

# RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors

Zhangxi Tan, Andrew Waterman, Rimas Avizienis, Yunsup Lee, Henry Cook,  
David Patterson, Krste Asanović  
The Parallel Computing Laboratory  
CS Division, EECS Department, University of California, Berkeley  
{xtan,waterman,rimas,yunsup,hcook,pattsrn,krste}@eecs.berkeley.edu

## ABSTRACT

We present RAMP Gold, an economical FPGA-based architecture simulator that allows rapid early design-space exploration of manycore systems. The RAMP Gold prototype is a high-throughput, cycle-accurate full-system simulator that runs on a single Xilinx Virtex-5 FPGA board, and which simulates a 64-core shared-memory target machine capable of booting real operating systems. To improve FPGA implementation efficiency, functionality and timing are modeled separately and host multithreading is used in both models. We evaluate the prototype's performance using a modern parallel benchmark suite running on our manycore research operating system, achieving two orders of magnitude speedup compared to a widely-used software-based architecture simulator.

## Categories and Subject Descriptors

C.5.3 [Computer System Implementation]: Microprocessors; I.6.8 [Simulation and Modeling]: Discrete Event

## General Terms

Design, Performance, Experimentation

## Keywords

Multiprocessors, FPGA, Simulation

## 1. INTRODUCTION

Evaluating new architectural ideas with hardware prototyping is often prohibitively expensive. Computer architects have therefore employed simulation for early-stage microarchitectural experiments, such as exploring the memory hierarchy design space. Software simulators [4, 8, 19, 20] have been the most popular approach because of their low cost, simplicity, and extensibility. Furthermore, in the earlier era of exponentially increasing uniprocessor performance, software simulators became correspondingly faster without any software engineering effort.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2010, June 13-18, 2010, Anaheim, California, USA  
Copyright 2010 ACM ACM 978-1-4503-0002-5 ...\$10.00.

Unfortunately, the shift to multicore architectures [3] both increases the complexity of systems that architects want to model while largely eliminating the single-thread performance improvements that software simulators have relied upon for scaling. Worse yet, cycle-accurate software simulators are difficult to parallelize efficiently because fine-grained synchronization limits speedup, and relaxing this cycle-level synchronization reduces simulation accuracy [17, 18].

FPGAs have become a promising vehicle to bridge this simulation gap [22]. FPGA capacity has been scaling with Moore's Law, which perfectly matches the growth of the core count on a single processor die. Modern FPGAs have hundreds of SRAM blocks (e.g., Xilinx BRAM), which, as we later discuss, are essential for efficient timing modeling. Moreover, cycle-level synchronization is much faster on FPGAs than in software.

In this paper, we present RAMP Gold, a cycle-accurate FPGA-based architecture simulator<sup>1</sup>. RAMP Gold is efficient: we simulate a 64-core manycore system at almost 50 MIPS on an off-the-shelf \$750 Xilinx XUP board, achieving orders of magnitude speedup over software-based simulators. At the same time, RAMP Gold maintains much of the configurability and extensibility of software-based simulators by decoupling the correct execution of the ISA from the modeling of the simulated system's timing.

Designing a high-throughput simulator on an FPGA is a dramatically different exercise from prototyping the simulated machine itself. We discuss the design decisions behind RAMP Gold and its FPGA implementation, then analyze its performance and compare it to that of Simics+GEMS [14, 15] on the popular PARSEC benchmark suite [5], showing that while functional-only simulations run at about the same speed on both platforms, RAMP Gold runs 263× faster on average with detailed timing models.

