Midas: An FPGA-based Architecture Simulator for Multiprocessors

ABSTRACT

We present Midas, an economical FPGA-based architecture simulator that allows rapid early design-space exploration of manycore systems. Midas models target-system timing and functionality separately, and it employs host-multithreading for an efficient FPGA implementation. It is a high-throughput, cycle-accurate full-system simulator, capable of booting real operating systems. The Midas prototype runs on a single Xilinx Virtex-5 FPGA board and simulates a 64-core shared-memory target machine. We evaluate its 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.

1. INTRODUCTION

Hardware prototyping is often a prohibitively expensive means to evaluate new architectural ideas. Computer architects have thus employed simulation for early-stage microarchitectural experimentation, such as exploring the memory hierarchy design space. Software simulators [9, 4, 17, 18] have been the most popular variant 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 effort.

Unfortunately, the recent shift to multicore architectures [3] both increases the complexity of the systems 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 comes at the cost of simulation accuracy [16, 15].

As observed by the RAMP (Research Accelerator for Multiple Processors) project [19], FPGAs have become a promising vehicle to bridge this simulation gap. 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 Midas, a cycle-accurate FPGA-based architecture simulator. Midas is quite 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, Midas maintains much of the configurability and extensibility of software-based simulators by decoupling correct ISA execution 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 Midas and its FPGA implementation, then analyze its performance and compare it to that of Simics+GEMS [13, 14] on the popular PARSEC benchmark suite [5], showing that while functional-only simulations run at about the same speed on both platforms, Midas run 269× faster with detailed timing models.

2. MIDAS 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. Midas' efficient design is based on several observations that distinguish it from other FPGA-based simulators and soft-cores:

1. *FPGAs are poor at wide multiplexers.* This observation led to an unbypassed pipeline design that avoids wide forwarding path multiplexers. We found by removing forward logic in a popular FPGA processor soft-core [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 comes from lots of 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 FPUs, dominate LUT resource consumption when implementing a processor on an FPGA. This observation means we can devote more resources to timing simulation if we map our functional units to a couple of DSPs rather than just LUTs.

4. *DRAM accesses are fast on FPGAs.* Logic in FPGAs often runs slower than DRAM because of on-chip routing delay. This observation significantly reduces the complexity of Midas' host memory system: for example, large, associative caches are not necessary 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 pipeline.

Like many software simulators [9, 4], Midas decouples 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 how long an instruction takes to execute in the target machine. The benefits of decoupling are threefold:

1. *Decoupling simplifies the FPGA mapping of the functional model.* Decoupling 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 registers and muxes.

2. *Decoupling improves modeling flexibility and module reuse.* The timing model can be changed without modifying the functional model, reducing modeling complexity and amortizing the functional model's design effort.

3. *Decoupling enables a highly-configurable timing model.* Decoupling timing from function allows the timing model to be more abstract. For example, a timing model might only model target cache metadata; different cache sizes could then be simulated without resynthesis by changing at runtime how the metadata RAMs are indexed and masked.

3. RELATED WORK

Midas is inspired in part by several recent works from the FPGA community. Fort et al [10] employed multithreading to improve utilization of soft processors with little area cost. ProtoFlex [7] is an FPGA-based full-system simulator that employs multithreading to simulate multiple SPARC V9 cores with a single pipeline. Its primary purpose is to accelerate functional warming for a sample-based software simulator, so although it provides a functional cache model to speed up cache warming, it lacks a timing model. ProtoFlex targets commercial workloads like OLTP, so it lacks a floating-point unit; its performance thus suffers on arithmetic-intensive parallel programs, like those in the PARSEC suite.

HAsim [8] is another FPGA-based simulator that employs a decoupled architecture similar to Midas. FAST [6] is a hybrid FPGA-based simulator whose timing model is in FPGAs but whose functional model is in software.

4. MIDAS DESIGN AND IMPLEMENTATION

Midas 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[1].

Midas 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 decoupled architecture of Midas. 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 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 validate 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 a prototype manycore research OS based on [12]. As a result of decoupling, the functional model can be highly optimized to the Virtex5/6 FPGA fabric without loss of timing model flexibility. Below we list our mapping optimizations.

1. *Routing-optimized pipeline:* The functional pipeline is 13 stages. Some pipeline stages are dedicated to signal routing to BRAMs and DSPs.

2. *Microcode for complex operations:* With the help of decoupled timing and functionality, we 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.

