

HySIM: Towards a Scalable, Accurate and Fast Simulator for Manycore Processors

Kramer Straube*
kkstraube@ucdavis.edu

Huan Zhang*
ecezhang@ucdavis.edu

Christopher Nitta†
cjnitta@ucdavis.edu

Matthew Farrens†
farrens@cs.ucdavis.edu

Venkatesh Akella*
akella@ucdavis.edu

ABSTRACT

Simulation is the primary means to explore the design space of computer architecture. As the number of cores on a die increases, and the use of heterogeneous cores and on-chip networks leads to more and more complex systems, fast and cycle accurate simulation presents a formidable challenge. In this paper we propose the rationale for and design of a hybrid simulator called HySIM that takes RAMP Gold and moves the timing model of the memory subsystem to the host. This has four important advantages - first, it makes it possible to use multiple FPGAs in order to scale to a large number of cores, including heterogeneous cores and processors with hardware accelerators; second, it enables the modeling of detailed cache-coherence protocols by interfacing the simulator to a memory model such as Ruby; third, it provides a cycle-accurate model for the on-chip network that is flexible enough to support different topologies, routing schemes, and router micro-architectures; and fourth, it frees up resources on the FPGA to increase the number of physical cores or to incorporate an on-chip L1 cache. We present preliminary results to validate HySIM and describe ongoing effort to improve its performance.

1. INTRODUCTION

There is a general consensus that core level parallelism offers a straightforward way to improve performance as technology continues to scale. This trend started with processors used in servers, but now even processors used in notebooks and emerging embedded systems like tablets and smartphones use multicore processors. For example, the Snapdragon 800 processor used today in phones from Google, Samsung and others has 4 cores, with each core running at 2.3GHz. Tilera, meanwhile, has recently announced a 72-core TILE-Gx72 processor for use in high-performance computing. It is quite conceivable that in the near 'future systems will have well over a hundred cores on a die.

A key challenge that confronts designers is a fast and accurate way to simulate such chips, so that the design space can be explored thoroughly to find the optimal configura-

tion for the number of cores, caches, coherence protocols, the network topology etc. Pure software simulation can be used, but the more accurate the simulation is the slower it is going to run (on the order of hundreds of thousands of instructions per second), which means it can take months to do a cycle-level accurate simulation of a reasonable application. Starting with Quickturn's RPM in 1989 (which was used to emulate the Pentium microarchitecture in 1991) and Virtual Machine Works¹, FPGAs have been used to accelerate simulation, though primarily for ASIC verification. With the advent of the BEE (Berkeley Emulation Engine) [4] and its successor RAMP, FPGAs have become an important tool in the computer architect's toolkit. Over the past few years there have been numerous efforts [5, 12, 11, 1, 7] to strike the right balance between accuracy and simulator performance in FPGA-based processor simulation frameworks.

The main idea in many of these efforts is *time-multiplexing*, which involves mapping the simulated cores to actual physical cores implemented on an FPGA. However, scaling these approaches to be able to simulate hundreds of cores with an acceptable simulation time will require the use of multiple FPGAs, as well as ensuring the resources on a given FPGA are utilized wisely. This involves making some trade-offs - for example, in the RAMP Gold [14] approach the interconnection network is fixed, and the timing of the network and the cache-coherency protocols is not modeled accurately. Furthermore, it is difficult (if not impossible) to extend the RAMP Gold approach to work across multiple FPGAs, because the memory timing model is implemented on the FPGA. If multiple FPGAs are going to be used, there must be a way for these timing models to communicate with each other, which is difficult when the models are distributed and designed to only deal with on-chip behavior.