## 2. RAMP GOLD DESIGN STRATEGY

We call the machine being simulated the *target* and the machine on which the simulation runs the *host*. The most intuitive approach to simulating a manycore target on an FPGA is to replicate hardware just as in the target machine, using a soft-core processor implementation. Naively mapping these cores to FPGAs, however, is inefficient and inflexible. RAMP Gold's efficient design is based on several observations that distinguish it from other FPGA-based simulators and soft-cores:

<sup>1</sup>RAMP Gold source code is available at <http://ramp.eecs.berkeley.edu/gold>

1. *FPGAs don't implement wide multiplexers well.* This observation led to an unbypassed pipeline design that avoids wide forwarding-path multiplexers. We found by removing forwarding logic in a popular FPGA soft-core processor [2], pipeline area is reduced by 26%-32% and frequency is boosted by 18%-58% under different CAD tool optimization strategies.
2. *FPGAs have plenty of RAM.* This observation, combined with the lack of bypass paths, led to a multithreaded design. Simulation performance arises from many simulation threads per FPGA rather than from complex simulation pipelines optimized for single-thread performance. We call this strategy *host-multithreading*: using multiple threads to simulate different target cores. Note that host-multithreading neither implies nor prohibits a multithreaded target architecture.
3. *Modern FPGAs have hard-wired DSP blocks.* Execution units, especially FPU, dominate LUT resource consumption when implementing a processor on an FPGA. If we map functional units to DSP blocks rather than just LUTs, we can devote more resources to timing simulation
4. *DRAM accesses are relatively fast on FPGAs.* Logic in FPGAs often runs slower than DRAM because of on-chip routing delays. This insight greatly simplifies RAMP Gold's host memory system, as large, associative caches are not needed for high performance.
5. *FPGA primitives run faster but have longer routing delays.* FPGA primitives, such as DSPs and BRAMs, run at high clock rates compared to random logic, but their fixed on-die location often exacerbates routing delays. This observation led to a deep model pipeline.

Like many software simulators [4, 8], RAMP Gold separates the modeling of target timing and functionality. The *functional model* is responsible for executing the target ISA correctly and maintaining architectural state, while the *timing model* determines the time the target machine takes to run an instruction. The benefits of this functional/timing split are:

1. *Simplified FPGA mapping of the functional model.* The separation allows complex operations to take multiple host cycles. For example, a highly-ported register file can be mapped to a block RAM and accessed in multiple host cycles, avoiding a large, slow mapping to FPGA registers and muxes.
2. *Improved modeling flexibility and reuse.* The timing model can be changed without modifying the functional model, reducing modeling complexity and amortizing the functional model's design effort.
3. *Enable a highly-configurable abstracted timing model.* Splitting timing from function allows the timing model to be more abstract. For example, a timing model might only contain target cache metadata. Different cache sizes could then be simulated without resynthesis by changing how the metadata RAMs are indexed and masked at runtime.

### 3. RELATED WORK

RAMP Gold is inspired in part by several recent works from the FPGA community. Fort et al. [9] employed multithreading to improve utilization of soft processors with little area cost. Protoflex [7] is an FPGA-based full-system simulator without a timing model, and is designed to provide similar functionality to Simics [14] at FPGA-accelerated speeds. ProtoFlex employs host multithreading to simulate multiple SPARC V9 target cores with a single host pipeline but lacks a hardware floating-point unit as it targets commercial workloads like OLTP; its performance thus suffers on arithmetic-intensive parallel programs.

HASim [16] is another FPGA-based simulator that employs a split functional/timing architecture similar to RAMP Gold. FAST [6] is a hybrid FPGA-based simulator, whose timing model is in FPGAs but whose functional model is in software. FAST requires substantial communication bandwidth between the CPU and FPGA, which may limit simulation scalability.

Researchers have explored many FPGA-based simulation techniques that vary in modeling fidelity, performance, and ease of modification. A taxonomy of these approaches and further discussion of related work is found in [21].

### 4. RAMP GOLD DESIGN AND IMPLEMENTATION

RAMP Gold comprises about 36,000 lines of SystemVerilog with minimal third-party IP blocks. Our first production system targets the Xilinx Virtex-5 and is deployed on a low-cost XUP board<sup>2</sup>.