[1]The Xilinx Virtex-5 OpenSPARC Evaluation Platform costs $750 for academics. http://www.digilentinc.com/

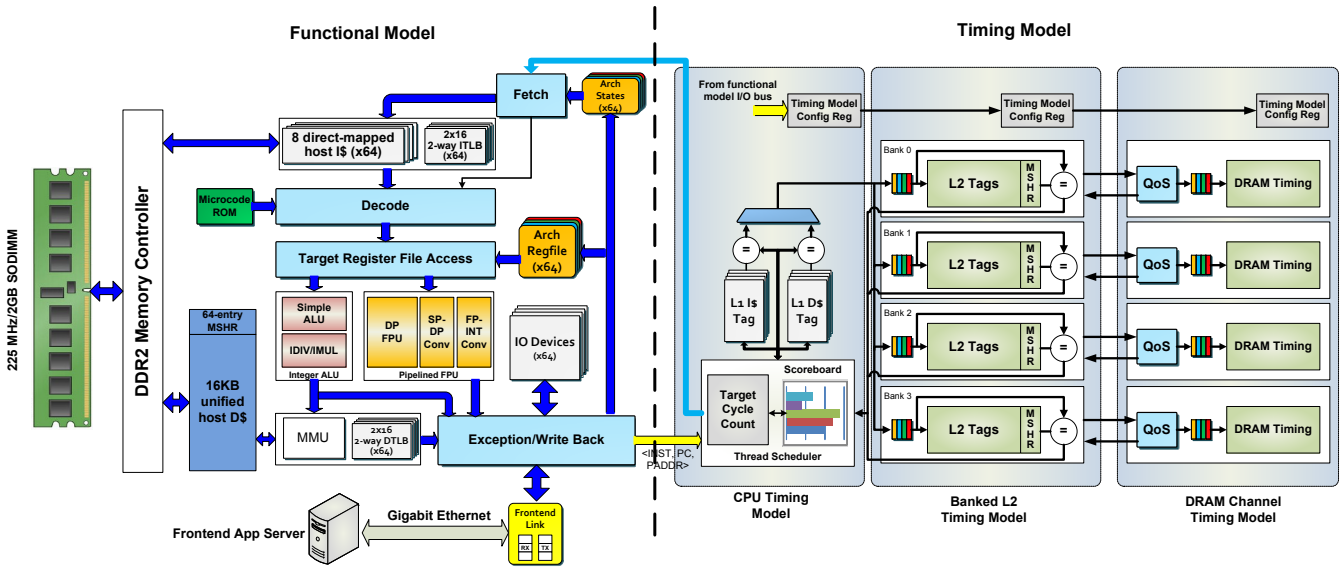

Figure 1: Midas Microarchitecture

4. *Simple host cache and TLB:* Each thread has a private direct-mapped 256B instruction cache, a 32-entry instruction TLB, and a 32-entry data TLB. 64 threads share a small 16KB 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:* Midas 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.

4.2 Host Memory Interface

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

Implementing a reliable, high-speed memory datapath is challenging on a low-cost FPGA board because of analog issues and uncertainties introduced by CAD tools. Such design problems become more obvious with the significant memory-level parallelism offered by a multithreaded design.

To ameliorate signal-integrity issues, we protect the 128-bit memory datapath with ECC, end-to-end from the host caches to the DIMM. We also designed our own memory controller based on the work in [1]; the controller has a better analog interface and a more robust asynchronous CPU-side interface than most existing controllers. Finally, we floorplanned the memory controller to improve routing quality and mitigate the nondeterminism introduced by the CAD tools.

4.3 Timing Model

The timing model just tracks the performance of the 64 target cores. The target processor model is a single in-order 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.

The decoupled timing model greatly expands the domain of systems Midas models; it also amortizes the functional model's considerable design effort over a large range of target machines. For example, we can capture timing 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 are mapped to registers. Among the events we count are target L1 and L2 cache hits, misses, and writebacks, instructions retired by type, and target cycles. We also have a number of host performance counters to quantify the simulator's performance itself. The counters are chained using 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, Midas provides a systematic design and verification infrastructure. Our target compiler toolchain is directly built from the latest GCC without any modification. All user programs running in the target machine adhere to the OpenSolaris ABI, so the same application binaries can run on Midas and commercial SPARC machines. To help verify model functionality, the front-end server supports multiple backends other than the Midas hardware, including a complete RTL simulation interface and a fast SPARC V8 instruction set simulator written in C++.

5. EVALUATION

5.1 Physical Implementation

To map Midas 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 90MHz, with some components double-clocked 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 owed to the functional model's deep pipelining. Overall, Midas utilizes 28% of the LUTs, 34% of the registers, and 90% of the BRAMs available on a LX110T FPGA.

Even though Midas takes advantage of 16 DSP primitives, the execution unit uses more than 50% resources of the functional model, with the FPU dominating the integer ALU. This result highlights the importance of mapping computation to DSPs: these already-large structures would be substantially larger if mapped to LUTs.

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

Compared to the functional model, the timing model uses many 18Kb 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; this result is visible 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.

