A Case for FAME: FPGA Architecture Model Execution

Zhangxi Tan, Andrew Waterman, Henry Cook, Sarah Bird, Krste Asanović, David Patterson

The Parallel Computing Laboratory
CS Division, EECS Department, University of California, Berkeley

{xtan,waterman,hcook,slbird,krste,pattrsn}@eecs.berkeley.edu

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, RAMP Gold, to research the interaction between hardware partitioning mechanisms and operating system scheduling policy. The study demonstrates FAME's 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×—enables experiments with more realistic time scales and data set sizes than are possible with SAME.

Categories and Subject Descriptors

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

General Terms

Design, Performance, Experimentation

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, 19], however, has both increased the complexity of the systems architects want to simulate and removed the straightforward path to simulator performance scaling. Parallel applications exhibit more complex behaviors than sequential ones, including timing-dependent non-deterministic execution, cache coherence traffic, and operating system scheduler interactions. Applications now also commonly have dynamically generated code, including code that is automatically tuned to the underlying hardware platform. Although it is generally well understood how to model these 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, however, those instructions were simulated across an average of 16 times as many processors, so the number of instructions per processor significantly decreased over that decade. To tackle this *simulation gap*, several research groups, including those in the RAMP Project [41], 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 then present the detailed design of the RAMP Gold simulator, a very efficient architectural simulator for early-stage design exploration. We describe how RAMP Gold supports various forms of architecture experimentation and

ISCA'10, June 19–23, 2010, Saint-Malo, France.

how modifications could be made to support others. The following section presents a case study to illustrate the effectiveness of RAMP Gold in modern systems research. We also provide a quantitative evaluation of the RAMP Gold simulator against Simics+GEMS [24, 25], a popular SAME simulator. We compare simulation speeds for the PARSEC benchmark suite [7]. We show that while pure functional simulations of a few cores run at about the same speed, RAMP Gold runs detailed models on 64 cores on average 263× faster than Simics+GEMS, with a maximum speedup of 806×. Not only does this provide a dramatic increase in simulation capacity, it also markedly reduces simulation latency thereby improving experimenter productivity.

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. In this paper, we refer to the machine being simulated as the *target* and the machine on which the simulation runs the *host*.

Simulating parallel target machines is much more difficult 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. Also, a parallel software runtime must be present to support multithreading or multiprogramming across the multiple cores of the simulated target, which adds additional complexity. 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 model user-level activity of a single application. One popular early example was RSIM [29], which provided detailed models of out-of-order superscalar processors connected via coherent shared memory but without support for an operating system. The SimOS project demonstrated how to run an operating system on top of a fast software simulator [34]. SimOS supported multiple levels of simulation detail, with the fastest version of SimOS using dynamic binary translation to speed target instruction emulation while emulating cache hierarchies in some detail [44]. A similar technique was incorporated into the commercial product Simics, which allows researchers to study large application programs and the operating system running together. Although the architecture research community is not a large market, Simics offers an interface that allows researchers to plug in their own detailed execution-driven performance model to evaluate detailed portions of the computer [24]. Augmented with detailed performance models developed by other researchers [25], Simics has become a popular tool in the architecture community; we provide a detailed evaluation of its performance in Section 5.

Sadly, parallelizing detailed multiprocessor simulations to run efficiently on parallel host machines is difficult. The need for cycle-by-cycle interaction between components limits the parallel speedup 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 [26, 33].

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 [6]. For uniprocessors, the program execution should be the same independent of 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, an experiment exploring the impact of relaxed consistency models on multithreaded programs might never encounter 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 [43].

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 be measured without understanding the problems being solved or the algorithms and data structures being used. Thus, collections of old software like SPEC were considered reasonable benchmarks for future architectures, as long as the 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 applications, new programming models, and new programming languages have been largely ignored. The move to ubiquitous parallel processing is such a disruptive change that this *laissez faire* approach will no longer work, since by definition we need new programs, models, and languages. Moreover, as mentioned above, modern software dynamically adapts to the hardware microarchitecture by generating and measuring code at runtime. Existing simulation time shortcuts are not likely to be useful, and simulation latency is now at least as important as simulation throughput.

3. FPGA-BASED SIMULATION

FPGAs have become a promising vehicle for architecture experiments, providing a highly parallel programmable execution substrate that can run simulations several orders of magnitude faster than software [41]. Previously, FPGA processor models were hampered by the need to partition a processor across multiple FPGAs, which increases hardware costs, reduces performance, and significantly increases development effort. But FPGA capacity has been scaling with Moore's Law and so now, depending on complexity, multiple processors can fit on 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 RAMP Gold design, for example, uses a $750 commercial board.