RAMP Gold employs many advanced FPGA optimizations and is designed from the ground up with reliability in mind. We have operated five boards for two weeks at a wide range of die temperatures—between 40 and 110 degrees Celsius—without any hard or soft errors.

Figure 1 shows the structure of RAMP Gold. The timing and functional models are both host-multithreaded. The functional model maintains architected state and correctly executes the ISA. The timing model determines how much time an instruction takes to execute in the target machine and schedules threads to execute on the functional model accordingly. The interface between the functional and timing models is designed to be simple and extensible to facilitate rapid evaluation of alternative target memory hierarchies and microarchitectures.

#### 4.1 Functional Model

The functional model is a 64-thread feed-through pipeline with each thread simulating an independent target core. The functional model supports the full SPARC V8 ISA in hardware, including floating point and precise exceptions. It also has sufficient hardware to run an operating system, including MMUs, timers, and interrupt controllers. We have validated the functional model using the SPARC V8 certification suite from SPARC International, and we can boot the Linux 2.6.21 kernel as well as ROS, a prototype manycore research operating system [10, 13]. The functional model has been highly optimized for the Virtex-5 FPGA fabric, and employs the following mapping optimizations:

<sup>2</sup>The Xilinx Virtex-5 OpenSPARC Evaluation Platform costs academics \$750. <http://www.digilentinc.com/>

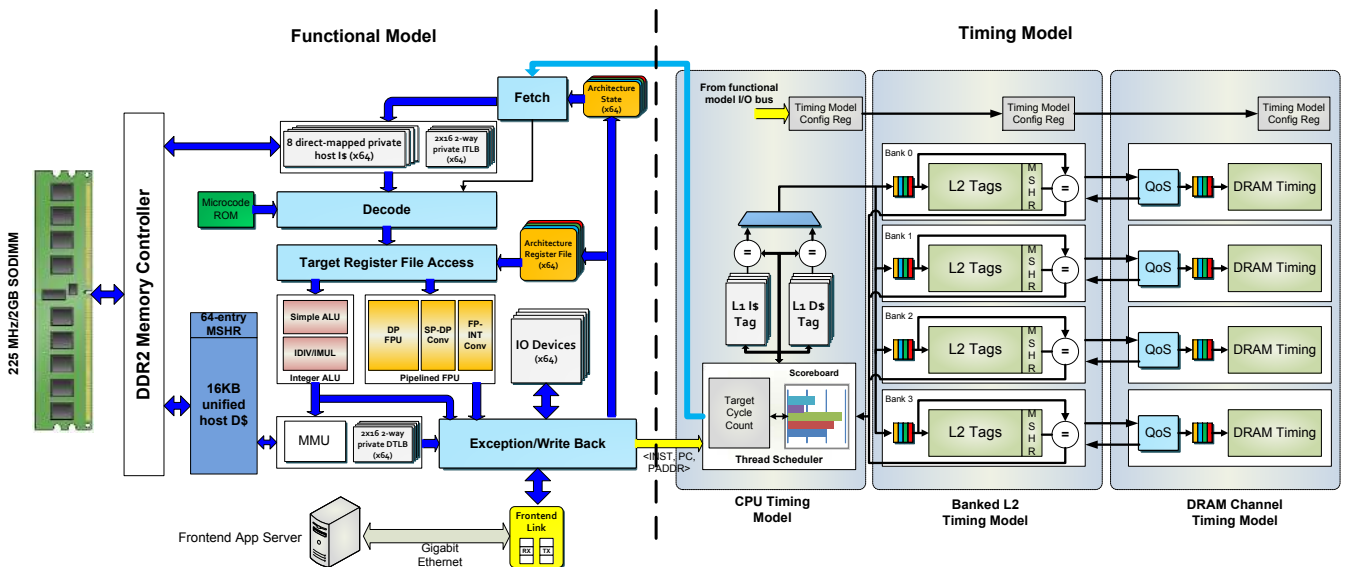


Figure 1: RAMP Gold Structure.

