A Case for FAME: FPGA Architecture Model Execution

Abstract

Given the multicore microprocessor revolution, we argue that the architecture research community needs a dramatic increase in simulation capacity. We believe FPGA Architecture Model Execution (FAME) simulators can increase the number of useful architecture research experiments per day by two orders of magnitude over Software Architecture Model Execution (SAME) simulators. To clear up misconceptions about FPGA-based simulation methodologies, we propose a FAME taxonomy to distinguish the cost-performance of variations on these ideas. We demonstrate our simulation speedup claim with a case study wherein we employ a prototype FAME simulator, Midas, to research the interaction between hardware partitioning mechanisms and operating system scheduling policy. The study demonstrates the FAME capabilities: we run a modern parallel benchmark suite on a research operating system, simulate 64-core target architectures with multi-level memory hierarchy timing models, and add experimental hardware mechanisms to the target machine. The simulation speedup achieved by our adoption of FAME—250×–allows longer experiments that reach different conclusions than those from shorter experiments necessitated by SAME.

1 Introduction

Computer architects have long used software simulators to explore instruction set architectures, microarchitectures, and approaches to implementation. Compared to hardware prototyping, their low capital cost, relatively low cost of implementation, and ease of change have made them the ideal choice for the early stages of research exploration. In addition, when uniprocessor performance was doubling every 18 months, simulation speed correspondingly doubled every 18 months without any special programming effort.

The recent abrupt transition to multicore architectures [5], however, has both increased the complexity of the systems architects want to simulate and lost the straightforward path to simulator performance scaling. Parallel applications exhibit more complex behaviors than sequential applications, including timing-dependent non-deterministic execution, cache coherence traffic, and operating system scheduler interactions.

The move to multicore was primarily driven by the end of traditional technology scaling, causing power and thermal issues to come to the fore. Application software complexity has also increased, and dynamically generated code is common, including code that is automatically tuned to the hardware implementation on which it runs. Although it is generally well understood how to model these various phenomena, accurate models require detailed cycle-level simulation. Unfortunately, detailed cycle-level simulation is notoriously difficult to parallelize due to the need for regular cycle-by-cycle synchronization, and hence sequential software simulators have fallen far behind the performance required to support the new wave of parallel systems research.

This paper argues that architecture research now faces a crisis in simulation because of the new requirements and the consequences of the move to multicore processors. Indeed, we found that the median instructions simulated per benchmark was similar in ISCA papers in 1998 and 2008. In recent papers, those instructions were simulated across an average of 16 times as many processors, so the number of instructions per processor actually decreased over that decade. To tackle this *simulation gap*, several research groups have been exploring the use of FPGAs to build various forms of FPGA-accelerated architecture simulators. In this paper, we use the terms *Software Architecture Model Execution (SAME)* or *FPGA (Field-Programmable Gate Array) Architecture Model Execution (FAME)* to label the two approaches to multicore simulation. Due to the rapid progress made by the whole FAME community over the last few years, there appears to be considerable confusion about the structure and capabilities of FAME simulators in the broader architecture community. This paper proposes a four-level taxonomy of increasingly sophisticated FAME levels to help explain the capabilities and limitations of various FPGA simulation approaches.

We next present the detailed design of the Midas FAME simulator, a very efficient architectural simulator for early-stage design exploration. We describe how Midas supports various forms of architecture experiment and how modifications would be made to support other studies. The following section provides a quantitative evaluation of the Midas FAME simulator against Simics+GEMS [18, 19], a popular SAME simulator. We compare simulation speeds for the PARSEC benchmark suite [8]. We show that while pure functional simulations on a few cores run at about the same speed, the Midas FAME simulator runs detailed models on 64 cores on average 269× faster than Simics+GEMS, with a maximum speedup of 806×.

2 Multiprocessor Simulation

A modern processor running an application workload is difficult to model analytically, yet building a prototype for each design point is prohibitively expensive. Software simulators have therefore become the primary method used to evaluate architectural design choices. We call the machine being simulated the *target* and the machine on which the simulation runs the *host*.

Simulating parallel target machines is much more complex than simulating uniprocessors. Part of the added complexity is simply that the target hardware is more complex, with multiple cores and a cache-coherent shared memory hierarchy. However, another complexity is that a parallel software runtime must be present to support multithreading or multiprogramming across the multiple cores of the simulated target. For multiprocessor research, trace-driven simulation is still often used despite the inability of traces to capture the effects of timing-dependent execution interleaving, as developing a full system environment capable of running large workloads is difficult.

As with uniprocessor simulators, many parallel simulators only modeled user-level activity of a single application (e.g., RSIM [22]). The SimOS project demonstrated how to run an operating system on top of a fast software simulator [25]. SimOS supported multiple levels of simulation detail, and the fastest version of SimOS used dynamic binary translation to speed target instruction emulation while emulating cache hierarchies in some detail [33]. This research was later incorporated into the commercial product Simics, which allowed researchers to study large application programs and the operating system running together. Augmented with detailed performance models developed by other researchers [19], Simics has become a popular tool in the architecture community; we provide a detailed evaluation of its performance in Section 5.

Sadly, it is difficult to parallelize detailed multiprocessor simulations to run efficiently on parallel host machines. The need for cycle-by-cycle interaction between components limits the parallel speedup possible due to the high cost of software synchronization. If this cycle-by-cycle synchronization is relaxed, parallelized software simulators can attain some speedup but at the cost of needing to validate that the missing interactions do not affect the experiment's results [20, 24].

As with uniprocessors, researchers have considered using sampling to reduce multiprocessor simulation time. Alas, mixed mode simulation and sampling do not work well in general for multiprocessor simulations [7]. For uniprocessors, the program execution should be the same no matter the underlying microarchitecture, and so the architectural state of the processor is correct at the beginning of any sample point for

any target microarchitecture. Such is not the case for multiprocessors, because software thread interleavings change depending on the behavior of the microarchitecture in each core. For example, if you were interested in exploring the impact of relaxed consistency models on multithreaded programs, you might never see the interesting events if the functional simulation used sequential consistency to obtain sample start points. Sampling can give representative results for multiprogrammed workloads, since processes running on different cores generally do not interact via shared memory. Others have argued that transaction-oriented applications such as databases are also amenable to sample-based simulation, since any sample is representative of some legal overlap of independent transactions [32].

Implicit in many techniques used to reduce simulation time is the assumption that software is compiled statically, changes slowly and is independent of the target architecture. Architects have generally treated benchmark programs as static artifacts to measure without understanding the problems being solved or the algorithms and data structures being used. Thus, collections of old software like SPEC were reasonable benchmarks for architectures of the future, as long as SPEC suite changed every five years to reduce gamesmanship. In fact, many architectural studies have just used precompiled binaries distributed with a shared simulator infrastructure. New programs, new programming models, and new programming languages were largely ignored. The multicore revolution is such an abrupt change that this *laissez faire* approach will no longer work, since by definition we need new programs, models, and languages. Existing simulation time shortcuts are less likely to be useful, and simulation latency becomes at least as important as simulation throughput.

