Simulation of Exascale Nodes through Runtime Hardware Monitoring

Joseph L. Greathouse, Alexander Lyashevsky, Mitesh Meswani, Nuwan Jayasena, Michael Ignatowski
AMD Research

The node-level architecture of exascale systems may look significantly different than current designs. Heterogeneous cores [13] will be severely power- and heat-constrained [6]. They may have large amounts of 2.5D [11] or 3D [17] stacked DRAM and may have even larger non-volatile storage [5]. Simulating the large design space of these systems is important in order to find the best candidates for further study. Current simulation techniques offer good visibility into low-level details but are too slow and memory intensive for such explorations.

We propose a simulation model that is based on the idea that reducing insight into some low-level details can greatly increase the simulator's performance. This allows both the exploration of a broader design space as well as the analysis of larger and more representative workloads. Figure 1 presents our simulation methodology, which utilizes existing hardware, along with fast performance and power estimation models, to simulate HPC benchmarks at nearly full speed. This can yield better insights into exascale node designs by allowing faster turnaround of tests at different design points and more workloads that better represent exascale applications.

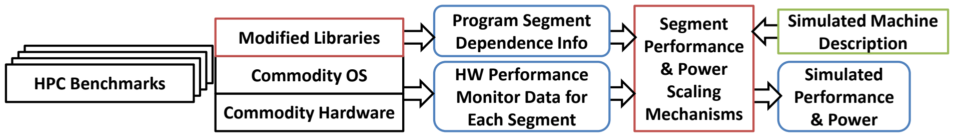
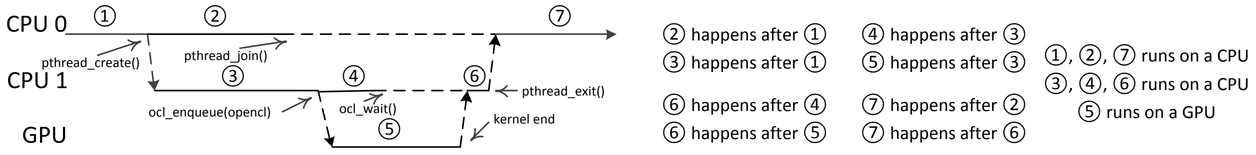

HPC Benchmarks
Modified Libraries
Commodity OS
Commodity Hardware
Program Segment Dependence Info
HW Performance Monitor Data for Each Segment
Segment Performance & Power Scaling Mechanisms
Simulated Machine Description
Simulated Performance & Power

Figure 1. Simulation Overview. Monitoring data and ordering relationships are gathered for dynamic program segments. An offline tool then estimates the runtime and power of each segment for a simulated machine and calculates the total runtime.

We propose a trace-based simulator that gathers performance and ordering information while running the program-under-test on commodity hardware. These traces are saved for later use in an offline power and performance estimation tool. This tool uses the performance information, ordering constraints, and a description of the simulated machine to estimate how the program would run on the simulated hardware.

There are two types of traces saved when executing the program. Each offers different information about program segments that are divided at, for instance, library calls or changes in program phase [22]. The first is a performance data trace that includes a selection of hardware counters, memory traces, and other values that summarize a segment's execution and that will later be used to estimate how that segment would run on a different hardware design. The second is a segment dependence trace that contains a collection of transitive relationships between segments that allow the offline analysis tool to ascertain which segments are able to execute at any point in time.



(a) Example Execution Trace (b) Segment Dependence Information

Figure 2. Example Execution Trace. (a) shows an example execution where the numbered segments are delimited by calls to parallel library functions. (b) lists the segment dependence information that is calculated from this example.

An example of these traces is shown in Figure 2. Figure 2(a) shows an execution where the numbered segments are delimited by function calls that communicate between parallel tasks. The ordering information, shown in Figure 2(b), describes the segment dependence trace for this small program. The hardware performance data may be information such as "Segment 1 had 2 million instructions and 200k cache misses."

As shown in Figure 1, this information is gathered as the program executes by making modifications to the benchmarks' shared libraries. For the example shown in Figure 2, this would involve intercepting calls to Pthreads and OpenCL functions that demarcate the segments [20]. Each of these functions would then save out performance and dependence information while also performing their intended tasks.