In this paper we propose HySIM, which can be viewed as a redesign (or an extension) of RAMP Gold. The key idea in HySIM is to pose the problem of designing a simulator as a hardware/software codesign of a full-system software simulator (such as GEM5). The goal is to find the right partition between hardware (that runs on the FPGA) and the rest of the infrastructure. In our initial work reported in this paper we propose a partition at the Ruby level (using the GEM5 terminology), meaning everything from the memory timing model on up is in software on the host. There are several advantages to such a partitioning. First, it frees up FPGA resources which can be used to implement more physical cores

*Department of Electrical & Computer Engineering
University of California, Davis

†Department of Computer Science
University of California, Davis

¹Logic Emulation and Prototyping, Mike Butts, RAMP at Stanford, August 2010

and/or elements of the memory hierarchy. Second, by moving the memory timing model to software, we automatically get the ability to model complex coherence protocols. Third, this approach allows us to use multiple FPGAs - since the memory timing model is a centralized structure, handling that in software means no shared information is kept localized within any FPGAs. We are limited by the bandwidth between the host and the FPGA using this approach; however, with the advent of high bandwidth interconnects such as Infiniband and PCI express, this is not likely to be an insurmountable problem. In addition, the ability to spread the simulation across multiple FPGAs means that we can not only simulate more cores, but we can also now simulate heterogeneous processors [6, 8] and processors with hardware accelerators like the Snapdragon. Finally, partitioning in this manner allows us to do detailed and cycle-accurate simulation studies of interconnection networks, including routing protocols, router microarchitectures, and topologies.

In this paper we describe our current efforts in designing and validating HySIM. This is a work in progress, so we will present only preliminary results of the viability of the approach. The rest of the paper is organized as follows. We start with a survey of related work highlighting the key advantages and disadvantages of the various approaches to accelerating processor simulation. We then present the challenges we faced when trying to get RAMP Gold up and running in our laboratory. Next we describe our approach in more detail, including results from initial experiments, and conclude with ongoing work.

2. RELATED WORK

Several simulators exist for multiprocessor target systems. They fall into two main classes: software-based and FPGA-based.

2.1 Software-based Simulators

Software-based simulators use a host computer to perform all of the simulation steps of the multiprocessor system model. They are capable of being highly accurate, but because they must emulate all the hardware in software, the amount of work that the host processor needs to complete is enormous.

GEM5 [2], which is based on a combination of GEMS [9] from the University of Wisconsin, Madison and M5 [3] from the University of Michigan, performs cycle-accurate simulations of multi-core systems. The models are implemented to allow both precision and flexibility. One major drawback with GEM5 is that the complexity of the models significantly affects the simulation time and memory requirements. As a result, GEM5 only supports up to 64 cores, and the simulations of certain benchmarks can take on the order of months. It is widely accepted that GEM5 is a very accurate simulator, but it is too slow for many practical uses. Due to these limitations, the design scope of the proposed system is significantly limited.

Graphite [10] from MIT sought to address the slow simulation time of GEM5 by using multiple cores to increase the computation power available to model the system. To accomplish this, the modeling of the cores in the system is accomplished by using separate processing threads running on the host system. Graphite intentionally gives up the ability to do cycle-accurate simulation in order to allow this multi-threaded approach to succeed. However, cycle-accurate sim-

ulation is necessary for proper modeling of certain shared resources, such as the on-chip network and shared memory systems. Thus, the types of investigations Graphite can be used for is limited.

ZSim [13] from MIT and Stanford seeks to take advantage of the multi-threaded approach to doing software-based simulation by using various techniques to ameliorate the accuracy penalties. ZSim generates traces of the memory hierarchy and of the basic blocks for each thread, in what it calls the *bound* phase. This phase properly communicates between the threads to maintain cache coherence in the memory model. After the *bound* phase, ZSim uses these traces in parallel among different domains to get parallelized simulation in the *weave* phase. While this implementation is more accurate than Graphite, it still suffers from significant accuracy penalties by not being strictly cycle-accurate. For example, if all of the threads need to access a single location (as is done for a barrier synchronization), then the trace-based approach will not properly indicate the queuing delay of the burst of requests.