1. *Routing-optimized pipeline:* The functional pipeline is 13 stages long. Some pipeline stages are dedicated to signal routing to BRAMs and DSPs.
2. *Microcode for complex operations:* The functional/timing split allows us to implement the functional pipeline as a microcode engine. Complex SPARC operations, such as atomic memory instructions and traps, are handled using microcode in multiple pipeline passes. The microcode engine also makes it easier to prototype extensions to the ISA.
3. *DSP-mapped ALU:* DSP blocks in FPGAs have been greatly enhanced in recent generations to support logical operations and pattern detection, in addition to traditional multiply-accumulate operations. We mapped the integer ALU and flag generation to two Virtex-5 DSPs and the FPU to fourteen DSPs.
4. *Simple host cache and TLB:* Each thread has a private direct-mapped 256-byte instruction cache, a 32-entry instruction TLB, and a 32-entry data TLB. 64 threads share a small 16 KB lockup-free direct-mapped data cache that supports up to 64 outstanding misses. The host caches and TLBs have no effect on the target timing — they exist solely to accelerate functional simulation. The unusually small host data cache is a deliberate, albeit peculiar, design decision that we discuss in the next section.
5. *Fine-tuned block RAM mappings:* RAMP Gold is a BRAM-dominated design. In the functional model, the register files, host caches, and TLBs are manually mapped to BRAMs for optimal resource usage. In addition, we double-clocked all BRAMs for higher bandwidth. Each BRAM is protected by either ECC or parity for longer and larger scale experiments that require many FPGAs, as the BRAM soft-error rate of 65 nm Xilinx FPGAs nearly doubles compared to the 90 nm generation [12].

## 4.2 Host Memory Interface

The functional model connects to a single-channel 2 GB DDR2 SODIMM on the XUP board running at 225 MHz through a multiport crossbar with an asynchronous request interface. Currently the DRAM serves only as the storage for the target machine, but in a future revision it will also store expanded timing metadata.

Implementing a reliable, high-speed DRAM controller is challenging on a low-cost FPGA board because of analog signal issues and uncertainties introduced by the CAD tools. Such design problems become more obvious with the significant memory-level parallelism offered by a multithreaded design. We created our own DRAM controller based on the Microsoft BEE3 design [1], and added ECC to ameliorate signal-integrity issues. Finally, we floorplanned the memory controller to improve routing quality and mitigate the nondeterminism introduced by CAD tools.

## 4.3 Timing Model

The timing model tracks the performance of the 64 target cores. Our initial target processor is an in-order single-issue core that sustains one instruction per cycle, except for instruction and data cache misses. Each target core has private L1 instruction and data caches. The cores share a lockup-free L2 cache via a nonblocking crossbar interconnect. Each L2 bank connects to a DRAM controller, which models delay through a first-come-first-serve queue with a fixed service rate. A detailed model of cache coherence timing on realistic interconnects is among our future work; we expect it to fit within the current design framework.

Only the timing model has to change to model a wide variety of systems, amortizing the considerable design effort of the functional model. For example, we can model performance of a system with large caches by keeping only the cache metadata in the timing model. In our current timing model, we store all L1 and L2 cache tags in a large number of BRAMs on FPGAs and leverage the parallelism in the circuit to perform multiple highly associative lookups in one host cycle. On the Virtex-5 LX110T FPGA, the design

supports up to 12 MB of total target cache.

Most of the timing model parameters can be configured at runtime by writing control registers that reside on the I/O bus. Among these are the size, block size, and associativity of L1 and L2 caches, the number of L2 cache banks and their latencies, and DRAM bandwidth and latency. To support dynamic cache configuration, we fix the maximum cache sizes and associativities at synthesis time; at runtime, we mask and shift the cache index according to the programmed configuration.