Multiple groups have now developed working FPGA-

based systems [11, 14, 28, 21, 12], but using perhaps an even greater variety of techniques than software simulators, and correspondingly a wider range 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 [16] 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. 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 [18]. For example, direct mapping of a multiported 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 [18, 31].

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 [30] was an early example of the *Abstract* FAME option, where the processor model was 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 to change to experiment with different microarchitectures, and the timing model can include parameters such as cache size and associativity that can be set at runtime 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 experience long host latencies, for example, to communicate with off-chip memory or other FPGAs, reducing simulator performance. For processors, a standard approach to tolerate latencies and obtain greater performance is to switch threads every clock cycle so that all dependencies are resolved by the next time a thread is executed [3]. The same approach can be applied when implementing FAME models in a technique we call *host multithreading*, and is particularly applicable to models of parallel target machines.

When the target system contains multiple instances of the same component, such as 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 or 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 makes 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 [12]. The same concept has also been used in SAME simulators, *e.g.*, later versions of the Wisconsin Wind Tunnel were also host multithreaded [27].

3.2 FAME Levels

A combination of these FAME implementation techniques

Level	*Name*	*Example*	*Strength*	*Experiments per Day*	*Experiments per Day per $1000*
000	Direct FAME	Quickturn, Palladium	Debugging logical design	1	0.001
001	Decoupled FAME	Green Flash	Higher clock rate; lower cost	24	0.667
011	Abstract FAME	HAsim	Simpler, parameterizable design; faster synthesis; lower cost	40	40.000
111	Multi-threaded FAME	RAMP Gold	Lower cost; higher clock rate	128	170.000

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

often makes sense. 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 target 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, experiment setup time (*e.g.* design synthesis time) must also be included. To obtain 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 2010 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 [39]. 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 superior to 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 [42] can fit two Tensilica cores with floating-point units per medium-sized FPGA, but it runs at only 50 MHz [35]. 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 [15] 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 40 seconds (2 GHz/50 MHz) 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 [35]. 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/$36K 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 reasonable area, power and timing numbers from actual chip layout [37]. 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 imple-

ment 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-performance.

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 [32], and an average number of host cycles per target cycle of 4. The time for one experiment is then $4 \times 2\,\text{GHz}/65\,\text{MHz} = 123$ 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/$18K 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 improvement 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., RAMP Gold)

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 logic unit can be exchanged for 64-bits of RAM in a Virtex-5 FPGA. 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 RAMP Gold's host cycles per target core-cycle to 1.90 (measured) and enables a clock rate of 90 MHz. Since the simulator is threaded, the time for a 16-core simulation is $16 \times 2\,\text{GHz}/90\,\text{MHz} \times 1.9 = 675$ seconds. The number of experiments per day is 24 hours / 675 seconds = 128. The number of experiments per day per $1000 is then 128/$0.75K 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 Version 2 of RAMP Gold 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 cost of entry by a factor of 24–48 versus Abstract or Decoupled FAME, making it possible for many more researchers to use FAME for 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 often combine modules at different levels, or even use hybrid designs partly in FPGA and the rest in software. For example, System-on-a-Chip IP providers will often use a mixed design to provide a fast *in situ* emulation of their IP blocks for customers. The IP block is modeled by mapping the final ASIC RTL to the FPGA (Direct FAME, Level 000), but the enclosing system is described at an abstract level (Abstract FAME, Level 011). FAST [11] is an example of a hybrid FAME/SAME system, where the functional model is in software and the timing model is in hardware.

3.2.7 FPGA Computers

Another use of FPGAs is to build *FPGA computers*, where FPGAs are the final target technology. For example, FPGAs are used to build computers to run real production compute workloads (e.g., Xilinx Microblaze [2], Convey HC-1 [9]). Research processors can also be prototyped using FPGAs at much lower cost, risk, and design effort compared to a custom chip implementation [36, 28, 21, 40]. Although an FPGA research prototype bears little resemblance to a custom chip in terms of cycle time, area, or power, it can yield valuable insights into the detailed implementation of a new architectural mechanism, as well as provide a fast platform to evaluate software interactions with the new mechanisms.

Although often confused with Direct FAME (level 000), FPGA computers are not intended to model a target machine; rather, they *are* the target machine. As such, we do not include FPGA computers in the FAME taxonomy.