2.2 FPGA-based Simulators

FPGA-based simulators use an FPGA to emulate one or more of the cores of the design, and these emulated cores are time-multiplexed in order to provide the desired number of cores. FPGA-based simulators are able to emulate the cores much faster than software-based simulators can, without loss in accuracy. However, one challenge for FPGA-based simulators is how to properly model the network. In addition, any modifications to the simulator to create new configurations or alter certain aspects of the simulator require significant implementation time to complete.

RAMP Gold [14] from UC Berkeley accomplishes fast and accurate simulation by time-multiplexing a single SPARC core, and can support a maximum number of 64 simulated cores at a time. It avoids the network modeling issue by using a bus based shared memory model. RAMP Gold can achieve a high simulation speed and provide cycle-accurate results, but limiting the network to a shared bus is overly restrictive. Real bus-based parallel systems do not scale past approximately 10 cores because of increased bus contention, and many-core machines will require a more complex on-chip network that supports a cache coherency protocol. Also, the on-chip timing model used in RAMP Gold to model the delays due to memory accesses is simplistic, and does not account for additional delays that will be incurred due to cache invalidates or on-chip network congestion.

HAsim [12] from MIT and Intel is another approach to FPGA-based simulation which uses a detailed model of the processing cores. HAsim also time-multiplexes a single emulated core to support up to 16 modeled cores at a time. To model the network, HAsim uses queues that cycle back to the time-multiplexed core. This is an improvement over RAMP Gold, but cannot truly model all possible networks and thus limits the expressiveness of the network model. Unfortunately, when using an inaccurate model, the data traffic of the core is also inaccurate, which can lead to significant execution time inaccuracies. HAsim can properly model networks close to a mesh, but more complicated networks (such as fat trees) cannot be modeled.

3. PREREQUISITE WORK

HySIM is based on the RAMP Gold simulator architec-

ture, so the first step to creating HySIM was to get the RAMP Gold simulator running. We did not expect this to take much time or be very difficult - unfortunately, we were sadly mistaken. Getting RAMP Gold running proved to be an enormous task, requiring multiple code fixes before RAMP Gold would function properly. Many of these issues could be traced back to the non-standard Akaros operating system that is used in RAMP Gold.

For example, the latest version of Akaros no longer supports the RAMP Gold architecture. This is not immediately obvious, because the RAMP Gold website points to the Akaros website without any mention of a specific version requirement. Using some date and version investigation, we finally tracked down the correct version of Akaros. Unfortunately, this version does not benefit from any improvements or refinements to Akaros, because Akaros support for SPARC (which is the processor that RAMP Gold emulates) was dropped in favor of RISC-V on May 9, 2011. This means that many of the improvements and bug fixes are not included in the version required for RAMP Gold.

Once the correct operating system was obtained, the issue of getting meaningful benchmarks running on the system still remained. As we attempted to run different benchmarks on the system, we discovered that Akaros lacked proper Pthread library support. The Pthread library is a key resource for running many multi-threaded applications. Basic Pthread support in Akaros exists, but it was incomplete for the SPARC target system. This resulted in even simple test programs failing with related Pthread errors. One of the main errors was related to the thread scheduler not properly detecting when all of the cores were idle before returning to the shell. We modified the `__smp_idle()` function to periodically check whether there were threads running on other cores before returning to the management shell, and only returned to the shell when all cores were idle for a specified duration. This is a clear workaround instead of a permanent fix, but due to the complicated nature of the operating system thread scheduler, we could not provide a permanent fix.

Another key issue with Akaros was a bug in the filesystem code. This bug caused a memory access exception during execution, but not during FPGA simulation. We eventually tracked this problem down to a bug (an uninitialized variable `k_i_info->init_size`) in the file system code. This bug does not occur in the functional simulator because the memory is initialized to zero in the simulator, but does on the FPGA hardware, where the memory contains random values that may not be properly initialized. By properly initializing the `k_i_info->init_size` variable, the system successfully accessed memory.