To measure target performance, we implement 657 64-bit hardware performance counters. The 64 cores each have 10 private counters mapped to LUTRAMs, and 17 global counters mapped to registers. Among the events we count are target L1 and L2 cache hits, misses, and writebacks; instructions retired by type; and target clock cycles. We also have a number of host performance counters to measure the simulator’s performance itself. The counters are accessed via a ring interconnect to ease routing pressure.

The timing model largely consists of behavioral SystemVerilog and relies on the memory compiler for BRAM allocation. This coding style enables the rapid prototyping of different microarchitectural ideas. Leveraging many high-level language constructs in SystemVerilog, our manycore timing model comprises only 1,000 lines of code. As an example of its flexibility, we implemented a simplified version of the Globally Synchronized Frames [11] framework for memory-bandwidth QoS in about 100 lines of SystemVerilog code and three hours of implementation effort.

#### 4.4 Debugging and Infrastructure

To communicate with the simulator, we embed a microcode injector into the functional pipeline, which we connect to a front-end Linux PC via a gigabit Ethernet link. This front-end link doubles as a debugging interface and as the simulator control mechanism: we use it to start and stop simulation, load programs, and modify or checkpoint architectural state without affecting target timing. The front-end link also allows us to forward file I/O and console system calls to the Linux PC.

In addition to hardware simulation models, RAMP Gold provides a systematic design and verification infrastructure. Our target compiler toolchain is based on GCC. All user programs running in the target machine adhere to the OpenSolaris ABI, so the same application binaries can run on RAMP Gold and commercial SPARC machines. To help verify model functionality, the front-end server supports multiple backends other than the RAMP Gold hardware, including a fast SPARC V8 instruction set simulator written in C++ and an interface to a Verilog simulator.

## 5. EVALUATION

In this section we compare RAMP Gold against a popular software simulator.

### 5.1 Physical Implementation

To map RAMP Gold to the Xilinx Virtex-5 LX110T-1, we synthesize our design with Synopsys Synplify Premier c-200906sp1 and use Xilinx ISE 11.3 for place and route. The core clock rate is 90 MHz, with some components double-clocking at 180 MHz. It takes about 2 hours to synthesize, place, and route the design on a mid-range workstation.

Figure 2 depicts the final layout of the placed and routed

design. Table 1 shows the detailed breakdown of resource usage. The functional model uses 6,928 LUTs and 9,981 registers; it is thus the largest module on the FPGA. The high register utilization (relative to LUT utilization) is due to the functional model’s deep pipeline. Overall, RAMP Gold utilizes 28% of the LUTs, 34% of the registers, and 90% of the BRAMs available on a LX110T FPGA.

Even though RAMP Gold takes advantage of 16 DSP primitives, the arithmetic units still dominate resource usage in the functional model, especially the FPU. This result highlights the importance of mapping arithmetic to DSP slices, as these structures would be substantially larger if mapped to LUTs.

As we expected, the overhead of multithreading the RAMP Gold functional pipeline is minimal. Only 320 LUTs, 266 registers, and 20 18-Kb BRAMs are used to hold the architected state of 64 SPARC cores.

Compared to the functional model, the timing model uses many 18-Kb BRAM resources to emulate the target’s large cache. Since 192 BRAMs are spread across the whole FPGA, as shown in the floorplan, we see that pipelining the timing model to tolerate routing delay is essential.

The 657 performance counters and their interconnect consume 3,543 LUTs and 4,446 registers—as many resources as the execution units. The ring interconnect gives the placement tool considerable freedom as compared to a bus, which can be seen in the floorplan as the counters are not clustered together but rather distributed about the FPGA.

Interestingly, the memory controller is divided into multiple regions. This results from the ad-hoc layout of the DRAM I/O on the XUP board.