Finally, these traces are sent through an offline estimation mechanism that, in conjunction with a description of the machine being simulated, models the runtime and power of each segment. Segments that do not depend upon one another can be co-scheduled if there is enough available hardware on the simulated system, while other segments must be serialized with one another. The dependence information is useful at this point to ensure correctness when: 1) the simulated machine is a different size than the machine used to generate the traces and 2) segment performance scaling would otherwise cause segments to start before all of their predecessors finished.

The methods used to perform these power and performance estimations are active areas of research. They depend on the property being estimated, the desired level of detail, and the hardware being modeled. For example, rough estimates of CPU performance under frequency and memory scaling [7][18][21] require different techniques than GPU performance estimates [23][24]. Power estimation also requires different values and formulae [9][10]. More accurate estimates will require more information in the performance trace and will increase the runtime of the offline analysis. As an extreme, a user that is particularly interested in accurately estimating a particular detail of a segment could go so far as to send it through a conventional low-level simulator.

Related Work: Node-level performance modeling has been an active area of research for decades, as simulators became necessary to guide architectural and microarchitectural design decisions. Tools such as gem5 [2], multi2sim [25], MARSSx86 [19], and GPGPU-Sim [1] model the systems at a relatively low level in order to accurately gauge performance changes from microarchitectural modifications. Power models such as Wattch [3], GPU-Wattch [15], and McPAT [16] also require low-level architectural details or cycle-by-cycle activity information.

In contrast, the simulation infrastructure presented in this paper can rely on higher-level statistics in order to greatly increase performance. Rather than requiring microarchitectural details, we rely on higher-level estimation mechanisms (though the low-level simulators can also be used as accurate estimation tools).

Sniper [4] simulates hardware at a higher-level through interval analysis, an analytical model that primarily pays attention to events that cause pipeline bubbles [8]. Other systems even create whole-system analytical models that can be designed based on regression tests [14] or other automated tests [12]. These systems present interesting performance and power estimation mechanisms. We hypothesize that performance monitoring values gathered on real hardware can form good inputs to the analytical models without requiring the overhead of detailed simulation, and that the additional accuracy of executing longer runtimes with more realistic data sets.

Challenges addressed: The DOE co-design centers have put significant effort into scaling down a selection of large, long-running HPC applications for study on modern nodes. Nonetheless, these proxy applications are still too computationally and memory intensive to run in many traditional low-level simulators. Large simulation runtimes can severely hamper node design space explorations, and the memory usage of low-level simulators will limit what sizes of problems can even be run. Unfortunately, reducing the benchmarks' sizes loses applications fidelity even as it enables more accurate hardware simulation [26]. A reduced input that is not carefully designed may not stress the hardware in the same way as a larger input (e.g. the FLOP/byte ratios may change), while truncated simulations rarely test the majority of an application. Even if this scaling were done well, however, many hardware problems addressed by exascale research require long runtimes. Thermal-aware computation, for instance, requires seconds or minutes of application runtime to appropriately analyze changes in hardware temperature. This high-level simulator trades off low-level hardware visibility in order to address these issues.

Maturity and Effort: AMD has developed an initial version of this simulator and used it to study several novel hardware designs as part of the FastForward effort. The work required to make incremental improvements (e.g. better performance estimation models, support for additional libraries when gathering ordering constraints, etc.) is relatively self-contained, and is part of AMD's continued efforts into exascale computing research. The largest effort often goes into gathering new hardware performance statistics and building accurate models from them.

Applicability: A high-level simulator is applicable not just to high-performance computing, but to any hardware design space exploration that need not necessarily model precise microarchitectural changes. If an estimator can be built to model the hardware changes, this system can allow a fast and easy way to evaluate their worth. This could be useful in areas such as: big-core/little-core heterogeneous designs, CMP energy optimization studies, data center power optimizations, and others. In addition, the performance and power estimation mechanisms that are a vital component of this system are useful in their own right (e.g. [21]).

Uniqueness: The underlying approach for this simulator is useful outside of exascale node analysis, but the exascale design effort has a unique need for the ability to analyze large hardware designs space on long-running and memory-intensive applications. Essentially, other area of computing may benefit from fast simulation, but exascale node design requires it.

