousands of simple hardware threads. Simulators such as PriME [7], Zsim [18], Graphite [17], and Sniper [4] are focused on increasing simulated parallelism with varying degrees of cycle-accuracy. Rigel [10] is probably the most similar to SimTRaX. Rigel simulates thousands of execution threads, allows configuration of the interconnect and the memory system, and includes development tools based on LLVM. Rigel applications are developed using a simple task-based bulk synchronous parallel programming model to be executed in a SPMD paradigm. While Rigel development seems to have stopped in 2012, it demonstrates that simulation infrastructures that support massive parallelism and compiler integrations are interesting in a variety of domains.

3 SIMULATOR ARCHITECTURE

SimTRaX was designed to help explore application-specific accelerator architectures, modeled as a tiled, hierarchical, parallel architecture with a few thousand processing units. An example is shown in Figure 1. The lowest level is composed of simple in-order Thread Processors (TPs) with limited execution resources. At the middle level, Thread Multiprocessors (TMs) tile a number of TPs alongside units they share access to. At the highest level, a chip can consist of many TMs which share access to chip-wide resources, such as L2 data caches. Such hardware representation offers the ability to simulate a range of architectures, some of which would not map easily to other simulation platforms.

The simulator is designed to keep overhead low when modeling the execution flow for each TP. During each clock, first we simulate all TM-wide units that can enqueue requests into globally-shared units, then global units and finally dynamic random access memory (DRAM).

SimTRaX is accelerated via multiple software simulation threads, each in charge of several TMs. Although access to TM-wide resources requires no simulator synchronization, SimTRaX must serialize access to chip-wide units. Thus simulation times depend in part on the architecture design and the frequency of accesses to globally-shared units. All simulation threads must also synchronize after each simulated cycle. Although relaxing this synchronization requirement increases simulator speed [17], it introduces errors in predicted hardware performance.

4 IMPORTANCE OF ACCURATE DRAM MODELING

The main memory of a simulated architecture typically consists of DRAM. It must be modeled accurately because it can be a primary consumer of both energy and time in data-bound applications. Due to the internal structure and makeup of DRAM circuits, its operation and performance is far more complex than SRAM. In addition to being dynamic, and thus needing periodic refresh operations, the latency and energy consumption of individual accesses can vary widely based on the patterns and addresses of recent accesses.

When simulating memory-bound applications, particularly with thousands of threads, the accuracy of the memory model can drastically impact the results. SimTRaX relies on a modified version of the Utah Simulated Memory Module (USIMM) simulator [5] to accurately model DRAM behavior. SimTRaX reports detailed memory access statistics to help the user tune and evaluate their algorithmic/architectural innovations.

5 LLVM INTEGRATION

SimTRaX incorporates the LLVM toolchain [13] to facilitate developing applications used to benchmark new hardware units and architecture design. The use of LLVM allows extending the instruction set architecture (ISA) with relative ease. Applications are written in a high-level language, such as C++, and then compiled to the modified ISA. In addition, we also include debugging and profiling features within the simulator.

Each custom hardware unit may be supported by an API to expose it to the application. When compiling as the standalone functional simulator, each API call translates into a function call to

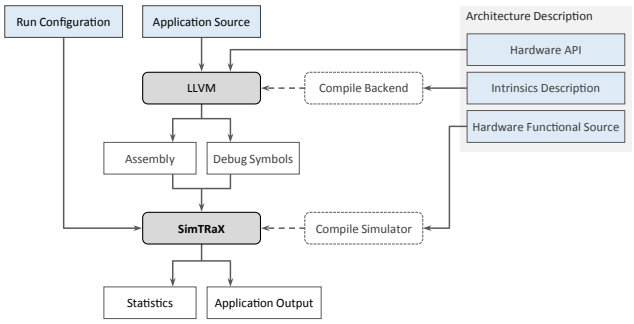
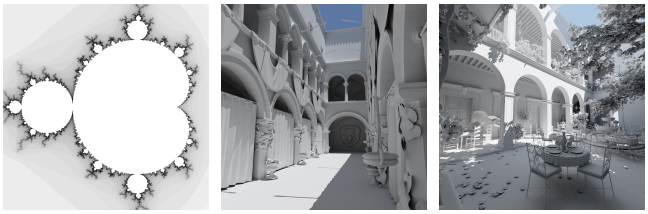