Eventually, however, we were able to run several PARSEC benchmarks on the RAMP Gold infrastructure, so we are now able to move on to the creation of HySIM.

4. DESIGN

The goal of HySIM is to place only the processor on the FPGA, and have all other functions handled by the host. To accomplish this, we started with RAMP Gold and replaced the timing model with an identical timing model on the host machine. By partitioning at the timing model boundary, the new system has the potential to be more scalable and more accurate. The improved scalability comes from the freed FPGA resources that can now be allocated toward increas-

ing the core states that are stored on board. The increased accuracy is accomplished by a more detailed timing model that can be realized on the host (which would be difficult or impossible to implement on the FPGA). These advantages are offset by the fact that there will be an increase in simulation time, due to the need to communicate between the FPGA and host on each timing model transaction. The structure of HySIM is shown in Figure 1.

We intend to follow a 4-step plan to create and validate HySIM. First, a modification to the link controller that creates an additional communication channel between the host and FPGA must be verified. After this is done, a simple loopback implementation that transfers the timing model data must be tested. The third step is to modify the host software to emulate the timing model that is implemented on the FPGA in RAMP Gold. The software timing model can be directly compared to the FPGA version of RAMP Gold for several benchmarks as validation that the timing model has been properly implemented on the host. Finally, the simple timing model will be replaced with the Ruby memory simulator to provide accurate cache coherency timing and support for on-chip network models. Thus far we have completed steps 1 and 2, and have made significant progress on the simple timing model implementation (step 3).

The HySIM interface reuses portions of the application server implementation from RAMP Gold. The specific packet structure uses a dedicated RAMP Gold header, and within the ethernet data payload there is a sub-header (to determine the type of packet) and a payload length. The timing model packets use a new sub-header which contains the data sent from the core to the timing model. On the host side in the loopback simulator, the data from the incoming packet is immediately put into an outgoing packet and sent back. The main issue with this implementation was adapting the ethernet interface control code to include the timing model interface. Due to the significant difference between the pre-existing interface and the timing model interface requirements, the modification of the control code was a significant challenge. The final interface ethernet data format is shown in Figure 2.

The application server was originally written to send a request and wait for a reply, since this approach allows the host and FPGA to remain in lock step; thus, it also required significant modification in order to allow it to interact with the new headers and properly interleave the timing model packets with the other packets necessary for proper simulation. One other notable design change was to add some appserver code and FPGA logic to prevent system call polling during a multi-cycle instruction. If these would occur at the same time, the simulator would enter a bad state and would not function properly. The additional logic sends a temporarily delay reply based on whether the current instruction needs to be replayed, which prevents the bad state that we found through testing.

5. RESULTS

The resource utilization of HySIM versus RAMP Gold are shown in Table 1. The table shows that removing the timing model from RAMP Gold and putting it on the host frees up significant FPGA resources. The on-chip timing model was the limiting factor for RAMP Gold scaling, due to its high BRAM usage. By freeing up that space, the BRAM utilization drops from 90% to 25%. This should allow HySIM to