Midas Component	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: Midas area breakdown on Virtex-5 LX110T

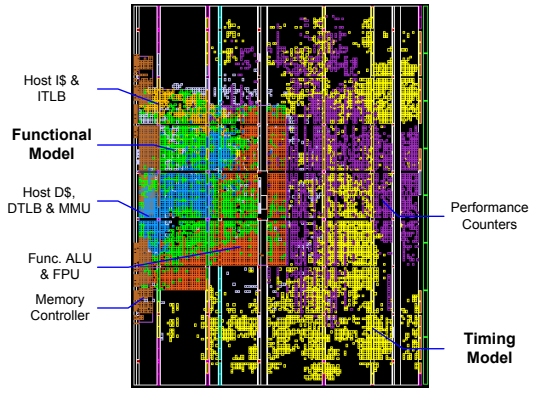

Figure 2: Midas floorplan on a Xilinx Virtex5 LX110T device

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

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

5.2 Simulation Performance Evaluation

To measure Midas' simulation performance, we run a subset of PARSEC [5], a state-of-the-art parallel benchmark suite. All benchmarks run on top of our manycore research OS. Table 2 shows the target machine configuration. We run the same benchmarks on Virtutech Simics [13], a popular software simulator. Simics is run with varying levels of architectural modeling details: pure functional simulation, the simple Simics g-cache timing module, and the Multifacet GEMS [14] memory hierarchy timing module, Ruby. We run Simics on a 2.2 GHz AMD Opteron with 4 GB of DRAM.

Figure 3 shows Midas' speedup over Simics, varying the number of target cores. Midas' performance improves as the number of target cores grows because multithreading gradually improves pipeline utilization. On the other hand, Simics' performance degrades super-linearly with more target cores. Simulating 8 target cores, Midas' 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 269× over GEMS, with a max speedup of 806×. In this configuration, Midas is even faster than Simics functional simulation.

5.3 Host Performance Evaluation

Since we decouple target timing and functionality, one target cycle may be simulated using multiple host cycles. For example, Midas takes multiple pipeline passes to resolve a host cache miss or executes 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 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, as it utilizes 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 the PARSEC benchmarks.

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

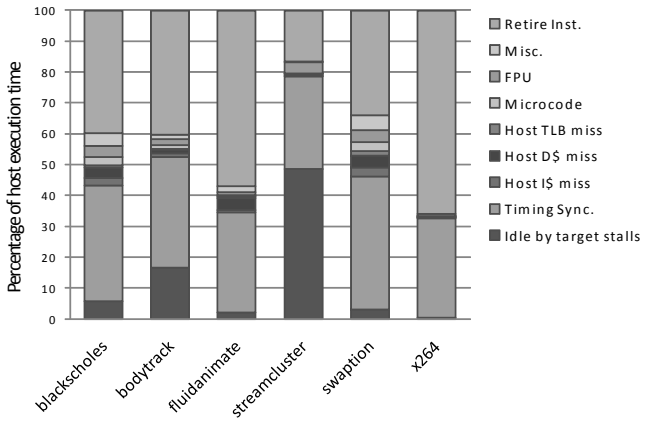

Figure 4: Midas Host Cycle Breakdown running PARSEC

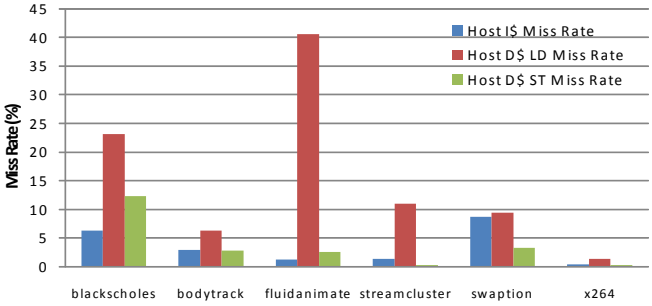

Figure 5: Midas Host Cache Miss Rate running PARSEC

6. DISCUSSION AND CONCLUSION

Midas' 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 by the BRAM consumption in the timing model for simulating large target caches; on-chip BRAM capacity limits the total cache capacity we can simulate. In the future, we plan to remove this constraint by moving the target cache