Figure 3: Cycle-accurate mode work flow. The following rely on *architecture description* during compilation: LLVM backend, application assembly which relies on this backend, and SimTRaX executable. Blue color highlights inputs.

```
Time Profile                          FPU Energy Profile
Main                        100.00%   Main                        100.00%
| Shade                      47.57%   | Shade                      51.10%
| | BVH::Intersect           31.16%   | | RandomReflection         15.61%
| | | Tri::Intersect         13.80%   | | | Vec::Normal            11.48%
| | | | Vec::Cross            2.15%   | | | | Vec::Length           7.40%
| | | | Vec::operator-        1.38%   | | | | | sqrt                7.27%
...                                   ...
```

Figure 4: An example profiler output showing the computation time (left) and energy use (right).



(a) Mandelbrot **(b) Path tracing the "Crytek Sponza" scene** **(c) Path tracing the "San Miguel" scene**

Figure 5: Representative output images generated using Mandelbrot and path tracing benchmark applications.

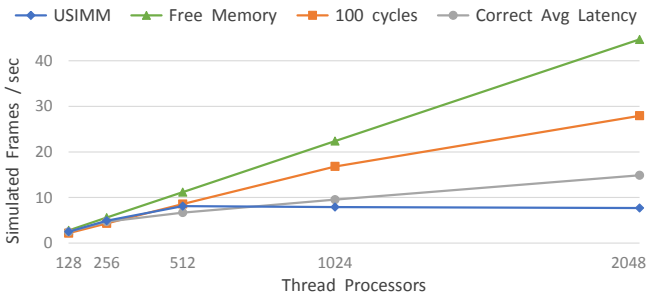


Figure 6: Effect of DRAM accuracy on simulated performance of path tracing on TRaX for San Miguel scene.

the appropriate implementation of the "instruction." Figure 2 shows the work flow for the functional operation.

When the application source is compiled for the cycle-accurate simulator, an assembly file is produced for input into the simulator. Custom instructions exposed by the API calls invoke compiler intrinsics that generate the appropriate assembly instructions. Figure 3 shows the work flow for the cycle-accurate operation.

The LLVM toolchain can embed debug symbols within the assembly of the compiled application. SimTRaX parses and interprets DWARF [6] debugging symbols, and includes a built-in debugger and profiler that operate on the full cycle-accurate state of the simulated machine rather than sampling. Fig. 4 shows an example of the profiler's output with two metrics: a profile for time (left) and the energy spent in floating point arithmetic units (right). The profiler can expose, at the application source code level, the exact source of various behaviors on the chip. As a result, SimTRaX can provide a more complete understanding of the impact aspects of the hardware have on performance when evaluating new architectures.

6 EVALUATION

We evaluate SimTRaX performance when simulating a typical tiled massively parallel architecture targeted at accelerating computer graphics: the TRaX architecture [19]. Selected graphics benchmarks, shown in Figure 5, evaluate the effects of accessing memory on simulator performance. The Mandelbrot benchmark includes a large percentage of computation, where the memory subsystem is accessed only for initial parameters and image output. Path tracing, an algorithm that generates photo-realistic images [9], relies on iterating over tree data structures per pixel and thus can access memory rather incoherently. Smaller scenes, like Crytek Sponza (262,000 triangles), generate more coherent accesses and thus higher cache hit rates than larger scenes like San Miguel (10.5 million triangles).