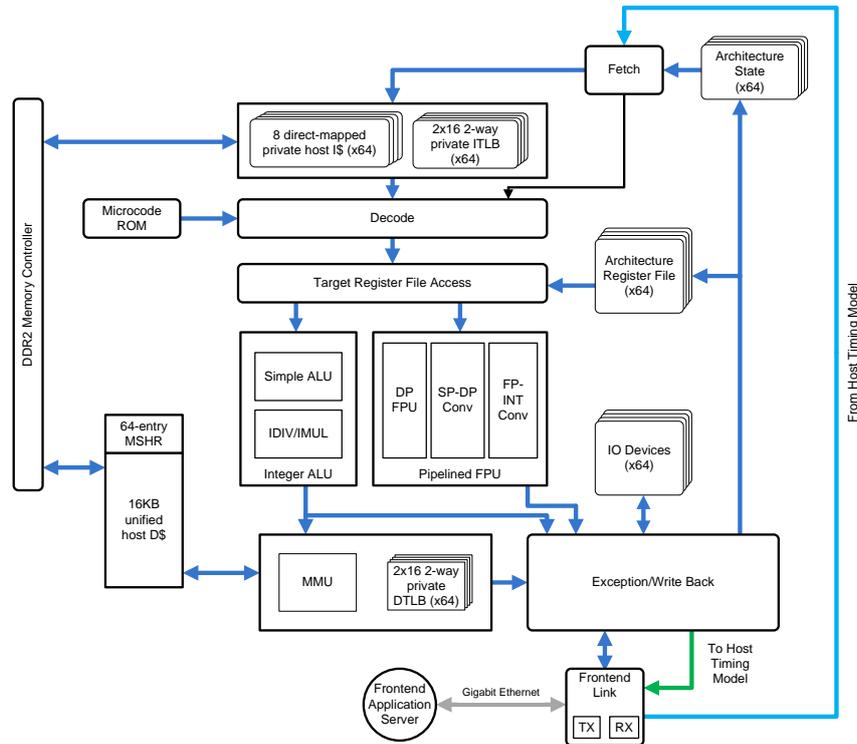


Figure 1: HySIM Structure

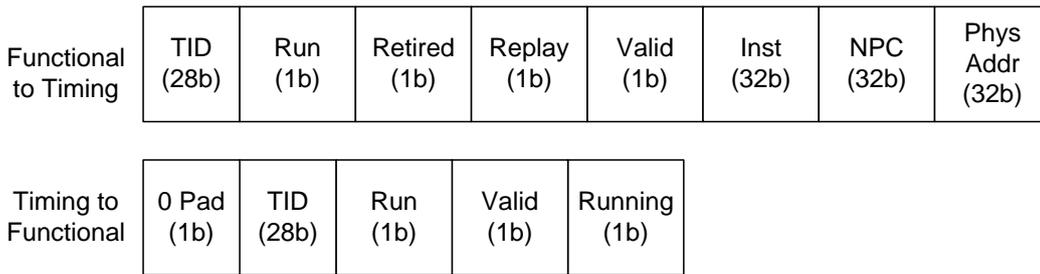


Figure 2: HySIM Ethernet Data Structures

scale linearly to at least 128 target cores. The LUT utilization also drops from 28% to 13%, which frees up space on the FPGA to add logic to improve the core model or implement other features (like the lowest level of the memory hierarchy). The digital signal processing (DSP) block utilization increases slightly, potentially due to a difference in the mapping software that found a less optimal DSP solution than the one used by RAMP Gold. Overall, the reduction of FPGA resource utilization allows for higher scalability and enables new features to be placed on-chip.

The run time of HySIM cannot be accurately measured at this point in the development process. However, a worst-case upper bound on the added simulation time can be calculated and added to the RAMP Gold simulation time. This provides insight into the worst case simulation time that HySIM could take. The overall round-trip latency from the FPGA to the host is 25 μ s. This means that the upper bound on the simulation speed with the current setup is 40 kHz. This number is very low, and we are currently running a variety of experiments to confirm that it is the correct number.

Whether it is correct or not, however, the communication path from the FPGA to the host is likely to be a bottleneck, so in Section 6 we outline several different techniques we intend to explore in order to increase the simulation speed by reducing the probability of incurring the latency penalty.

6. DISCUSSION

HySIM gives up speed for flexibility and scalability. The flexibility of having the timing model on the host allows the use of multiple FPGAs, proper memory system modeling, and reduced FPGA resource utilization. However, the communication overhead incurred by the host slows the overall simulation speed. Fortunately, there are things that can be done to reduce the proportional cost of this overhead, such as aggregating requests for all of the target cores into a single packet. This improvement increases the size of the data payload and reduces the total number of packets transferred, while keeping the header size the same. Thus, the proportion of relevant data to overhead increases, and continues to

Table 1: HySIM vs. RAMP Gold Resource Utilization

Simulator	LUT	Register	BRAM	DSP
HySIM	13%	14%	25%	29%
RAMP Gold	28%	34%	90%	25%

increase as more cores are added (up to the limit of the maximum ethernet data frame). This allows the FPGA to spend less time on communication and more time on simulation.