3 FPGA Architecture Model Execution

As observed by the RAMP (Research Accelerator for Multiple Processors) project [30], FPGAs have become a promising vehicle for architectural investigation, providing a highly parallel programmable execution substrate that can run simulations several orders of magnitude faster than software. Previously, FPGA processor models were hampered by the need to partition a processor across multiple FPGAs, increasing hardware costs, reducing performance, and significantly increasing development effort. But FPGA capacity has been scaling with Moore's Law. Now, depending on complexity, multiple processors can fit in a single FPGA. Furthermore, future scaling should allow FPGA capability to continue to track simulation demands as the number of cores in target systems grows. Due to high volumes, FPGA boards have also become relatively inexpensive; the Midas design, for example, uses a $750 commercial board.

Multiple groups have now developed working FAME simulators [10, 12, 21, 16, 11], but with perhaps an even greater variety than software simulators and correspondingly a larger variety of tradeoffs between simulator performance, accuracy, and flexibility. Consequently, we have heard much confusion in our discussions with other architects about how FAME relates to prior work using FPGAs for architecture prototyping and chip simulation, and what can and cannot be done using FAME.

In this section, we first present three binary dimensions within which we can categorize FAME approaches. Next, inspired by the five-level RAID classification, we present four levels of FAME that capture the most important design points in this space. Our hope is these FAME levels will help explain FPGA-based emulation approaches to the broader architecture community.

3.1 FAME Implementation Techniques

We use the following three binary dimensions to characterize FAME implementation approaches.

3.1.1 Direct vs. Decoupled

The *Direct* approach is characterized by the direct mapping of a target machine's RTL description into FPGA gates, where a single target clock cycle is executed in a single host clock cycle. An advantage of the direct approach is that, in theory, a re-synthesis of the target RTL for the FPGA provides a guaranteed cycle-accurate model of the target processor. The direct approach has been popular in chip verification; one of the first uses of FPGAs was emulating a new chip design to catch logic bugs before tapeout. Quickturn [14] was an early example, where boxes packed with FPGAs running at about 1–2 MHz could run much larger test programs than feasible with software ECAD logic simulators. Direct emulation is also often used by intellectual property (IP) developers to supply a new core design to potential customers for evaluation and software porting before committing to an ASIC. FPGAs are also used both for *FPGA prototyping*, where FPGAs are used to construct research processors at much lower cost, risk, and design effort compared to a custom chip implementation [21, 16, 27], or to build *FPGA computers*, where FPGAs are the final target technology (e.g., Xilinx Microblaze [2], Convey HC-1 [3]). Although these are often confused with Direct FAME, neither is really intended to provide a model of a target machine (they *are* the target machine).

Direct emulation has become much easier as the growth in FPGA capacity reduces the need to partition monolithic RTL blocks, such as CPUs, across FPGAs, but large system designs may still require many FPGAs. The inefficiency of FPGAs at emulating common logic structures—such as multiported register

files, wide muxes, and CAMs—exacerbates capacity problems.

A more powerful FAME option, which improves efficiency and enables other more advanced options, is to adopt a *Decoupled* design, where a single target clock cycle can be implemented with multiple or even a variable number of host clock cycles. For example, direct mapping of a multi-ported register file is inefficient on FPGAs because discrete FPGA flip-flops are used to implement each register state bit with large combinational circuits used to provide the read ports. A more efficient decoupled model would implement a target multi-ported register file by time-multiplexing a single-ported FPGA RAM over multiple FPGA clock cycles. The drawback of a decoupled design is that models have to use additional host logic to model target time correctly, and a protocol is needed to exchange target timing information at module boundaries if modules have different target to host clock cycle ratios.

3.1.2 Full RTL vs. Abstracted Machine

When the full RTL of a target machine is used to build a FAME model, it ensures precise cycle-accurate timing. However, the desired RTL design is usually not known during early stage architecture exploration, and even if the intended RTL design is known, it can require considerable effort to implement a working version including all corner cases. Even if full correct RTL is available, it may be too unwieldy to map directly to an FPGA.

Alternatively, we can use a higher-level description of the design to construct a FAME model. Abstraction can reduce both model construction effort and FPGA resource needs. The primary drawback is that an abstract model needs validation to ensure accuracy, usually by comparing against RTL or another known good model. If the mechanism is novel, an RTL prototyping exercise might be required to provide confidence in the abstraction. Once validated, however, an abstract component can be reused in multiple designs.

HAsim [12] is an example of the *Abstract* FAME option, where a processor model is divided into separate functional and timing models that do not correspond to structural components in the target machine. Split functional and timing models provide similar benefits as when used in SAME simulators. Only the timing model needs change to experiment with different microarchitectures, and the timing model can be readily parameterized. The parameters, e.g., cache size and associativity, can even be set at runtime in the FAME model without resynthesizing the design, dramatically increasing the number of architecture experiments that can be performed per day.

3.1.3 Single-Threaded vs. Multi-Threaded

A cycle-accurate FAME model synchronizes all model components on every target clock cycle. Some complex components might have long latencies, e.g., to communicate with off-chip memory or across multiple FPGAs. A standard approach to tolerate latencies and obtain greater performance from a processor is to switch threads every clock cycle so that all dependencies are resolved by the next time a thread is executed [4, 9]. The same approach can be applied to the host implementations of FAME models in a technique we call *host multithreading*, and it is particularly applicable to models of parallel target machines. When the target system contains multiple instances of the same component (e.g., cores in a manycore design), the host model can be designed so that one physical FPGA pipeline can model multiple target components by interleaving the component models' execution using multithreading. For example, a single FPGA processor pipeline might model 64 target cores. Alternatively, a single FPGA router pipeline might model 16 on-chip routers. Host multithreading greatly improves utilization of FPGA resources by hiding host communication latencies. For example, while one processor target model is making a request to a memory module, we can interleave the activity of 63 other target processor models. Provided modeling of the memory access takes fewer than 64 FPGA clock cycles, the emulation will not stall. Multithreaded emulation adds additional design complexity but can provide a significant improvement in emulator throughput. ProtoFlex is an example of a FAME simulator that host-multithreads its functional model [11].

3.2 FAME Levels

Level	*Name*	*Example*	*Strength*	*Experiments/day/$1K*
000	Direct FAME	Quickturn, Palladium	Debugging logical design	0.001
001	Decoupled FAME	GreenFlash	Higher clock rate; lower cost	0.667
011	Abstract FAME	HAsim	Simpler, parameterizable design; faster synthesis; lower cost	40.000
111	Multi-threaded FAME	Midas	Lower cost; higher clock rate	170.000

Table 1: Summary of four FAME Levels, including examples.