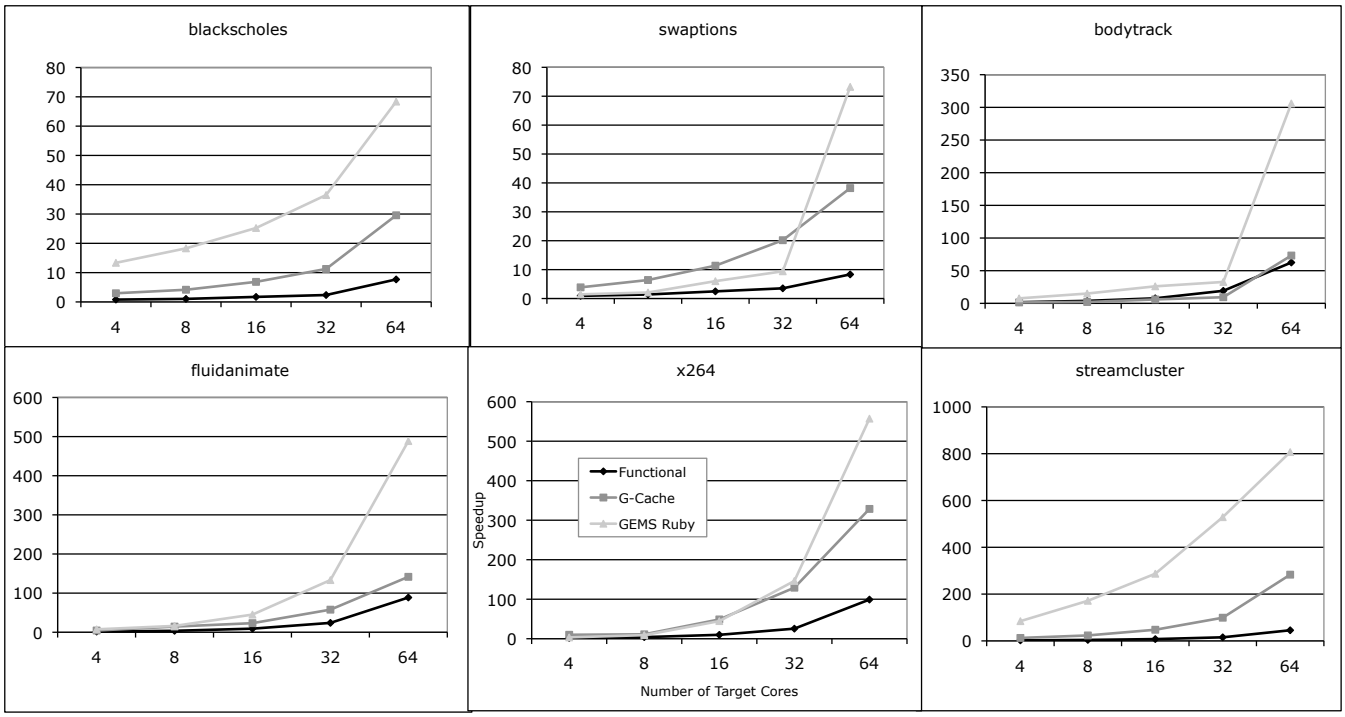

Figure 3: Parsec Benchmark Speedup on Midas

tags in the timing model 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 perform 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. FPGAs have become a promising vehicle to accelerate early-stage architectural investigation. Midas, which simulates 64 SPARC CPUs on a $750 Xilinx Virtex-5 board, demonstrates the cost-performance benefit FPGAs offer for multicore simulation by running over 250× faster than a popular software simulator. Midas' design also demonstrates that designing architecture simulators is dramatically different from designing multicore processors in either ASICs or in FPGAs.

7. REFERENCES

[1] DDR2 DRAM Controller for BEE3, online at http://research.microsoft.com/en-us/projects/BEE3/, 1986.

[2] Leon3 Processor, http://www.gaisler.com, 2009.

[3] K. Asanović et al. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, UC Berkeley, Dec 2006.

[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] N. Dave et al. Implementing a Functional/Timing Partitioned Microprocessor Simulator with an FPGA, Feb. 2006.

[9] J. Emer et al. Asim: A performance model framework. *Computer*, 35(2):68–76, 2002.

[10] B. Fort et al. A multithreaded soft processor for sopc area reduction. In *FCCM '06*, pages 131–142, Washington, DC, USA, 2006.

[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] R. Liu et al. Tessellation: Space-time partitioning in a manycore client os. In *HotPar09*, Berkeley, CA, 03/2009 2009.

[13] P. S. Magnusson et al. Simics: A Full System Simulation Platform. *IEEE Computer*, 35, 2002.

[14] M. M. K. Martin et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[15] J. E. Miller et al. Graphite: A Distributed Parallel Simulator for Multicores. In *HPCA-16*, January 2010.

[16] S. Mukherjee et al. Wisconsin Wind Tunnel II: A Fast, Portable Parallel Architecture Simulator. *IEEE Concurrency*, 8(4):12–20, 2000.

[17] V. S. Pai et al. RSIM Reference Manual. Version 1.0. Technical Report 9705, Department of Electrical and Computer Engineering, Rice University, July 1997.

[18] 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.

[19] J. Wawrzynek et al. RAMP: Research Accelerator for Multiple Processors. *IEEE Micro*, 27(2):46–57, 2007.