Our goal is to get the simulation time of HySIM to approach that of RAMP Gold, so that we will have a simulator which provides new functionality (in the form of memory system and network modeling) as well as increased scalability with similar performance. The ability to use multiple FPGAs means the simulated system could have over 1000 cores and still be cycle-accurate. This ability would enable new simulations of systems that have not yet been demonstrated.

HySIM could also get significantly closer to RAMP Gold simulation times by adding an L1 timing model onboard. Using the Intel Core i7 memory hierarchy and a simulator clock speed of 20 MHz, we calculated that adding an L1 cache could reduce the required link bandwidth from 429 MB/s to 45.9 MB/s. This is significant, because it reduces the bandwidth required to below the 1 Gb/s link bandwidth that is available. The instruction cache was assumed to be identical to the data cache for all performance metrics for a worst-case approximation, and the load and store instruction percentages were assumed to be 26% and 10%, respectively. This large reduction in link traffic reduces the time spent waiting for host communication, since most timing model requests would hit in the L1 cache. (This optimization is explained further in Section 7.) Without the L1 optimization, the simulator would still function correctly, but either the clock speed must be reduced or the logic must be throttled to avoid overloading the link. The addition of the L1 allows the FPGA to operate at full speed without requiring excessive bandwidth. Our preliminary tests indicate a round-trip communication latency of 25 μ s for single transactions - the larger data frames that would be used to transfer cache lines, when combined with aggregating multiple core requests, will greatly improve the payload efficiency.

7. FUTURE WORK

The previous sections have outlined the current state of the hybrid simulator HySim. There is much work to be done to improve upon the current implementation. This involves a series of steps that each enhance the current simulator in terms of speed, scalability and accuracy.

As stated previously, HySIM will feature a replacement of the simple timing model with the detailed memory and network simulator combination of Ruby and Garnet. The Ruby and Garnet code from GEM5 can be adapted and integrated into the application server code, which will provide accurate and flexible memory and network models which do not exist in the current timing models. Ruby will properly model the details of the memory hierarchy, including cache coherence, which is critical for accurate simulation of realistic target systems. By using Ruby, different candidate memory hierarchy configurations for the system being simulated can be easily tested and compared.

Garnet is a network simulator that is connected to the back-end of Ruby, and it provides a flexible framework to define any arbitrary network. The ability to define any network accurately is useful for modeling real-world target systems with over 16 cores. Using Ruby and Garnet together allow the architect to study a more “real-world” system by adding cache coherence, a realistic memory hierarchy, and an accurate network. These aspects of the model are especially important when doing cycle-accurate simulation because they can have a significant impact on the execution of the benchmark on the target system.

The current implementation of HySIM suffers from a non-optimal implementation that reduces the overall speed of the simulator. There are several techniques we will be exploring in order to reduce the overall simulation time. As outlined in the previous chapter, the link latency overhead can be reduced by reading and writing the data on the FPGA in larger groups, so that the overhead is proportionally reduced. This allows the FPGA to spend less time working on the link and dealing with latency limitations, and more time simulating. And as cores are added to the target system, the communication overhead will be proportionally reduced even further, allowing for more useful simulation time. For the current 64 core setup, this would speed the upper limit of the target core simulated frequency from 40 kHz to 2.5 MHz. This would continue to scale with more cores, and would bring the HySIM frequency much closer to the RAMP Gold frequency (and would definitely be faster than the GEM5 frequency). We also have discussed placing an L1 cache for each core on the FPGA to service most of the memory requests, to further reduce the simulation time. This prevents the link latency and host processing time from slowing down the majority of the memory operations in the simulated programs. The difficulties with this approach are synchronizing the modeled L1 on the host with the actual L1 on the FPGA for standard memory operations, and ensuring all relevant cache coherence is handled correctly.