Novelty: As discussed above, this simulation infrastructure is related to other high-level simulation systems. Nonetheless, the underlying idea of trace-based analysis with a generic (and changeable) performance and power estimation framework that are fed by hardware performance monitors is unique amongst existing tools.

References

[1] A. Bakhoda, G. Yuan, W. W. L. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in Int'l Symp. on Performance Analysis of Systems and Software (ISPASS), Apr 2009.

[2] N Binkert, et al. "The gem5 Simulator," in the ACM SIGARCH Computer Architecture News, 39(2):1-7, May 2011.

[3] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in Int'l. Symp. on Computer Architecture (ISCA), Jun 2000.

[4] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstarction for Scalable and Accurate Parallel Multi-Core Simulation," in Conf. on High Performance Computing Networking, Storage and Analysis (SC), Nov 2011.

[5] A. M. Caulfield, A. De, J. Coburn, T. I. Mollov, R. K. Gupta, and S. Swanson, "Moneta: A High-performance Storage Array Architecture for Next-generation, Non-volatile Memories," in Int'l Symp. on Microarchitecture (MICRO), Dec. 2010.

[6] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in Int'l Symp. on Computer Architecture (ISCA), Jun 2011.

[7] S. Eyerman and L. Eeckhout, "A Counter Architecture for Online DVFS Profitability Estimation," IEEE Transactions on Computers (ToC) 59(11):1576-1583, 2010.

[8] D. Genbruggeg, S. Eyerman, and L. Eeckhout. "Interval Simulation: Raising the Level of Abstraction in Architectural Simulation," In Int'l Symp. on High Performance Computer Architecture (HPCA), Feb 2010.

[9] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model," in Int'l. Symp. on Computer Architecture (ISCA), Jun 2010.

[10] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," in Int'l. Symp. on Microarchitecture (MICRO), Dec 2003.

[11] M. Jackson, "A Silicon Inteproser-based 2.5D-IC Design Flow, Going 3D by Evolution Rather than by Revolution" Synopsys White Paper, 2012

[12] A. Kerr, E. Anger, G. Hendry, and S. Yalamanchili, "Eiger: A Framework for the Automated Synthesis of Statistical Performance Models", 1st Workshop on Performance Engineering and Applications (WPEA), Dec. 2012.

[13] G. Kyriazis, "Heterogeneous System Architecture: A Technical Review," AMD White Paper, 2012.

[14] B. C. Lee, J. Collins, H. Wang, and D. Brooks, "CPR: Composable Performance Regression for Scalable Multiprocessor Models," in Int'l Symp. on Microarchitecture (MICRO), Nov 2008.

[15] J. Leng, et al. "GPUWattch: Enabling Energy Optimizations in GPGPUs," in Int'l. Symp. on Computer Architecture (ISCA), Jun 2013.

[16] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", in Int'l Symp. on Microarchitecture (MICRO), Dec. 2009.

[17] G. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," in Int'l Symp. on Computer Architecture (ISCA), Jun 2008.

[18] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, "Predicting Performance Impact of DVFS for Realistic Memory Systems," in Int'l Symp. on Microarchitecture (MICRO), Nov 2012.

[19] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A Full System Simulator for Multicore x86 CPUs," in Design Automation Conference (DAC), Jun 2011.

[20] K. Pulo, "Fun with LD_PRELOAD," presented at linux.conf.at, Jan 2009.

[21] B. Rountree, D. K. Lowenthal, M. Schulz, and B. R. de Supinski, "Practical Performance Prediction under Dynamic Voltage Frequency Scaling," in Int'l Green Computing Conference (IGCC), Jul 2011.

[22] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Oct 2002.

[23] J. Sim, A. Dasgupta, H. Kim, R. Vuduc, "A Performance Analysis Framework for Identifying Potential Benefits in GPGPU Applications," in Symp. on Principles and Practice of Parallel Programming (PPOPP), Feb 2012.

[24] S. Song, C. Su, B. Rountree, and K. Cameron., "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures," in Int'l Parallel & Distributed Processing Symp. (IPDPS), May 2013.

[25] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), Sep 2012.

[26] J. J. Yi, S. V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins, "Characterizing and Comparing Prevailing Simulation Techniques," in Int'l. Symp on High Performance Computer Architecture (HPCA), Feb 2005.

Corresponding Author's Contact Information: Joseph.Greathouse@amd.com