4. THE RAMP GOLD SIMULATOR

RAMP Gold [1] 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 RAMP Gold and how it attains high efficiency by tailoring the design to the FPGA environment.

The current RAMP Gold simulated target is a tiled, shared-memory manycore system with up to 64 single-issue, in-order SPARC V8 cores. RAMP Gold is a full-system simulator, which boots the Linux 2.6 kernel, as well as ROS [20, 23], our manycore research operating system. RAMP Gold's target machine is highly parameterized and 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 accelerates design space exploration and comes

[1]RAMP Gold source code is available at http://ramp.eecs.berkeley.edu/gold

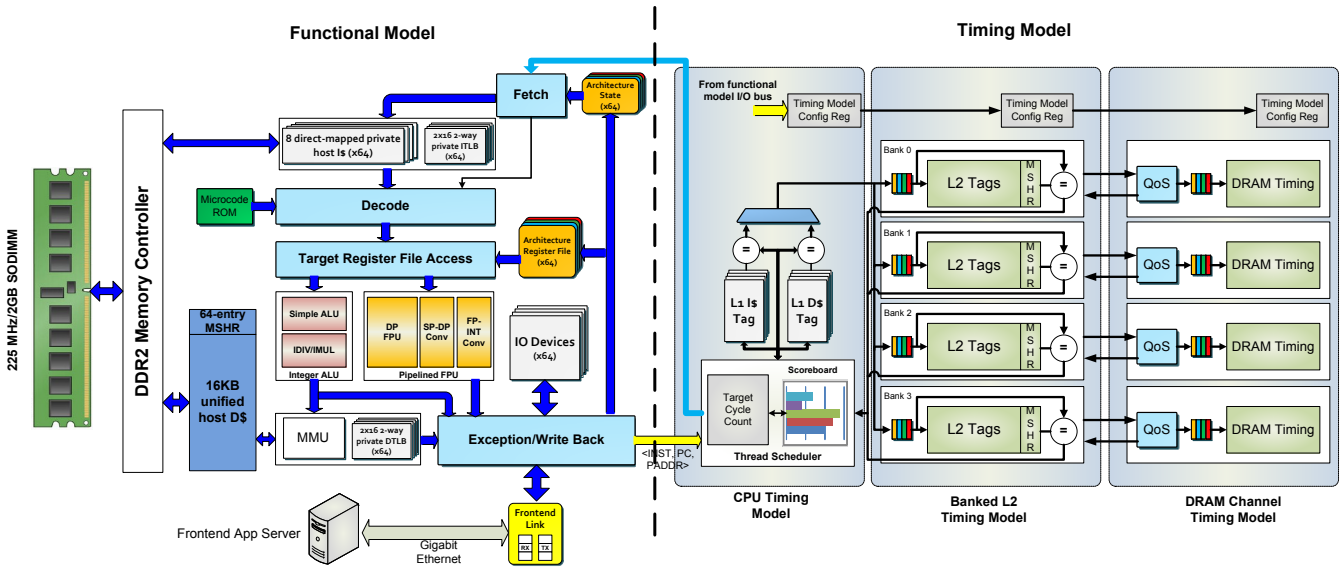

Figure 1: RAMP Gold Microarchitecture

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 RAMP Gold Microarchitecture

Figure 1 shows the microarchitecture of RAMP Gold. 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% worse than a perfect cache.)

The timing model tracks the performance of a target 64-core tiled manycore system. The target 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 range of systems RAMP Gold can model, and allows the effort expended on the functional model to be reused across many different target machines. For example, we can model the performance of a system with large caches by keeping only the cache metadata inside the FPGA. 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 RAMP Gold's 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, and supports up to 12 MB of aggregate target cache, while using over 90% of the on-chip FPGA block RAM resources.

Synthesis of RAMP Gold takes about two hours on a mid-

range workstation, resulting in 28% logic (LUT) and 90% BRAM utilization on a mid-size Virtex-5 FPGA. The low logic utilization is due in part to mapping computation to the built-in DSPs and by omitting bypass multiplexers. The high block RAM utilization is mainly due to the large target caches we support. More details on the RAMP Gold implementation can be found in [38].

4.2 Model Verification and Flexibility

RAMP Gold 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. Because it uses abstracted RTL, RAMP Gold requires the same simulator timing verification as SAME simulators, but the far greater performance eases the verification effort. We verify our timing models with custom microbenchmarks.