The output image resolution is 1024×1024, and path tracing benchmark uses maximum ray depth of three. Benchmarked TRaX configuration combines 32-wide TMs into a chip with 128 to 2048 total TPs running at 1GHz. Each TM contains shared execution units and a 32KB L1 data cache. The L1 data cache from each TM is assigned to one of four global 512KB L2 data caches. They connect to the DRAM memory controller set up as 8 channel GDDR5 for a maximum of 512GB/s bandwidth. The performance of the simulated architecture is measured in frames per second (FPS). The simulator is evaluated on a hexa-core Intel i7-5820K CPU running at 3.3GHz and 32GB of DDR4 memory.

6.1 DRAM

To compare the effects of simulating DRAM accurately, consider simulated performance executing path tracing on a TRaX architecture with varying number of TPs, shown in Figure 6. This test includes four different models for DRAM accesses. *USIMM* simulates DRAM behavior correctly. The other modes use a constant access latency: one cycle for *Free Memory*, 100 cycles for *100 cycles*, and 60 – 470 cycles for *Correct Avg Latency* which matches latencies from USIMM simulations. The simulated performance plateaus for the San Miguel scene at 512 TPs when simulating DRAM accurately because the configuration is DRAM bandwidth-limited. However,

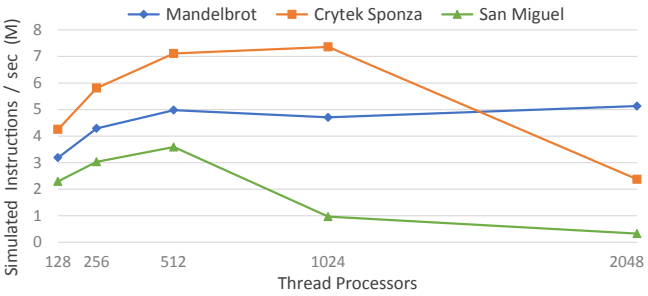


Figure 7: SimTRaX performance measured in millions of simulated instructions per second.

without accurate DRAM simulation, this behavior cannot be accurately captured, and the simulated performance results scale almost linearly with the number of TPs. It is clear that the accuracy of simulating the DRAM dynamic behavior can have a profound effect on reported results of the simulator.

6.2 Simulator Performance

Although cycle-accurate simulators enable highly accurate and detailed evaluation of proposed architectures, they are typically considered slow. However, the combination of architectural choices, using multi-threading and general improvements in processor performance help make cycle-accurate simulations feasible. Figure 7 shows SimTRaX performance measured in millions of simulated instructions per second (MSIPS) as a function of number of TPs while using four simulator threads. SimTRaX simulates at a rate of 0.33 to 7.38 MSIPS depending on how frequently the workload and hardware communicate with chip-wide units. This performance is good enough to enable co-processors with thousands of threads and new hardware units, and benchmark applications running to completion. The performance dips for both Crytek Sponza and San Miguel scenes around 1024 TPs because those configurations become DRAM bandwidth-limited. This results in higher DRAM latencies and more simultaneous requests to the DRAM simulation component, requiring heavy utilization of software locks.

7 CONCLUSION AND FUTURE WORK

We have described SimTRaX, a simulator designed to explore highly parallel co-processor architectures with thousands of simple threads sharing access to expensive computation and memory resources. The simulation infrastructure provides several components essential for quick exploration of possible architecture designs concurrently with targeted software modifications: combined cycle-accurate and functional simulation capability, flexibility in how functional units are connected, a highly accurate DRAM model, and integration of the LLVM toolchain for easy ISA extensions and compiling applications written in a high-level language.

The combination of LLVM debugging information and cycle-accurate system state can be used to generate a highly detailed profile with information beyond just execution time. For example, a user could generate a profile of energy spent in the various chip components during execution. This focus on massively parallel architecture, integration with LLVM, and the resulting ability to profile simulated code in detail on the simulated architecture is not available on other simulator platforms that we are aware of.