A combination of these FAME implementation techniques often makes sense. Inspired by the five levels of RAID, the next four sections present a four-level taxonomy of FAME that improves in cost, performance, or flexibility. The four levels are distinguished by their choices from the three options above, so we can number the levels with a three-bit binary number, where the least-significant bit represents Direct (0) vs. Decoupled (1) and the most-significant bit represents Single-Threaded (0) vs. Multi-Threaded (1). Table 1 summarizes the levels and gives examples and the strengths of each level. Each new FAME level lowers

cost and usually improves performance over the previous level, while moving further away from the concrete RTL design of the target.

To quantify the cost-performance difference of the four FAME levels, we propose as a performance measure the number of simulation experiments that can be performed per day. Given the complex dynamics of manycore processors, operating systems, and workloads, we believe the minimum useful experiment is simulating 1 second of target execution time at the finest level of detail for 16 cores at a clock rate of 2 GHz with shared memory and cache coherence. We employ this as an approximate unit to measure an experiment. The same experiment but running for 10 seconds is 10 units, the same experiment but running for 1 second at 64 cores is 4 units, and so on. Note that in addition to host simulation time, you must include the time to set up the experiment (for example, design synthesis time). To get a cost-performance metric, we simply divide the number of experiments per day by the cost of that FAME system. To keep the numbers from getting too small, we calculate experiments per day per $1000 of the cost of the FAME system. The last column of Table 1 estimates this metric for 2009 prices.

3.2.1 Direct FAME (Level 000): (e.g., Quickturn)

The common characteristic of Direct FAME emulation systems, such as Quickturn, is that they are designed to model a single chip down to the gate level with a one-to-one mapping of target cycles to host cycles.

Let's assume we could simulate the gates of 16 cores on a $1 million Direct FAME system at 2 MHz. Each run would then take 2 GHz/2 MHz =1000 seconds or 17 minutes. Because the model is not parameterized, we have to rerun the CAD tool chain for each experiment to resynthesize the design. Given the large number of FPGAs and larger and more complicated description of a hardware-ready RTL design, it can take up to 30 hours to set up a new design [29]. If Direct FAME can do 1 experiment per day, the number of experiments per day per $1000 is 0.001.

In addition to high simulation turnaround time, Direct FAME requires great design effort to change the RTL for each experimental machine, unlike some of the later FAME Levels. Although helpful in the later stages of debugging the design of a real microprocessor intended for fabrication, Direct FAME is too expensive and time consuming to use for early-stage architectural investigations.

Newer gate-level emulation products, such as Cadence Palladium and MentorGraphics Veloce, are no longer based on commercial FPGAs but instead use custom-designed logic simulation engines. However, they still have relatively low target clock rates [1] and cost millions of dollars, though the tools are vastly

superior to the FPGA tools for this purpose.

3.2.2 Decoupled FAME (Level 001): (e.g., Green Flash memory system)

Programmable logic is slow compared to hardwired logic, and some ASIC features, such as multiported register files, map poorly to FPGAs, consuming resources and cycle time. For example, Green Flash [31] can fit two Tensilica cores with floating-point units per medium-sized FPGA, but it runs at only 50 MHz [26]. The memory system uses off-chip DRAM, however, which runs much faster than the logic (200 MHz) and so decoupling is used in the memory system to match the intended target machine DRAM timing.

Performing a 16-core experiment needs 2 BEE3 [13] boards, which cost academics about $15,000 per board, plus the FPGAs and DRAMs, which cost about $3000 per board, or $36,000 total. It takes 8 hours to synthesize and place and route the design and about 2 GHz/50 MHz or 40 seconds to run an experiment. Since this level has a few timing parameters, such as DRAM latency and bandwidth, Green Flash can run about 24 experiments per synthesis [26]. (Alas, the state of FPGA CAD tools means FPGA synthesis is a human-intensive task; only one synthesis can be run per workday.) Thus, the number of experiments per day per $1000 is 24/36 or 0.667. Decoupled FAME (Level 001) improves the cost-performance over Direct FAME (Level 000) by a factor of almost 700×. This speedup is mostly due to processor cores fitting on a single FPGA, thus avoiding the off-chip communication that slows Direct FAME systems; also, Decoupled FAME uses a simple timing model to avoid resynthesis for multiple memory system experiments.

It is both a strength and a weakness of Decoupled FAME that the full target RTL is modeled. The strength is that the model is guaranteed to be cycle accurate. Also, the same RTL design can be pushed through a VLSI flow to obtain custom layout to yield reasonable area, power and timing numbers [28]. The weakness is that designing the full RTL for a system is labor-intensive, and rerunning the tools is slow. This makes Decoupled FAME less suitable for early-stage architecture exploration, where the designer is not ready to commit to a full RTL design. Decoupled FAME thus takes a great deal of effort to perform a wide range of experiments compared to Abstract and Multithreaded FAME. These higher levels, however, require decoupling to implement their timing models, and hence we assume that all the following levels are Decoupled (or odd-numbered in our enumeration).

3.2.3 Abstract FAME (Level 011): (e.g., HAsim)

Abstract FAME allows high-level descriptions for early-stage exploration, which simplifies the design and thereby reduces the synthesis time to under 1 hour. More importantly, it allows the exploration of many design parameters without having to resynthesize at all, which dramatically improves cost-peformance.

Let's assume we need 1 BEE3 board for 16 cores, so the cost is $18,000. To simulate cache coherency, the simulator will take several host cycles per target cycle for every load or store to snoop on the addresses. Let's assume a clock frequency of 65 MHz, as with HAsim [23], and an average number of host cycles per target cycle of 4. The time for one experiment is then 2 GHz/65 MHz $\times$ 4 = 123 seconds. Since human intervention isn't needed to program the FPGAs, the number of experiments per day is 24 hours/123 seconds = 702. The number of experiments per day per $1000 is then 702/18 or about 40. Abstract FAME (Level 011) makes a dramatic improvement in this metric over lower FAME levels: by a factor of almost 60 over Decoupled FAME (Level 001) and a factor of 40,000 over Direct FAME (Level 000).

In addition to the advance in cost-performance, Abstract FAME allows many people to perform architecture experiments without having to modify the RTL, which both greatly lowers the effort for experiments and greatly increases the number of potential experimenters. Once again, the advantages of abstract designs and decoupled designs are so great that we assume any subsequent level is both Abstract and Decoupled.

3.2.4 Multithreaded FAME (Level 111): (e.g., Midas)

The main cost of Multithreaded FAME is more RAM to hold copies of the state of each thread, but RAM is one of the strengths of FPGAs — a single programmable gate can be exchanged for 64-bits of RAM in a Virtex-5. Hence, Multithreaded FAME increases the number of cores that can be simulated efficiently per FPGA. Multithreading can also increase the clock rate of the host simulator by removing items on the critical path, such as bypass paths.

Since we are time-multiplexing the FPGA models, a much less expensive XUP board ($750) suffices. Multithreading reduces Midas' host cycles per target core-cycle to 1.90 (measured) and enables a clock rate of 90 MHz. The time for a 16-core simulation is then $2\,\text{GHz}/90\,\text{MHz} \times 16 \times 1.9 = 675$ seconds. The number of experiments per day is 24 hours / 675 seconds = 128. The number of experiments per day per $1000 is then 128 / .75 or about 170. Multithreaded FAME (Level 111) improves this metric by more than a factor of 4 over Abstract FAME (Level 011), by a factor of about 250 over Decoupled FAME (Level 001),

and by a factor of 170,000 over Direct FAME (Level 000). We expect that Midas V2 will fit three pipelines on a single FPGA and run at 100 MHz, allowing over 550 experiments per day per $1000.