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. It is thus easy to understand and prototype new architectural ideas. For example, we implemented a version of a novel quality-of-service framework, Globally-Synchronized Frames [22], in about 100 lines of code and three hours of implementation effort.

4.3 Related FAME Work

RAMP Gold is inspired in part by other recent projects in the FAME community. Fort et al. [17] employed multithreading to improve utilization of soft processors with little area cost. Protoflex [12] is an FPGA-based full-system simulator without a timing model, and is designed to provide similar functionality to Simics [24] 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 [31] is a FAME Level 011 simulator that decouples target timing from functionality, but models a more complex out-of-order superscalar processor. A newer version of HAsim with host multithreading is under development. FAST [11, 10] is a hybrid FAME/SAME simulator which uses a speculative execution scheme to allow a software functional model to efficiently run in parallel with a decoupled timing model in an FPGA. RAMP Gold is the first multithreaded, decoupled FAME Level 111 simulator with a detailed multi-level memory hierarchy timing model.

5. EVALUATION: A MANYCORE OS SCHEDULING CASE STUDY

To demonstrate the capabilities of Multithreaded FAME, we experiment with the interaction between hardware partitioning mechanisms and operating system scheduling policy using RAMP Gold. We chose this case study [8, 13] because we feel it contains many important components of modern architecture research: 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 additional simulation capacity of RAMP Gold enables us to move beyond simple benchmarking, and instead experiment with different operating system policies to see if the OS can effectively utilize these added mechanisms. We feel that our results illustrate the potential of FAME not only for parallel computer architecture research but also as an enabler of hardware and system software co-design.

The speedup in simulation time from FAME allows for a far more robust analysis than would have been possible using SAME, and the conclusions reached using the additional simulation capacity are qualitatively different from those that would be obtained using the slower SAME simulation capability.

5.1 Problem Statement

In the manycore era, operating systems face the *spatial resource allocation* problem of having to manage a growing number of on-chip resources shared by a growing number of concurrently running applications. 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 to help the scheduler make intelligent spatial scheduling decisions is to employ *predictive models* of application performance. The OS scheduler uses the model of each application's performance to predict which schedule will have the best overall system behavior (i.e., best performance or lowest energy). 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 resulting 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.

To partition off-chip memory bandwidth, we use Globally-Synchronized Frames (GSF)[22]. 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 use a modified version of the original GSF design, which tracks allocations per application instead of per core, does not reclaim frames early, and does not allow applications use the excess band-

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: Target machine parameters simulated by RAMP Gold.

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 [7].

width. These changes make GSF more suited to our study since we want to strictly bound the maximum bandwidth per application. Implementing GSF required some modifications to the target machine's memory controller in RAMP Gold to synchronize the frames and track application packet injections. Due to the functional/timing split in RAMP Gold, this modification was no more difficult than modifying a software simulator would have been.

5.3 Modeling and Scheduling Framework

To explore the relationship between model accuracy and model type for our problem space, we evaluate linear additive models, quadratic response surface models, and non-linear models. The OS scheduler uses the models of each application to optimally allocate resources between a mix of concurrently running applications. The algorithm maximizes an objective function, which serves to convert model outputs into a measure of overall decision fitness. We evaluate three objective functions. A robust evaluation of this framework requires full cross validation of possible alternate schedules, 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 [7], as well as two synthetic microbenchmarks. Table 3 summarizes the benchmarks. 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. (A core-cycle is 1 clock cycle of execution on 1 core; simulating a 64-core CMP for 1,000,000 cycles would be 64,000,000 core-cycles.)

5.5 Case Study Results

We found that predictive 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.

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 conclusion. For this particular scheduling problem, 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 improve the worst case while also easing the process of making good decisions, 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 a sample of possible schedules.

Figure 3 illustrates how limited validation approaches may skew our interpretation of experimental results. Figure 3a