We also intend to be able to spread the target system emulation across multiple FPGAs. This requires all of the functional data for the system to be on the host, and the host can connect to multiple FPGAs over different links. The application server will be modified to connect with and identify the different FPGAs through their MAC addresses. Then, traffic can be routed between the FPGAs using the Ruby and Garnet simulated network. If done correctly, this could enable a large number of simulated target cores. Significantly, it would also enable the simulation of heterogeneous cores, because one FPGA could instantiate a core of one type while another could instantiate a radically different one. Perhaps equally significantly, placing the memory on the host system enables the use of a standard operating system, and the use of a non-standard operating system can be avoided. Using Ruby and Garnet means it should be possible to boot and run benchmarks on a standard operating system such as Linux, and based on our experience

with RAMP, this should increase the flexibility, scalability and usability of the simulator.

8. CONCLUSIONS

In this paper we have presented our work on HySIM, which is based on RAMP Gold. HySIM moves the timing model of the memory subsystem on RAMP Gold to the host, which has four important advantages: 1) it makes it possible to employ multiple FPGAs, allowing the simulator to scale to a large number of cores, including heterogeneous cores and cores that use hardware accelerators; 2) it facilitates the modeling of detailed cache-coherence protocols by allowing the simulator to interface to an existing memory simulator such as Ruby; 3) it provides the ability to do cycle-accurate modeling of different on-chip network topologies, routing schemes, and router micro-architectures; and 4) it frees up resources on the FPGA to increase the number of physical cores or to incorporate a subset of the memory hierarchy. We have presented some of our preliminary work on HySIM and described our ongoing efforts to improve its performance.

9. REFERENCES

- [1] H. Angepat, D. Sunwoo, and D. Chiou, "RAMP-White: An FPGA-based coherent shared memory parallel computer emulator," in *8th Annual Austin CAS Conference*, 2007.
- [2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sadashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: <http://doi.acm.org/10.1145/2024716.2024718>
- [3] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The m5 simulator: Modeling networked systems," *Micro, IEEE*, vol. 26, no. 4, pp. 52–60, 2006.
- [4] C. Chang, J. Wawrzynek, and R. Brodersen, "BEE2: a high-end reconfigurable computing system," *Design Test of Computers, IEEE*, vol. 22, no. 2, pp. 114–125, 2005.
- [5] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 40. Washington, DC, USA: IEEE Computer Society, 2007, pp. 249–261. [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2007.16>
- [6] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, July 2008.
- [7] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz, "RAMP Blue: A message-passing manycore system in FPGAs," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 54–61.
- [8] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, ser. PACT '06. New York, NY, USA: ACM, 2006, pp. 23–32. [Online]. Available: <http://doi.acm.org/10.1145/1152154.1152162>
- [9] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005. [Online]. Available: <http://doi.acm.org/10.1145/1105734.1105747>
- [10] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010, pp. 1–12.
- [11] T. Oguntebi, S. Hong, J. Casper, N. Bronson, C. Kozyrakis, and K. Olukotun, "Farm: A prototyping environment for tightly-coupled, heterogeneous architectures," in *Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 221–228. [Online]. Available: <http://dx.doi.org/10.1109/FCCM.2010.41>
- [12] M. Pellauer, M. Adler, M. Kinsky, A. Parashar, and J. Emer, "HASim: FPGA-based high-detail multicore simulation using time-division multiplexing," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 2011, pp. 406–417.
- [13] D. Sanchez and C. Kozyrakis, "ZSim: fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 475–486. [Online]. Available: <http://doi.acm.org/10.1145/2485922.2485963>
- [14] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanovic, "RAMP Gold: an FPGA-based architecture simulator for multiprocessors," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 463–468.