In addition, Multithreaded FAME lowers the entry point cost by a factor of 24 to 48 versus Abstract or Decoupled FAME, making it possible for many more experimenters to do parallel architecture research.

3.2.5 Other Possible FAME Levels

By definition, direct mapping cannot be combined with abstract models or multithreading. An RTL design can be multithreaded, however, whereby every target register is replicated for each threaded instance but combinational logic is shared by time multiplexing. We ignored this Multithreaded RTL combination (101) as a FAME level because, although plausible, we have not seen instances of this combination in practice.

3.2.6 Hybrid FAME Simulators

Although we present levels as completely separate approaches for pedagogic reasons, real systems will combine modules at different levels, or even use hybrid designs partly in FPGA and the rest in software.

An example of a mixed FPGA-only design is often used by System-on-a-Chip IP providers to provide a fast emulation of their IP block to customers. The RTL is mapped to the FPGA is the same as will be mapped to the final ASIC implementation (Direct FAME, Level 001), but the rest of the system is described at an abstract level (Abstract FAME, Level 011). FAST [10] is an example of a hybrid FAME/SAME system, where the functional model is in software and the timing model is in hardware.

4 The Midas FAME Simulator

Midas is a Multithreaded FAME (Level 111) simulator deployed on a single, $750 Xilinx Virtex-5 FPGA board. In this section, we describe the design of Midas and how it attains high efficiency by tailoring the design to the FPGA environment.

The current Midas simulation target is a tiled, shared-memory manycore system with up to 64 single-issue, in-order SPARC V8 cores. Midas is a full-system simulator: it boots the Linux 2.6.21 kernel, as well as ROS, our manycore research operating system. Midas' target machine is highly parameterized. Most simulation options are runtime-configurable: without resynthesizing a new FPGA configuration, we can vary the number of target cores, cache parameters (size, associativity, line size, latency, banking), and DRAM configuration (latency, bandwidth, number of channels). This extensive runtime parameterization

enables expedient design space exploration and comes at little cost to simulator performance: with a detailed memory system timing model, we can simulate over 40 million target core-cycles per second.

4.1 Midas Microarchitecture

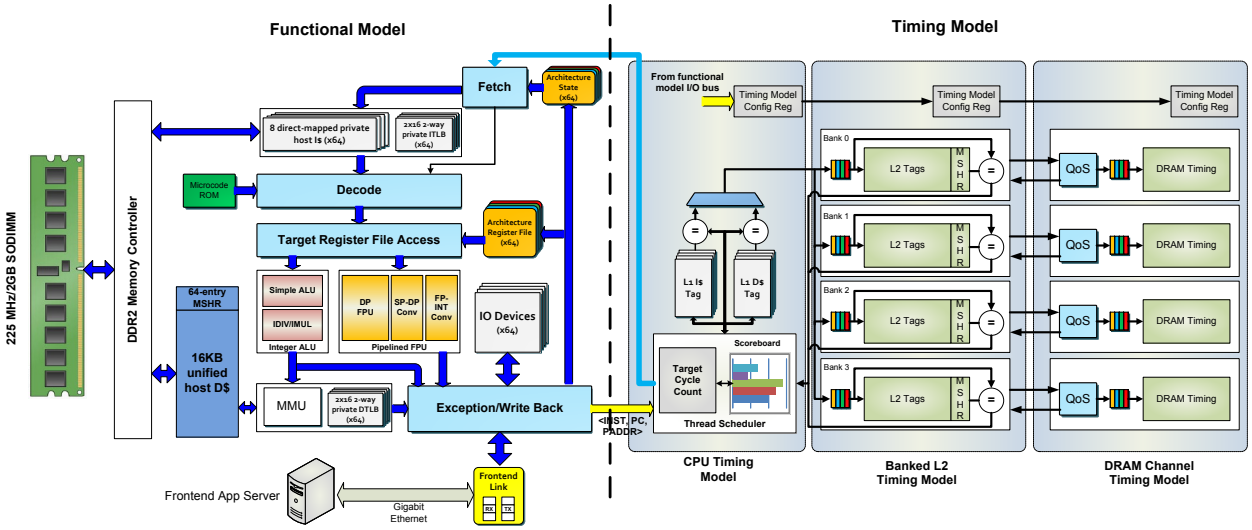

Figure 1: Midas Microarchitecture

Figure 1 shows the microarchitecture of Midas. The simulator decouples timing from function: the functional model faithfully executes the SPARC V8 ISA and maintains architected state, while the timing model determines instruction execution time in the target machine. Both models reside in one FPGA to minimize synchronization costs, and both are host-multithreaded to achieve high utilization of FPGA resources.

The functional model implements the full SPARC V8 ISA in hardware, including floating-point and precise exceptions. It also provides sufficient hardware to run an operating system, including MMUs, timers, and interprocessor interrupts. The functional model is deeply pipelined, and avoids features such as highly ported register files and wide bypass muxes that map poorly to FPGAs. The functional model carefully exploits Virtex-5 features, for example, double-clocking block RAMs and mapping target ALU instructions to hardwired DSP blocks. The single in-order issue functional pipeline is 64-way multithreaded, enabling functional simulation of 64 target cores. Each thread's private state includes a 7-window register file, a 32-entry instruction TLB and 32-entry data TLB, a 256-byte direct-mapped instruction cache, and the various processor state registers. Although 64 copies of this state seems large, trading state for increased pipeline utilization is attractive in the FPGA fabric, wherein storage is cheap relative to computation.

The threaded functional pipeline has a single, shared, lockup-free host data cache. It is 16 KB, direct-

mapped, and supports up to 64 outstanding misses. Sharing a very small host cache between 64 threads is a design point peculiar to the FPGA fabric: the low latency to the host DRAM, approximately 20 cycles in the worst case, is covered easily by multithreading. Thus, the lower miss rate of a large, associative host cache offers little simulation performance advantage. Indeed, across a subset of the PARSEC benchmarks, the small host cache incurs at most a 3.8% performance penalty compared to a perfect cache. (The tiny 256-byte instruction caches have even less of an impact on performance—at most 2.3%.)

The timing model tracks the performance of a 64-core tiled manycore system. The target processor core is currently a single-issue, in-order pipeline that sustains one instruction per clock, except for instruction and data cache misses. Each core has private instruction and data caches. The cores share a unified lockup-free L2 cache via a magic crossbar interconnect. Each L2 bank connects to a DRAM controller model, which models delay through a first-come, first-served queue with a constant service rate. Modeling the timing behavior of cache coherence traffic on realistic interconnects is a target for future work, though should fit easily within the current design framework.