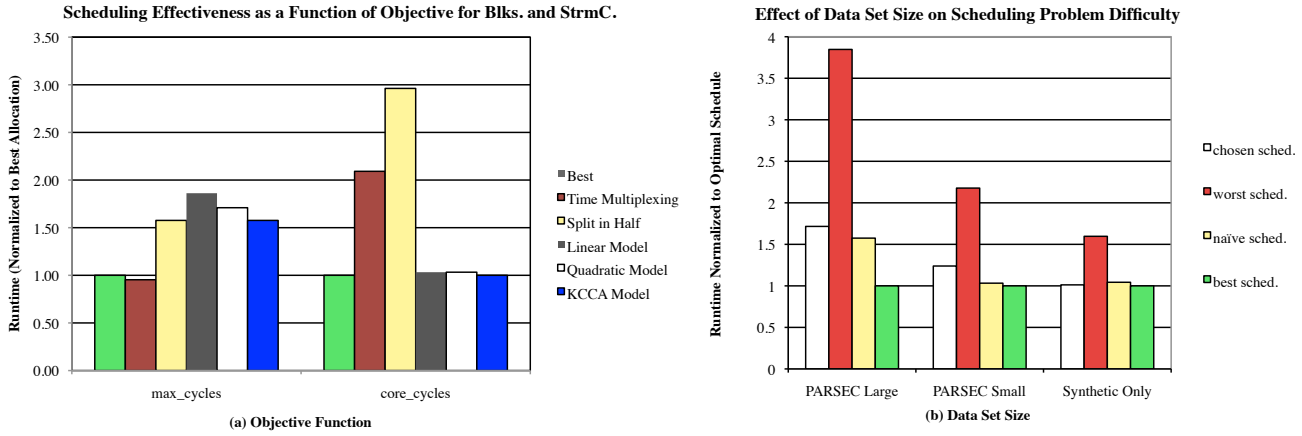

Figure 2: (a) The performance of various scheduling methodologies and objective functions for one scheduling problem, normalized to the globally optimal schedule's performance. The scheduler tries to minimize each objective function. (b) The effect of benchmark size on the difficulty of the scheduling 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.

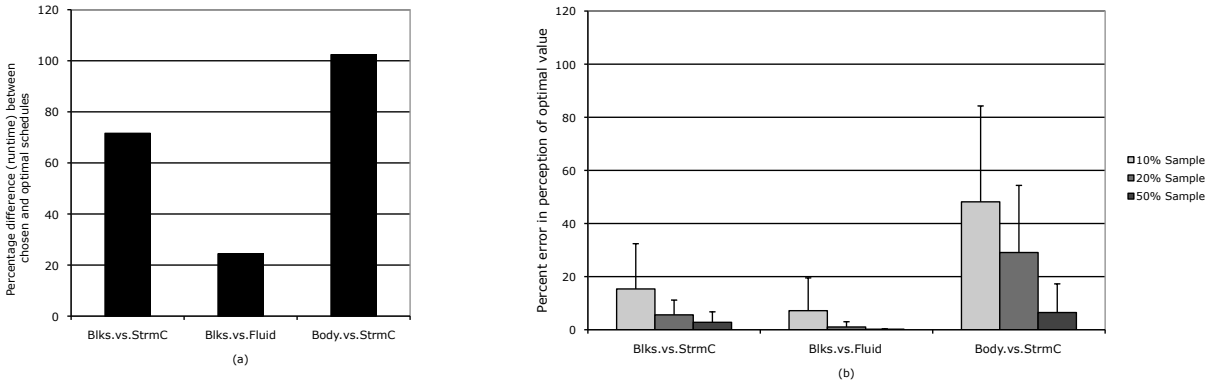

Figure 3: (a) The percentage difference in runtime between the average chosen schedule and the globally optimal schedule, for three scheduling problems. (b) The percentage error between the perceived optimal schedule for a given validation sample size and the true globally optimal schedule. Note that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

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 number of schedules examined during validation, covering different subsets of the total possible schedules. For each validation sample set size, we take many different 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 gave 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 more complete understanding of scheduler performance with new hardware was only possible by using FAME.

5.6 Simulator Speedup Results

To compare against RAMP Gold's performance, we run the PARSEC benchmarks inside Virtutech Simics [24], 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 [25] Ruby timing module.

We configure Simics to model the same case study target machine as closely as possible. However, both `g-cache` and GEMS Ruby modules implement timing for a MESI coher-