Module	LUT	Register	BRAM	DSP
Func. model	6,928	9,981	54	16
Int ALU	926	1,257	0	2
FPU	2,751	3,232	0	14
Architected state	320	266	20	0
Host Cache	760	942	18	0
Host TLB+MMU	666	754	16	0
Other	1,505	3,530	0	0
Timing model	5,612	5,893	192	0
L1 TM	1,182	2,827	64	0
L2/DRAM TM	4,430	3,066	128	0
Perf. Counters	3,543	4,446	0	0
Misc.	3,384	3,586	21	0
Overall	19,467	23,906	267	16
Percent Utilization	28%	34%	90%	25%

Table 1: RAMP Gold area breakdown on Virtex-5 LX110T

### 5.2 Simulation Performance Evaluation

To measure RAMP Gold’s simulation performance, we run a subset of PARSEC [5] on top of our manycore research OS. Table 2 shows the target machine configuration. We run the same benchmarks on Virtutech Simics [14], a popular software simulator. Simics is run with varying levels of architectural modeling detail: pure functional simulation, the simple Simics g-cache timing module, and the Multifacet GEMS [15] memory hierarchy timing module, Ruby. We run

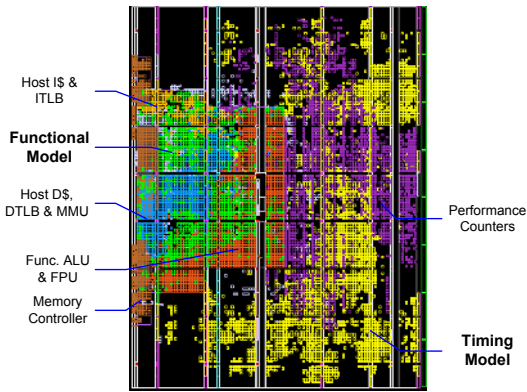


Figure 2: RAMP Gold floorplan on a Xilinx Virtex5 LX110T device

Attribute	Setting
CPUs	64 single-issue in-order cores @ 1 GHz
L1 Instruction Cache	Private, 32 KB, 4-way set-associative, 128-byte lines
L1 Data Cache	Private, 32 KB, 4-way set-associative, 128-byte lines
L2 Unified Cache	Shared, 8 MB, 16-way set-associative, 128-byte lines, inclusive, 4 banks, 10 ns latency
Off-Chip DRAM	2 GB, 4×3.2 GB/sec channels, 70 ns latency

Table 2: System parameters of the target machine simulated by RAMP Gold and Simics.

Simics on a 2.2-GHz AMD Opteron with 4 GB of DRAM.

Figure 3 shows RAMP Gold’s speedup over Simics as the number of target cores vary. RAMP Gold’s performance improves as the number of target cores grows because multithreading gradually improves pipeline utilization. In contrast, Simics’ performance degrades super-linearly with more target cores. At 8 target cores, RAMP Gold’s speedup is modest: the geometric mean speedup is 15× over GEMS or 10× over g-cache. When simulating 64 cores, on the other hand, we see a geometric mean speedup of 263× over GEMS, with a max speedup of 806×. In this configuration, RAMP Gold is even faster than Simics functional simulation.

### 5.3 Host Performance Evaluation

One target cycle may be simulated using multiple host cycles. For example, RAMP Gold takes multiple pipeline passes to resolve a host cache miss or execute a complex instruction. To quantify the performance impact of these pipeline ‘replays’, we added several host performance counters. Figure 4 illustrates the detailed host cycle breakdown running PARSEC benchmarks with 64 target cores.

The most significant overhead is timing synchronization: not until all instructions from a given target cycle have retired do we begin instruction issue for the next target cycle. We expect most of this overhead can be recovered by more efficient thread scheduling. The functional pipeline is also idle when target cores are stalled. Streamcluster, for example, has a high target cache miss rate. The functional model is thus underutilized while target stalls are modeled.