Separating timing from function expands the domain of systems Midas can model, and allows the effort we expend on the functional model to be reused across many different target machines. For example, we can capture the timing of a system with large caches by keeping only the cache metadata on chip. The functional model still fetches from its host instruction cache and performs memory accesses to its host data cache when the timing model schedules it to do so. Moreover, the flexibility of splitting timing and function allows us to configure Midas' timing models at runtime. To model different cache sizes, for example, we fix the maximum cache size at synthesis time, and at runtime we program configuration registers that determine how the cache tag RAMs are indexed and masked. Most timing model parameters can be set at runtime; among these are the size and associativity of the L1 and L2 caches, the number of L2 cache banks and their latencies, and DRAM bandwidth and latency. The current implementation of the timing model runs at 90 MHz on the Virtex-5 FPGA. In this configuration, it supports up to 12 MB of total target caches, using over 90% of the on-chip FPGA block RAM resources.

4.2 Model Verification and Flexibility

Midas comprises about 36,000 lines of SystemVerilog with minimal third-party IP blocks. We liberally employ SystemVerilog assertions to aid in RTL debugging and verification. The functional model is verified against the SPARC V8 certification suite from SPARC International. The timing models are verified by our

in-house microbenchmarks. Although Midas does not solve the simulator timing verification problem that SAME simulators have, the greatly enhanced performance of FAME simulators eases the verification effort.

The timing model and its interface to the functional model are designed to be simple and extensible to facilitate rapid evaluation of alternative memory hierarchies and microarchitectures. Despite its extensive runtime configurability, the timing model comprises only 1000 lines of SystemVerilog. As an example of its flexibility, we implemented a simplified version of the Globally-Synchronized Frames [17] quality-of-service framework in about 100 lines of code and three hours of implementation effort.

Synthesis takes about two hours on a mid-range workstation with a 28% logic (LUT) and 90% BRAM utilization on a mid-size Virtex-5 FPGA. The low logic utilization comes in part from mapping computation to the built-in DSPs and by omitting bypass multiplexers. The high block RAM utilization comes from both the multithreaded design and the large target caches we support.

4.3 Related FAME Work

Midas is inspired in part by several recent works from the FAME community. Fort et al [15] employed multithreading to improve utilization of soft processors with little area cost. ProtoFlex [11] is an FPGA-based full-system simulator that employs host-multithreading to simulate multiple SPARC V9 cores with a single pipeline. Its primary purpose is to accelerate functional warming for a SAME sample-based simulator, so although it provides a functional cache model to speed up cache warming, it lacks a timing model. ProtoFlex also lacks a hardware floating-point unit, as it targets commercial workloads like OLTP; its performance thus suffers on arithmetic-intensive parallel programs, like those in the PARSEC suite. HAsim [12] is a FAME Level 011 simulator that similarly decouples target timing from functionality. FAST [10] is a hybrid FAME/SAME simulator whose functional model is in software and whose timing model is in an FPGA.

5 Evaluation: A Multicore OS Scheduling/HW Partitioning Case Study

To demonstrate the capabilities of Midas Multithreaded FAME, we experimented with the interaction between hardware partitioning mechanisms and operating system scheduling policy. In this case study, we run a modern parallel benchmark suite, boot a research operating system, simulate manycore target architectures with multi-level memory hierarchy timing models, and add experimental hardware mechanisms to the target machine. The conclusions we reached using the additional simulation capacity of FAME are different from

what we would have concluded using the limited simulation capacity of SAME.

5.1 Problem Statement

In the manycore era, general-purpose OS scheduling mechanisms will have to address a growing number of on-chip resources shared by concurrently running applications, which we call *spatial resource allocation*. Spatial resource allocation is challenging because applications can have diverse resource requirements, and the range of possible schedules grows combinatorially with applications and resources. Our proposal is to help the scheduler make intelligent decisions about spatial scheduling by leveraging *predictive models* of application performance. Model-based scheduling is only compelling if the decision space is large enough to justify the overhead of the technique, which is true for manycore machines with shared memory hierarchies. To make the case for model-based control of spatial resource scheduling, we need to learn if the performance models we create are accurate, which metrics the scheduler should use, and how far from optimal are the produced schedules. Evaluating each issue requires seconds of simulated target time and/or hundreds of experiments. We thus need a simulator that can run at OS timescales with reasonable turnaround.

Furthermore, we believe architectural support for performance isolation can greatly improve the effectiveness of predictive modeling and scheduling, as well as improve application performance by reducing interference. To test this theory, we added hardware partitioning mechanisms to the target architecture.

5.2 Partitioning Mechanisms

Our allocation framework includes the following resources: the cores and their private caches, the shared last-level cache, and shared memory bandwidth. For each resource, we provide a mechanism to prevent applications from exceeding their allocated share. The OS assigns cores and their associated private resources to a specific application. For the shared last-level cache, we modify the OS page-coloring algorithm so that applications are never given a page from a different application's color allocation. Colors map to specific areas of the cache and do not overlap. To partition off-chip memory bandwidth, we use Globally-Synchronized Frames (GSF)[17]. GSF provides strict Quality-of-Service guarantees for minimum bandwidth and the maximum delay of a point-to-point network—in our case the memory network—by controlling the number of packets that each core can inject per frame. We added hardware GSF mechanisms to Midas by by modifying its timing model.

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 FAME.

Name	Type	Parallelism	Working Set	Bandwidth Demand
Blackscholes	financial PDE solver	coarse data parallel	2.0 MB	minimal
Bodytrack	vision	medium data parallel	8.0 MB	grows with cores
Fluidanimate	animation	fine data parallel	64.0 MB	grows with cores
Streamcluster	data mining	medium data parallel	16.0 MB	high
Swaptions	financial simulation	coarse data parallel	0.5 MB	grows with cores
x264	media encoder	pipeline	16.0 MB	grows with cores
Tiny	synthetic	one thread does all work	1 KB	minimal
Greedy	synthetic	data parallel	16.0 MB	high

Table 3: Benchmark description. PARSEC benchmarks use `simlarge` input set sizes, except for x264 and fluidanimate, which use `simmedium` due to limited physical memory capacity. PARSEC characterizations are from [8].

5.3 Modeling Framework

To explore the relationship between model accuracy and type of model for our problem space, we evaluate linear additive models, quadratic response surface models, and non-linear models based on Kernel Canonical Correlation Analysis (KCCA) [6]. The OS scheduler uses the models for each application to decide an optimal resource allocation between a mix of applications all running concurrently. The algorithm works by maximizing an objective function, which serves to convert model outputs into a measure of overall decision fitness. The form of the objective function influences the type of algorithm we can use to maximize fitness. We evaluate three different objective functions. A robust evaluation of this framework requires full cross validation, which is computationally demanding.