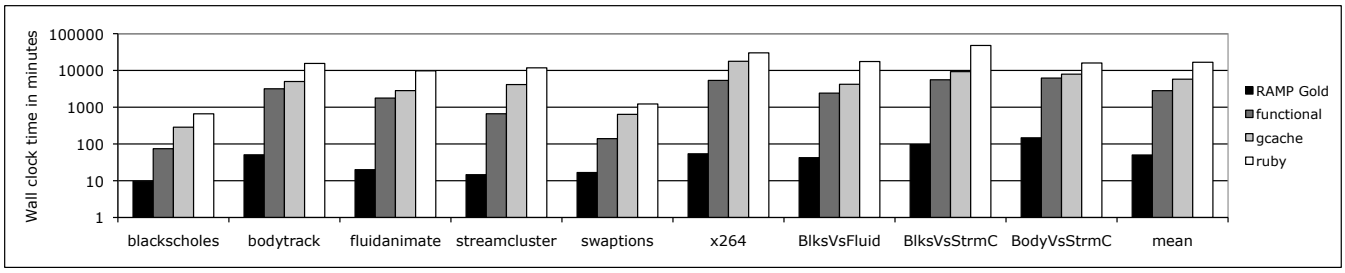

Figure 4: Wallclock time of RAMP Gold and Simics simulations. The target machine has 64 cores. Possible Simics configurations are functional modeling only, `g-cache` timing modules, and the GEMS Ruby module, with an interleave of 1 instruction. In the cases where two applications are run, each gets 1/2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

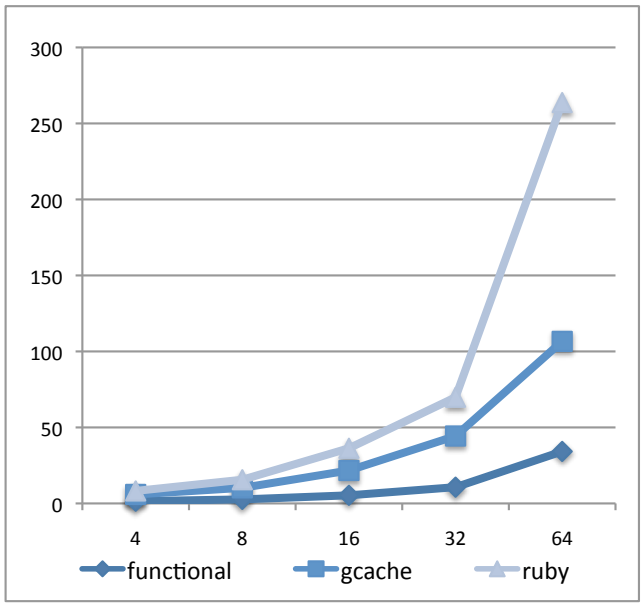

Figure 5: Geometric mean speedup of RAMP Gold over Simics across benchmarks. Possible Simics configurations are functional modeling only, `g-cache` timing modules, and the GEMS Ruby module, with an interleave of 1 instruction. The x-axis is target cores.

ence policy, whereas RAMP Gold does not at present do so. We configure Ruby to not simulate contention in the on-chip interconnection network (neither `g-cache` nor RAMP Gold do so presently). We are in the process of developing coherence timing models for RAMP Gold, and we believe these would have very little impact on simulation speed as they would run in parallel to the functional model.

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

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

RAMP Gold 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, RAMP Gold is even faster than Simics's functional simulation. Figure 5 shows the geometric mean speedup of FAME over SAME across the different benchmarks and for different SAME configurations. The maximum speedup is a factor of 806×.

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 Simics and RAMP Gold. 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 (8.3 years) of wall clock time using Simics versus 257 hours with RAMP Gold. 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 simulated resource allocation sizes. This application slowdown was around 2–5× for RAMP Gold. 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 require waiting days or weeks (33 *days* worst case) for feedback on which design decisions proved worthwhile, which studies should be done next, or even whether a new mechanism is implemented correctly. We believe performing a case study as presented here, incorporating both novel software and hardware, and operating over realistic stretches of simulated time, would be completely untenable with SAME methods.

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 RAMP Gold (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.

6. DISCUSSION

In our conversations with the broader architecture community, the use of FPGA-based simulation technology raises several major concerns.

The larger development effort of FAME versus SAME is perhaps the biggest valid concern. Simulator developers need to learn a hardware description language and FPGA CAD tools, and develop a highly concurrent hardware simulation model. In our experience, the inherent complexity of FAME simulators is somewhat greater than SAME simulators, but not vastly so. In any case, few researchers build simulators from scratch, and so the effort to modify an existing simulator is a more important measure. For many experiments, only the timing model needs to be changed, and the entire timing model for RAMP Gold is only around 1000 lines of System Verilog code. The RAMP Gold functional model is around 35,000 lines of System Verilog code, but much new functionality can be added by simply changing the microcode in the existing pipeline. The massively improved simulator performance is itself a significant boost to developer productivity, as many tasks (verification, software porting, calibration) require many simulation cycles. A much bigger problem than simulator complexity for developer productivity is the poor state of FPGA CAD tools, which are even worse than ASIC CAD tools. During the 2-year development of RAMP Gold, we encountered 78 formally tracked bugs in four tools from two companies, ranging from logic synthesis and simulation to formal verification. We can only hope the tools will improve in quality.

