AFRL-RY-WP-TR-2009-1281

FIELD-PROGRAMMABLE GATE ARRAY (FPGA) EMULATION FOR COMPUTER ARCHITECTURE

J. Wawrzynek and K. Asanovic

University of California at Berkeley

AUGUST 2009
Final Report

Approved for public release; distribution unlimited.

See additional restrictions described on inside pages

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Defense Advanced Research Projects Agency (DARPA) and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2009-1281 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

*//Signature//

KERRY L. HILL, Project Engineer
Advanced Sensor Components Branch
Aerospace Components Division

//Signature//

BRADLEY J. PAUL, Chief
Chief, Advanced Sensor Components Branch
Aerospace Components Division

//Signature//

TODD A. KASTLE, Chief
Aerospace Components Division
Sensors Directorate

The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS**.

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
August 2009	Final	25 November 2008 – 31 August 2009

4. TITLE AND SUBTITLE
FIELD-PROGRAMMABLE GATE ARRAY (FPGA) EMULATION FOR COMPUTER ARCHITECTURE

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA8650-09-C-7907

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)
J. Wawrzynek and K. Asanovic

5d. PROJECT NUMBER
ARPR

5e. TASK NUMBER
YD

5f. WORK UNIT NUMBER
ARPRYD16

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of California at Berkeley
The Regents of the University of California
2150 Shattuck Avenue, Room 313
Berkeley, CA 94704-5940

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency/
Information Processing Techniques Office
(DARPA/IPTO)
3701 N. Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYDI

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2009-1281

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
PAO Case Number: DARPA DISTAR Case 15405; Clearance Date: 29 Apr 2010. This report contains color.

14. ABSTRACT
The scope of this program is to explore the use of FPGA-based systems for different applications including architecture-level FPGA emulation in support of future multiprocessor architecture research. We evaluate FPGA emulation approaches against conventional simulation techniques, and determine the missing capabilities that will require further development. In particular, we compare the various possible modeling approaches and evaluated their capabilities in terms of model fidelity, emulator performance, and design effort. We also investigate techniques to incorporate energy, power, and thermal models into architecture-level FPGA emulation frameworks.

15. SUBJECT TERMS
Computer architecture, simulation, emulation, FPGA

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kerry L. Hill
Unclassified	Unclassified	Unclassified	SAR	42	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

GLOSSARY

ABI	application binary interface
ASIC	application-specific integrated circuit
CAD	computer aided design
CISC	complex instruction set computer
DRAM	dynamic random access memory
DSP	digital signal processor
ECAD	electronic computer aided design
FAME	FPGA architecture model execution
FDIV	floating-point division
FPGA	field programmable gate array
GB	giga-bytes
GPU	graphics processing unit
I/O	input/output
ISA	instruction set architecture
ISCA	International Symposium on Computer Architecture
IT	information technology
MIPS	millions of instructions per second
MMU	memory management unit
OS	operating system
POSIX	portable operating system for Unix
RAID	redundant array of inexpensive disks
RAM	random access memory
RAMP	research accelerator for multiple processors
RISC	reduced instruction set computer
RTL	register transfer language
SAME	software architecture model execution
SODIMM	small outline dual in-line memory module
SPEC	standard performance evaluation corporation
VHDL	very-high-speed hardware description language
VLSI	very large scale integration
XUP	Xilinx university program

1.0 OVERVIEW

Given the multicore microprocessor revolution, the architecture research community needs a dramatic increase in simulation capacity. We believe that architecture-level FPGA emulation will provide a critical modeling framework for future multiprocessor architecture research. In this report, we introduce the terms FAME, for FPGA Architecture Model Execution, and SAME for Software Architecture Model Execution, to distinguish between conventional simulation techniques from emerging FPGA based methods. We evaluate FPGA emulation approaches against conventional simulation techniques, and determine the missing capabilities that will require further development. In particular, we compare the various possible modeling approaches and evaluated their capabilities in terms of model fidelity, emulator performance, and design effort. We also investigate techniques to incorporate energy, power, and thermal models into architecture-level FPGA emulation frameworks. To help clear up misconceptions, we propose a FAME taxonomy to distinguish the cost-performance of variations of these ideas. We show that FAME simulators can increase the number of useful architecture research experiments per day by a factor of 100 over SAME simulators.