5.4 Experimental Setup

Our experimental platform for this case study consists of five Xilinx XUP FPGA boards. Each board is programmed to simulate one instance of our target architecture. Table 2 lists the target machine parameters. We run six applications from the PARSEC benchmark suite [8], as well as two synthetic microbenchmarks. Table 3 lists the relevant benchmark qualities. The performance models are built based on applications running alone on a partition of the machine but are tested against data collected from multiprogrammed

scheduling scenarios. We simulated all possible allocations for each benchmark running alone and then all possible schedules of allocations for 3 pairs of benchmarks running simultaneously for a combined total of 68.5 trillion target core-cycles.

5.5 Case Study Results

We found that predictive-based modeling has potential to successfully manage some applications, depending on the scheduler's objective function. For example, if the objective is to minimize energy, the approach works quite well. However, if the objective is to minimize the time it takes to complete both applications, the naive baselines such as splitting the machine in half or time-multiplexing often performed as well or significantly better than model-based allocation. Figure 2(a) presents an example of these results. More importantly, our conclusions about the value of model-based scheduling would have been different had we *not* simulated the entire execution of benchmarks, with large input sets, for all possible allocations.

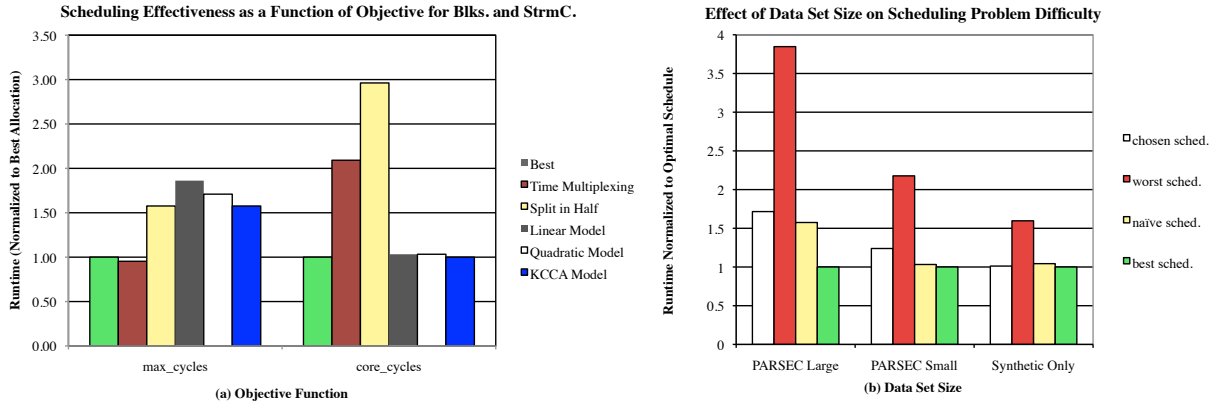

Figure 2: (a) The performance of various scheduling methodologies and objective functions for one scheduling problem, normalized to the globally optimal schedule's performance. The scheduler tries to minimized both objective functions. (b) The effect of benchmark size on the difficulty of the scheduling problem. The average chosen schedule performance, global worst case and naive scheduling case are normalized to the globally optimal schedule's performance for each dataset. The scheduling decision is Blackscholes vs. Streamcluster, the objective function is to minimize runtime.

Effect of Benchmark Input Set Size. We quantified the effect of benchmark configuration size by rerunning our experiments using either the PARSEC `simsmall` size input sets or only synthetic benchmarks. While we would expect different inputs or simpler benchmarks to produce slightly different results, Figure 2b reveals that these modifications significantly change the shape of the space of possible schedules. With both the small input set and synthetic benchmarks our scheduling technique is very close to optimal performance. However, for the larger input set the prediction-based schedule is 50% worse than optimal and

naively dividing the machine in half is always better. The reduced benchmarks make the space of possible decisions smaller and the process of making good decisions easier, giving us an unwarranted confidence in the scheduler's abilities.

Effect of Testing All Possible Allocations. Full cross validation of our scheduler requires collecting data on all possible schedules. We would otherwise have no idea what the optimal or pessimal scheduling decisions are, and thus no way to tell how well the scheduler is doing in comparison. The space of scheduling performance is complicated and discontinuous, so we are not guaranteed to capture true global optima if we validate by only testing against sample of possible schedules.

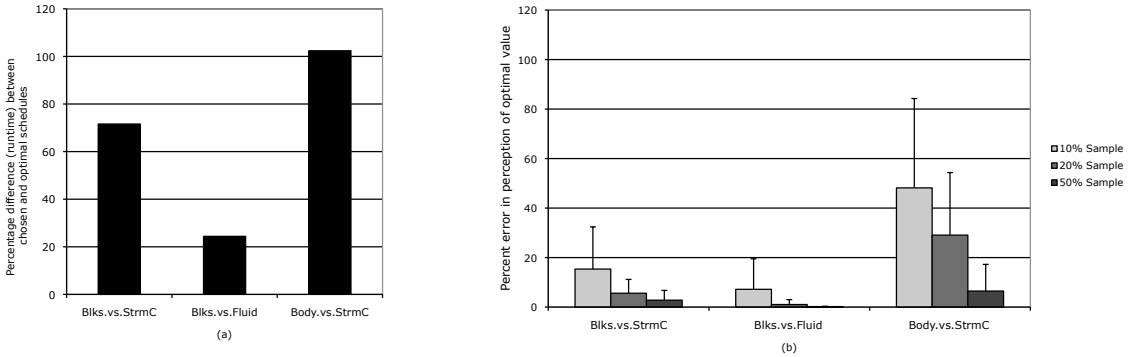

Figure 3: (a) The percentage difference in runtime between the average chosen schedule and the globally optimal schedule, for three scheduling problems. (b) The percentage error between the perceived optimal schedule for a given validation sample size and the true globally optimal schedule. Note that error in perception increases as sample size decreases, and can be a significant fraction of the percentage difference in runtime from (a).

Figure 3 illustrates how limited validation approaches may skew our interpretation of experimental results. Figure 3a plots the percentage difference in runtime between the average chosen schedule and the globally optimal schedule, for three scheduling problems. This difference illustrates how far our scheduler really is from finding the best schedule, but it can only be determined by evaluating the performance of all possible schedules.

In Figure 3b, we vary the amount of schedules examined during validation, covering different subsets of the total possible schedules. For each validation sample set size, we take many samples and report the percentage error in the average observed optima. The observed error varies because samples that are not inclusive of all possible schedules may miss optimal points, and smaller samples are more likely to exclude these important points. Note that the errors in Figure 3b can be a significant fraction of the true performance in Figure 3a for the smaller validation samples.

This data indicates that using reduced validation methods would have caused us to *overestimate* the quality of our scheduling solutions. As the perceived optimum degrades, our scheduler falsely appears to perform better. Only examining all possible schedules can give us an unbiased view of algorithmic and modeling quality. In conclusion, we could not have discovered the problem cases for our scheduling framework had we done a study with only reduced benchmarks or using only a small set of validation points. Gaining a complete understanding of scheduler performance in order to further improve it has only been made possible by FAME 111.