The cost of FPGA boards is another common concern, but when Multithreaded FAME techniques are used, the FPGA hardware is far more cost-effective than conventional clusters. For example, consider what our experiments would have cost had we run them in the cloud, which would be far cheaper than maintaining a large private cluster for most institutions. The five XUP boards we used were attached to a single workstation and cost $3750 and took 257 total hours (11 XUP days) to run our case study, with a maximum latency of 3 hours for one experiment. Had we run the experiments on Amazon EC2/S3, we estimate the cost would have been about $12,500 for the 73,000 hours (3000 CPU days) it would take to run the case study[2]. The cloud run costs a little over a factor of three more than our initial investment, yet this is only for one set of experiments. We can continue to use the XUP boards for our architecture research, amortizing this capital investment over 2–3 years and hundreds of research experiments. Additionally, the latency of individual experiments is not helped using a cluster, which means many of our simulations would still take longer than 33 days on EC2. FAME techniques dramatically *lower* the cost of large-scale simulations compared to SAME.

Improving the performance of SAME simulation via sampling is sometimes proposed as an alternative to FAME simulation. But serious questions remain for the practical application of sampling for many multiprocessor experiments. First, to greatly reduce simulation time, the sampling system has to support flexible microarchitecture-independent snapshots, which remain an open research problem for multiprocessors [6, 43]. Without snapshots, simulation time is dominated by the functional warming needed to fast-forward between sample points (ProtoFlex was developed, in part, to accelerate functional warming [12]). Second, for parallel systems, program behavior depends on microarchitectural details, for example, due to operating system scheduling, lock contention, or dynamic code adaptation, so detailed timing models are needed between sample points, reducing the speedup from sampling. Finally, it is not clear that fast, accurate sampling simulators are actually easier to develop and modify than a brute-force FAME simulator.

Obviously, not all architecture research projects need FAME simulators. For example, SAME simulators likely work well enough when trying to improve branch predictors or pipeline designs of small-scale multiprocessors. For experiments that do not require accurate timing models or many cores, SAME can be accelerated through the use of dynamic binary translation. However, we believe that many current computer architecture research challenges concern large core counts, detailed microarchitecture models, significant time durations, non-deterministic behavior, and dynamic software, in which case SAME is no longer tractable.

7. CONCLUSION

For many reasons, we believe the move to parallel computing 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 taxonomy for FPGA Architecture Model Execution (FAME). 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. We use a research case study using RAMP Gold, which simulates 64 SPARC CPUs on a $750 Xilinx Virtex5 board, to demonstrate the benefits of the highest FAME level. The 250× decrease in simulation time versus a Software Architecture Model Execution (SAME) simulator led to vastly different conclusions than had we been constrained by slower SAME simulation to fewer experiments with smaller input sets. We believe that the dramatic improvement in simulation latency

[2]We assume Amazon's High-CPU Medium Instances [4], which most closely match the machines on which we performed our SAME experiments; each instance costs $0.17 per hour.

and throughput offered by FAME simulators opens the door for exciting research in computer architecture and hardware-software co-design.

ACKNOWLEDGEMENTS

We'd like to thank the many members of the RAMP community for their contributions to the various levels of the FAME methodology over the last five years. We especially thank SPARC International, Inc. for donating the SPARC v8 verification suite. Thanks to the ROS implementers for their assistance on the RAMP Gold port, and to Kevin Klues in particular for providing page coloring support to facilitate our case study. We'd also like to thank Doug Burger, Derek Chiou, John Davis, Joel Emer, Mark Hill, David Wood, James Hoe, Chuck Thacker, Kees Vissers, and the anonymous reviewers for their insightful feedback on drafts of this paper.

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 donation of FPGAs and development tools. We also appreciate the financial support provided by the Gigascale Systems Research Center (GSRC).

8. 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] R. Alverson et al. The Tera Computer System. In *Proc. of the 4th Int'l Conference on Supercomputing*, 1990.

[4] Amazon Inc. *Amazon Elastic Compute Cloud (Amazon EC2)*. Amazon Inc., http://aws.amazon.com/ec2/#pricing, 2010.