Perhaps the most interesting result is the effectiveness of

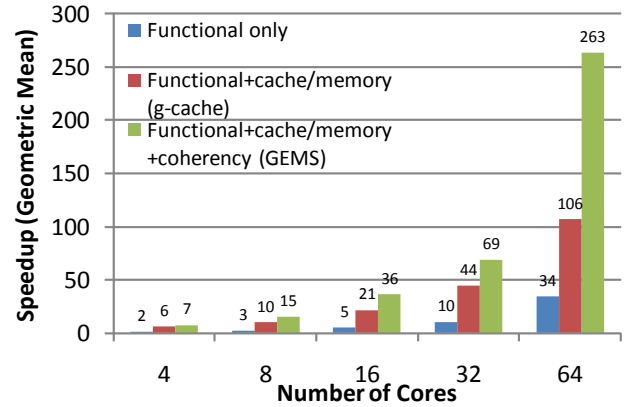


Figure 3: Speedup of RAMP Gold over Simics.

the small host caches. Figure 5 shows that their small size—256 bytes for instructions and 16 KB for data—sometimes results in high miss rates, as we would expect. Nevertheless, host cache misses collectively account for no more than 6% of host clock cycles. DRAM’s relatively low latency—about 20 clock cycles—and the ability of multithreading to tolerate this latency are largely responsible for this peculiar design point. Thus, rather than spending BRAMs on large, associative host caches, we can dedicate these resources to timing models. Nevertheless, providing a small cache is still valuable to exploit the minimum 32-byte DRAM burst.

Other causes of replay include host TLB misses, floating-point operations, integer multiplication and division, and three-operand store instructions. Collectively, these account for less than 10% of host cycles across these benchmarks.

We also measured host DRAM bandwidth utilization, and found we never exceed 15% of the peak bandwidth, indicating that a single-channel memory system is sufficient for this design for these benchmarks.

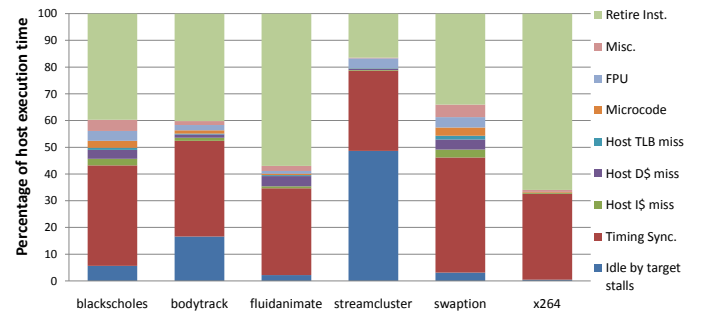
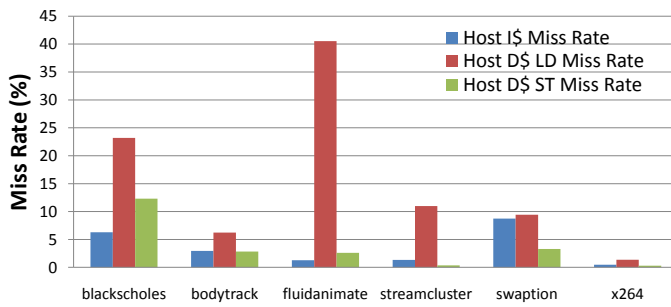


Figure 4: Host cycle breakdown for RAMP Gold running PARSEC.

## 6. DISCUSSION AND CONCLUSION

RAMP Gold’s current implementation employs a single functional pipeline and a single timing pipeline with moderate logic resource consumption on a mid-size FPGA. The design is limited in the total cache capacity we can simulate by the BRAM consumption of the timing model. In the future, we plan to remove this constraint by moving the timing



**Figure 5: RAMP Gold Host Cache Miss Rate running PARSEC**

model’s target cache tags to external DRAM and using on-chip BRAM as a “cache tag cache.” We are also interested in using multiple functional and/or timing pipelines to provide two types of scaling: 1) weak scaling: increase number of simulated cores with the same per-thread performance; 2) strong scaling: increase per-thread performance without increasing simulated cores.