5.6 Simulator Speedup Results

To compare against Midas' performance, we run the PARSEC benchmarks inside Virtutech Simics [18], a popular SAME simulator. We run Simics with varying levels of architectural modeling detail: pure functional simulation, Simics `g-cache` timing modules, and the Multifacet GEMS [19] Ruby timing module.

We configure Simics to match the target machine simulated by Midas FAME in the case study as closely as possible. However, both `g-cache` and GEMS Ruby modules implement timing for a MESI coherence policy, whereas Midas FAME does not at present do so. The function/timing split within Midas leads us to believe that adding the timing information for MESI would have very little impact on simulation speed.

We vary the number of target machine cores simulated in both Midas and Simics. The applications spawn as many threads as the target machine has cores, but the workload size is fixed. Simics was run on 2.2GHz dual-socket dual-core AMD Opteron processors with 4GB of DRAM. Reducing the frequency at which Simics interleaved between different target processors offered a limited performance improvement.

The longest running Simics simulation takes over 192 hours (8 days), whereas the longest Midas FAME simulation takes 66 minutes. Figure 4 plots the wall clock runtime of a 64 core target machine simulated by Midas FAME and different Simics configurations across benchmarks and pairs of co-scheduled benchmarks. Midas is up to two orders of magnitude faster. Critically, this speedup allows the research feedback loop to be as short as minutes or tens of minutes, rather than tens or hundreds of hours.

Midas FAME runtimes generally improve as the number of cores is increased because multithreading becomes more effective, whereas Simics' performance degrades super-linearly with the number of cores simulated. With 64-core target machines, Midas FAME is even faster than Simics's functional simulation. Figure 5 shows the speedup of FAME over SAME for the different benchmarks and SAME configurations. The maximum speedup is a factor of 806×.

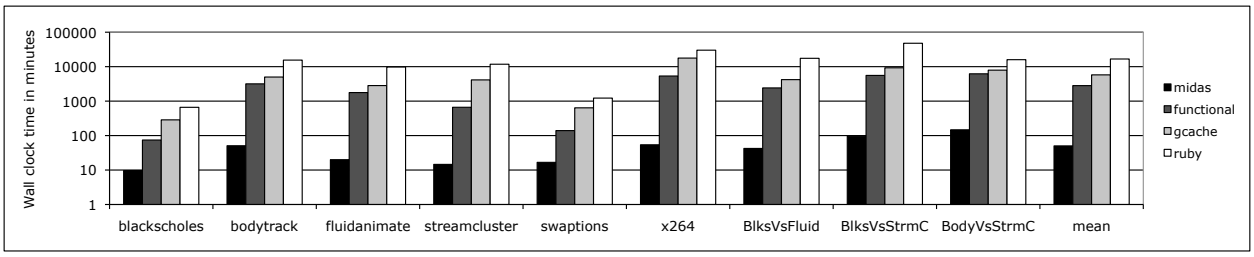

Figure 4: Wallclock time of FAME and SAME simulations. The target machine has 64 cores. Possible SAME configurations are functional modeling only, `g-cache` timing modules, and the GEMS Ruby module, with an interleave of 1 instruction. In the cases where two applications are run, each gets 1/2 of the partitionable hardware resources.

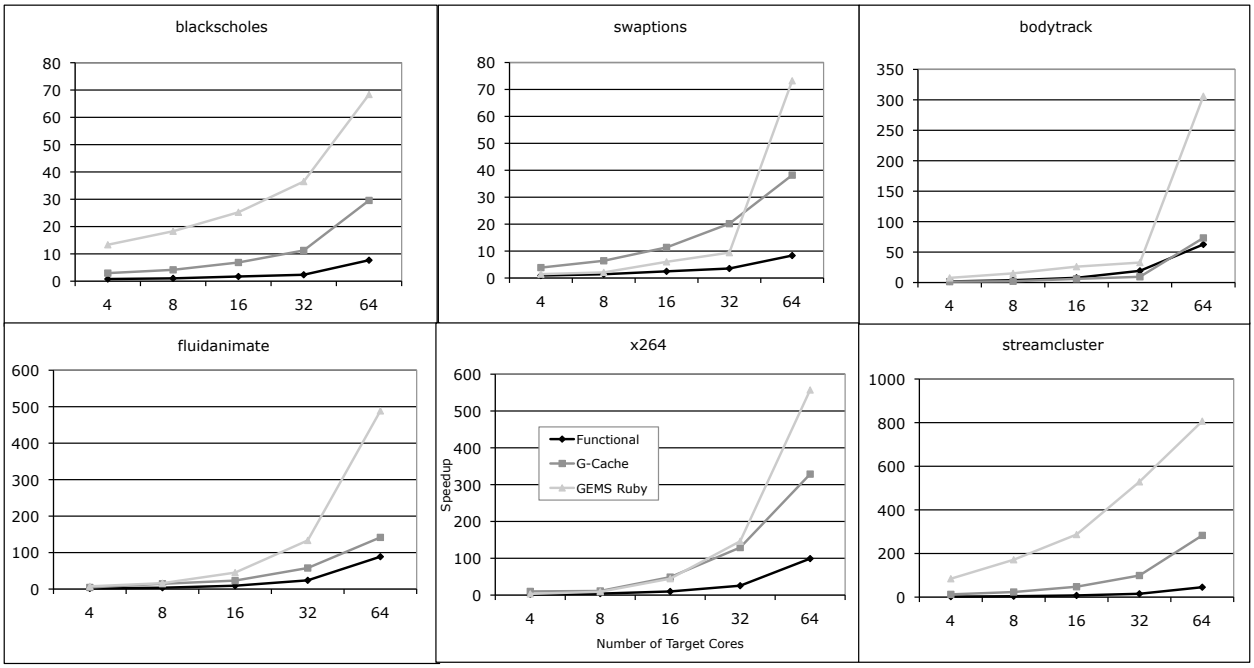

Figure 5: Speedup of FAME over SAME. Possible SAME configurations are functional modeling only, `g-cache` timing modules, and the GEMS Ruby module, with an interleave of 1 instruction.

The slowdowns incurred by Simics are due nearly entirely to host machine performance, as the benchmarks themselves scale in performance across more target cores equivalently on Ruby and Midas. The fact that the slowdowns also correlate with the size of the benchmarks' inputs and working set suggests that host cache and TLB misses may present a major performance bottleneck. Unfortunately, Simics is closed–source, so we were not able to diagnose its poor performance more precisely.

In terms of our case study, we calculate that just the verification of the scheduling decisions' optimality would have taken over 73,000 hours of wall clock time using Simics (8.3 years), versus 257 hours with Midas FAME. Table 4 includes estimates for the entire case study. These estimates are generous to SAME, as they do not include increased simulation time caused by decreased application performance due to limited

Case Study Component	FAME Total Runtime (hours)	Est. SAME Total Runtime (hours)	Median/Max FAME Latency (hours)	Median/Max SAME Latency (hours)
Model Sample Sets	49	2,848	0.5 / 2	44 / 196
Model Validation	145	16,150	0.5 / 2	44 / 196
Scheduling Validation	257	73,073	1 / 3	179 / 796
Misconfigured Experiments	82	8,612	1 / 3	179 / 796

Table 4: Comparison of case study simulation times with Midas FAME and Simics SAME. Total hours could be concurrently divided across multiple simulator instances. SAME times are estimated based on observed wall clock times for 2B target cycles, and assuming unrestricted target machine resource allocations

simulated resource allocation sizes. This application slowdown was around 2-5x for Midas. Ironically, we were not able to run Simics long enough to calculate the actual slowdown, which is why we use estimates based on a sample of 2 billion instructions.