[5] K. Asanović et al. A view of the parallel computing landscape. *Commun. ACM*, 52(10):56–67, 2009.

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

[7] C. Bienia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of the 17th Int'l Conference on Parallel Architectures and Compilation Techniques*, 2008.

[8] S. Bird. Software Knows Best: A Case for Hardware Transparency and Measurability. Master's thesis, EECS Department, University of California, Berkeley, May 2010.

[9] T. Brewer. Instruction set innovations for the Convey HC-1 computer. *IEEE Micro*, March/April 2010.

[10] D. Chiou, H. Angepat, N. P. Patil, and D. Sunwoo. Accurate Functional-First Multicore Simulators. *Computer Architecture Letters*, 8(2), July 2009.

[11] D. Chiou et al. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In *MICRO*, 2007.

[12] E. Chung et al. ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs. *ACM Trans. on Reconfigurable Technology and Systems*, 2009.

[13] J. Colmenares et al. Resource Management in the Tessellation Manycore OS. In *HotPar10*, Berkeley, CA, June 2010.

[14] N. Dave et al. Implementing a functional/timing partitioned microprocessor simulator with an FPGA. In *Proc. of the Workshop on Architecture Research using FPGA Platforms*, held at HPCA-12, Feb. 2006.

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

[16] L. L. Dick et al. US Patent 5425036 - Method and apparatus for debugging reconfigurable emulation systems, http://www.patentstorm.us/patents/5425036.html.

[17] B. Fort et al. A Multithreaded Soft Processor for SoPC Area Reduction. In *FCCM '06: Proc. of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

[18] G. Gibeling, A. Schultz, and K. Asanović. RAMP architecture and description language. In *2nd Workshop on Architecture Research using FPGA Platforms*, February 2006.

[19] L. Hammond, B. A. Nayfeh, and K. Olukotun. A single-chip multiprocessor. *Computer*, 30(9):79–85, 1997.

[20] K. Klues et al. Processes and Resource Management in a Scalable Many-core OS. In *HotPar10*, Berkeley, CA, June 2010.

[21] A. Krasnov et al. RAMP Blue: A Message-Passing Manycore System In FPGAs. In *Proc. of the Int'l Conference on Field Programmable Logic and Applications*, 2007.

[22] J. W. Lee, M. C. Ng, and K. Asanović. Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks. In *ISCA*, 2008.

[23] R. Liu et al. Tessellation: Space-Time Partitioning in a Manycore Client OS. In *HotPar09*, Berkeley, CA, March 2009.

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

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

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

[27] S. Mukherjee et al. Wisconsin Wind Tunnel II: A Fast, Portable Parallel Architecture Simulator. *IEEE Concurrency*, 2000.

[28] N. Njoroge et al. 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.

[29] 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.

[30] M. Pellauer et al. Quick performance models quickly: Closely-coupled partitioned simulation on FPGAs. In *ISPASS*, 2008.

[31] M. Pellauer et al. A-Port Networks: Preserving the Timed Behavior of Synchronous Systems for Modeling on FPGAs. *ACM Trans. on Reconfigurable Technology and Systems*, 2009.

[32] M. Pellauer et al. Soft Connections: Addressing the Hardware-Design Modularity Problem. In *DAC '09: Proc. of the 46th Annual Design Automation Conference*, 2009.

[33] S. Reinhardt et al. The Wisconsin Wind Tunnel: virtual prototyping of parallel computers. *SIGMETRICS Performance Evaluation Review*, 1993.

[34] M. Rosenblum et al. Using the SimOS machine simulator to study complex computer systems. *ACM Trans. on Modeling and Computer Simulation*, 1997.

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

[36] 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.

[37] S. Swanson et al. Area-Performance Trade-offs in Tiled Dataflow Architectures. In *ISCA*, 2006.

[38] Z. Tan et al. RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors. In *DAC '10: Proceedings of the 47th Annual Design Automation Conference*, 2010.

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

[40] C. Thacker. Beehive: A many-core computer for FPGAs, January 2010.

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

[42] M. Wehner, L. Oliker, and J. Shalf. Towards Ultra-High Resolution Models of Climate and Weather. *International Journal of High Performance Computing Applications*, 2008.

[43] T. Wenisch et al. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro*, July 2006.

[44] E. Witchel and M. Rosenblum. Embra: Fast and Flexible Machine Simulation. *SIGMETRICS Performance Evaluation Review*, 1996.