An interesting future direction is extending SimTRaX to support more sophisticated memory sharing mechanisms, which would enable SimTRaX to simulate more general architectures. It would also be interesting to explore how we can enable simulations to rely on task-based parallelism while relaxing how often simulation threads need to synchronize. To aid in evaluating radically different architectures faster, SimTRaX's modularity should extend further and enable a node-based graphical interface to configure how functional units are connected and shared. Finally, an automatic system can be used to find optimal architecture configurations by using gradient-descent-like algorithms to optimize a metric produced by the simulator (like energy or performance).

ACKNOWLEDGMENTS

This material is supported in part by the National Science Foundation under Grant No. 1409129. The authors thank Solomon Boulos, Al Davis, Spencer Kellis, Andrew Kensler, Steve Parker, Paymon Saebi, Pete Shirley, and Utah Architecture group for discussions and simulator contributions. Crytek Sponza is from Frank Meinl at Crytek and Marko Dabrovic and San Miguel is from Guillermo Leal Laguno.

REFERENCES

[1] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS*. 163–174.
[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Arch. News* 39, 2 (Aug. 2011).
[3] D. Burger and T. Austin. 1997. *The Simplescalar Toolset, Version 2.0.* Technical Report TR-97-1342. University of Wisconsin-Madison.
[4] T. E. Carlson, W. Heirman, and L. Eeckhout. 2011. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-core Simulation. In *SC*.
[5] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti. 2012. *USIMM: the Utah SImulated Memory Module.* Technical Report UUCS-12-02. Univ. of Utah.
[6] DWARF Debugging Information Format Committee. 2010. DWARF Debugging Information Format V. 4. http://www.dwarfstd.org/doc/DWARF4.pdf. (2010).
[7] Y. Fu and D. Wentzlaff. 2014. PriME: A parallel and distributed simulator for thousand-core chips. In *ISPASS*. IEEE, 116–125.
[8] V. Govindaraju, P. Djeu, K. Sankaralingam, M. Vernon, and W. R. Mark. 2008. Toward A Multicore Architecture for Real-time Ray-tracing. In *IEEE/ACM Micro*.
[9] J. T. Kajiya. 1986. The Rendering Equation. In *Proceedings of SIGGRAPH*. 143–150.
[10] J. H. Kelm, D. R. Johnson, M. R. Johnson, N. C. Crago, W. Tuohy, A. Mahesri, S. S. Lumetta, M. I. Frank, and S. J. Patel. 2009. Rigel: an architecture and scalable programming interface for a 1000-core accelerator. In *ISCA*.
[11] D. Kopta, K. Shkurko, J. Spjut, E. Brunvand, and A. Davis. 2013. An energy and bandwidth efficient ray tracing architecture. In *Proc. HPG*. 121–128.
[12] D. Kopta, K. Shkurko, J. Spjut, E. Brunvand, and A. Davis. 2015. Memory Considerations for Low Energy Ray Tracing. *Comp. Gr. Forum* 34, 1 (2015), 47–59.
[13] C. Lattner. 2008. LLVM and Clang: Next generation compiler technology. In *The BSD Conference*. 1–2.
[14] W. Lee, Y. Shin, J. Lee, J. Kim, J. Nah, S. Jung, S. Lee, H. Park, and T. Han. 2013. SGRT: A mobile GPU architecture for real-time ray tracing. In *Proc. HPG*.
[15] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. 2008. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *Micro, IEEE* 28, 2 (2008), 39–55.
[16] P. S Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. 2002. Simics: A full system simulation platform. *Computer* 35, 2 (2002), 50–58.
[17] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. 2010. Graphite: A distributed parallel simulator for multicores. In *HPCA*. IEEE, 1–12.
[18] D. Sanchez and C. Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems. In *ISCA*.
[19] J. Spjut, A. Kensler, D. Kopta, and E. Brunvand. 2009. TRaX: A Multicore Hardware Architecture for Real-Time Ray Tracing. *IEEE Trans on CAD* 28, 12 (2009).