While the extreme SAME simulation overhead could have been partially mitigated by using a cluster of machines, the latency of individual simulations would remain unchanged. In our experience, long latencies are harmful to experimenter productivity, as they necessitate waiting days or weeks (33 *days* worst case) for any feedback on which design decisions proved worthwhile, which studies should be done next, or even whether a new mechanism is implemented correctly. Trying to perform a case study like the one presented here, incorporating both novel software and hardware and operating over realistic stretches of simulated time, has grown completely untenable with SAME methods.

6 Conclusion

For many reasons, we believe the multicore revolution means the research community needs a boost in simulation performance. To clarify the many efforts at using FPGAs to deliver that boost, we propose a four-level nomenclature for FPGA Architecture Model Execution (FAME), inspired by the five levels of RAID. By estimating experiments per day per dollar, we show improvements in cost-performance by factors of 200,000 between the lowest and highest FAME levels. Midas, which simulates 64 SPARC CPUs on a $750 Xilinx Virtex5 board, demonstrates the benefits of the highest FAME level in a research case study of multicore HW/SW. The FAME boost in simulation time leads to different conclusions in this case study than had we been limited to fewer experiments with smaller input sets necessitated by the 250× decrease in simulation speed encountered when using a Software Architecture Model Execution (SAME) simulator.

References

[1] Incisive Enterprise Palladium Series with Incisive XE Software, http://www.cadence.com/rl/Resources/datasheets/incisive_enterprise_palladium.pdf.
[2] MicroBlaze Soft Processor Core, http://www.xilinx.com/tools/microblaze.htm, 2009.

[3] The Convey HC-1: The Worlds First Hybrid-Core Computer, http://www.conveycomputers.com/, 2009.

[4] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera Computer System. In *Proceedings of the 4th International Conference on Supercomputing*, pages 1–6, New York, NY, USA, 1990. ACM.

[5] K. Asanović, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[6] F. R. Bach and M. I. Jordan. Kernel Independent Component Analysis. *J. Mach. Learn. Res.*, 3:1–48, 2003.

[7] K. Barr. *Summarizing Multiprocessor Program Execution with Versatile, Microarchitecture-Independent Snapshots*. PhD thesis, MIT, Sept 2006.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81, New York, NY, USA, 2008. ACM.

[9] S. H. Bokhari, D. J. Mavriplis, and B. H. Elton. The Cray MTA and Unstructured Meshes. Sept 2000.

[10] 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 *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 249–261, Washington, DC, USA, 2007. IEEE Computer Society.

[11] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, K. Mai, and B. Falsafi. ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 2(2):1–32, 2009.

[12] N. Dave, M. Pellauer, Arvind, and J. Emer. Implementing a Functional/Timing Partitioned Microprocessor Simulator with an FPGA, Feb. 2006.

[13] J. Davis, C. Thacker, and C. Chang. BEE3: Revitalizing Computer Architecture Research. Technical Report MSR-TR-2009-45, Microsoft Research, Apr 2009.

[14] L. L. Dick, J.-T. Li, T. B. Huang, and S. K. C. Kenneth. US Patent 5425036 - Method and apparatus for debugging reconfigurable emulation systems, http://www.patentstorm.us/patents/5425036.html.

[15] B. Fort, D. Capalija, Z. G. Vranesic, and S. D. Brown. A multithreaded soft processor for sopc area reduction. In *FCCM '06: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 131–142, Washington, DC, USA, 2006.

[16] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. RAMP Blue: A Message-Passing Manycore System In FPGAs. In *Proceedings of International Conference on Field Programmable Logic and Applications*, pages 54–61, Amsterdam, The Netherlands, 2007.

[17] J. W. Lee, M. C. Ng, and K. Asanović. Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 89–100, Washington, DC, USA, 2008.

[18] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, 35, 2002.

[19] 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 Computer Architecture News*, 33(4):92–99, 2005.

[20] J. E. Miller, H. Kasture, G. Kurian, C. G. III, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A Distributed Parallel Simulator for Multicores. In *HPCA-16: Proceedings of the 16th IEEE International Symposium on High-Performance Computer Architecture*, January 2010.

[21] N. Njoroge, J. Casper, S. Wee, T. Yuriy, D. Ge, C. Kozyrakis, , and K. Olukotun. ATLAS: A Chip-Multiprocessor with Transactional Memory Support. In *Proceedings of the Conference on Design Automation and Test in Europe (DATE), Nice, France, April 2007*, pages 1–6, 2007.

[22] V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM Reference Manual. Version 1.0. Technical Report 9705, Department of Electrical and Computer Engineering, Rice University, July 1997.

[23] M. Pellauer, M. Adler, D. Chiou, and J. Emer. Soft Connections: Addressing the Hardware-Design Modularity Problem. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 276–281, New York, NY, USA, 2009. ACM.

[24] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood. The Wisconsin Wind Tunnel: virtual prototyping of parallel computers. *SIGMETRICS Perform. Eval. Rev.*, 21(1):48–60, 1993.

[25] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod. Using the SimOS machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Simulation*, 7(1):78–103, 1997.

[26] J. Shalf. private communication, June 2009.

[27] G. E. Suh, C. W. O'Donnell, and S. Devadas. Aegis: A Single-Chip Secure Processor. *IEEE Design and Test of Computers*, 24(6):570–580, 2007.

[28] S. Swanson, A. Putnam, M. Mercaldi, K. Michelson, A. Petersen, A. Schwerin, M. Oskin, and S. J. Eggers. Area-Performance Trade-offs in Tiled Dataflow Architectures. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, pages 314–326, Washington, DC, USA, 2006. IEEE Computer Society.

[29] C. Thacker. private communication, May 2009.

[30] J. Wawrzynek, D. A. Patterson, M. Oskin, S.-L. Lu, C. E. Kozyrakis, J. C. Hoe, D. Chiou, and K. Asanović. RAMP: Research Accelerator for Multiple Processors. *IEEE Micro*, 27(2):46–57, 2007.

[31] M. Wehner, L. Oliker, and J. Shalf. Towards Ultra-High Resolution Models of Climate and Weather. *International Journal of High Performance Computing Applications*, 22(2):149–165, 2008.

[32] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro*, 26(4):18–31, 2006.

[33] E. Witchel and M. Rosenblum. Embra: Fast and Flexible Machine Simulation. *SIGMETRICS Perform. Eval. Rev.*, 24(1):68–79, 1996.