For many reasons, we believe the multicore revolution means the research community needs a boost in simulation performance. RAMP Gold, which simulates 64 SPARC CPUs over  $250\times$  faster than a popular software simulator on a \$750 Xilinx Virtex-5 board, demonstrates the cost-performance benefit of FPGA-based simulation. RAMP Gold’s design also demonstrates that designing FPGA-based architecture simulators is dramatically different from designing multicore processors in either ASICs or in FPGAs.

## 7. ACKNOWLEDGMENTS

The RAMP collaboration was supported by funding from NSF grant number CNS-0551739. The evaluation study was funded by DARPA Award FA8650-09-C-7907. This research was also supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, Samsung, and Sun Microsystems. Special thanks to Xilinx for their continuing financial support, and donations of FPGAs and development tools. We also appreciate the financial support provided by the Gigascale Systems Research Center (GSRC). We’d also like to thank SPARC International, Inc. for donating the SPARC v8 verification suite.

## 8. REFERENCES

- [1] DDR2 DRAM Controller for BEE3, online at <http://research.microsoft.com/en-us/projects/BEE3/>, 2007.
- [2] Leon3 Processor, <http://www.gaisler.com>, 2009.
- [3] K. Asanović et al. A view of the parallel computing landscape. *Commun. ACM*, 52(10):56–67, 2009.
- [4] T. Austin et al. SimpleScalar: An Infrastructure for Computer System Modeling. *Computer*, 35(2):59–67, 2002.
- [5] C. Bienia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *PACT ’08*, pages 72–81, New York, NY, USA, 2008. ACM.
- [6] D. Chiou et al. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In *MICRO ’07*, pages 249–261, Washington, DC, USA, 2007.
- [7] E. S. Chung et al. ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 2(2):1–32, 2009.
- [8] J. Emer et al. Asim: A Performance Model Framework. *Computer*, 35(2):68–76, 2002.
- [9] B. Fort et al. A Multithreaded Soft Processor for SoPC Area Reduction. In *FCCM ’06*, pages 131–142, Washington, DC, USA, 2006.
- [10] K. Klues et al. Processes and Resource Management in a Scalable Many-core OS. In *HotPar09*, Berkeley, CA, 03/2010 2010.
- [11] J. W. Lee et al. Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks. In *ISCA ’08*, pages 89–100, Washington, DC, USA, 2008.
- [12] A. Lesea. Continuing experiments of atmospheric neutron effects on deep submicron integrated circuits. *Xilinx White Paper 286*, 2009.
- [13] R. Liu et al. Tessellation: Space-Time Partitioning in a Manycore Client OS. In *HotPar09*, Berkeley, CA, 03/2009 2009.
- [14] P. S. Magnusson et al. Simics: A Full System Simulation Platform. *IEEE Computer*, 35, 2002.
- [15] M. M. K. Martin et al. Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 33(4):92–99, 2005.
- [16] P. Michael et al. A-port networks: Preserving the timed behavior of synchronous systems for modeling on fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 2(3):1–26, 2009.
- [17] J. E. Miller et al. Graphite: A Distributed Parallel Simulator for Multicores. In *HPCA-16*, January 2010.
- [18] S. Mukherjee et al. Wisconsin Wind Tunnel II: A Fast, Portable Parallel Architecture Simulator. *IEEE Concurrency*, 8(4):12–20, 2000.
- [19] V. S. Pai et al. RSIM Reference Manual. Version 1.0. Technical Report 9705, Department of Electrical and Computer Engineering, Rice University, July 1997.
- [20] M. Rosenblum et al. Using the SimOS machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Simulation*, 7(1):78–103, 1997.
- [21] Z. Tan et al. A Case for FAME: FPGA Architecture Model Execution. In *ISCA ’10*, 2010.
- [22] J. Wawrzynek et al. RAMP: Research Accelerator for Multiple Processors. *IEEE Micro*, 27(2):46–57, 2007.