2.0 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 in the early stages of research exploration. Architects can explore many variations of a design simply by changing software simulator parameters, and in the era 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 [7], however, has both increased the complexity of the systems architects want to simulate and removed the straightforward path to simulator performance scaling.

This report surveys the evolution of simulators as architectures increased in complexity and argues that architecture research now faces a crisis in simulation because of the new requirements and the consequences of the multicore revolution. We label the two paths forward in multicore simulation as Software Architecture Model Execution (SAME) or FPGA (Field-Programmable Gate Array) Architecture Model Execution (FAME). While SAME certainly merits continued investigation, we and others in the Research Accelerator for Multiple Processors (RAMP) project [49] are excited by the progress and the potential of FAME.

In discussions with architects outside the RAMP community, there has been confusion about the different approaches and goals of the various RAMP projects. Inspired by the original five RAID levels [36], this report proposes a four-level taxonomy of increasingly sophisticated FAME levels that attempts to capture the progress that has been made by the whole RAMP community in the last few years.

The report concludes with a fallacies and pitfalls section, followed by a conclusion section with recommendations for future directions.

3.0 PROTOTYPING

We begin by briefly reviewing the role of prototyping in architecture research and how prototyping is different from simulation. The goals of an architecture research prototype are quite different from that of a simulator. A research prototype is a working implementation that provides insight into the implementation issues behind a proposed new architecture or new architectural mechanisms. A simulator, on the other hand, is a model of a machine, and can be highly parameterized and quite abstract, to allow rapid exploration of a large design space of possible options.

Hardware prototyping has a long history, reaching back to the very first computers built in universities, such as the Harvard Mark-I and EDSAC. In the 1980s, many researchers would build prototype chips to illustrate the value of their architectural innovations. For example, the case for RISC architectures was substantially strengthened by the prototype RISC chips built at Berkeley [37] and Stanford [23], which ran programs faster than commercial machines despite being produced by small academic teams. Similarly, the later Stanford DASH [30] and MIT Alewife [3] projects provided considerable insight into the implementation and viability of large-scale directory-based cache-coherent shared memory architectures. More recent prototyping efforts include Raw [46], TRIPS [39], Imagine [4], and Scale [27].

The process of constructing a working architectural prototype is usually far more valuable than the end result, helping inventors understand how a new architectural idea can be implemented. When completed, a successful prototype will provide a credible proof-of-concept to help explain the idea and convince practitioners to adopt the technology.

Although prototypes are sometimes justified as a way to gather evaluation data on much larger and longer programs than possible with software simulators, usually this is (or becomes) a secondary concern. In our experience, few computer architecture prototypes support extensive parameterization and instrumentation due to the additional design effort and resources required. Even when hardware hooks are added, the supporting software infrastructure to exploit these features rarely materializes, either because of limited project resources or because experimental needs were not well understood or have changed from when the prototype design was frozen. Prototyping projects also tend to focus implementation effort on the novel mechanisms and often make expedient simplifications in other well-understood areas (e.g., by omitting floating-point hardware or virtual memory), which limits the kinds of software that can be run. For these reasons, prototyping is not an alternative to simulation.

Conversely, when the implementation of a mechanism is not well understood, it is by definition not possible to construct accurate simulator models. Hence, simulation and prototyping are complementary techniques.

Despite their value, the bar for prototypes has been raised. Shrinking feature sizes have led to multi- gigahertz microprocessor clock rates and a rapid growth in architectural complexity. Consequently, the engineering skill, design effort, and fabrication cost required to build a compelling prototype micro- processor have risen to the point where few researchers now

contemplate such a project. Even when research prototypes are successfully completed, the quality of implementation is often markedly inferior to production designs, leading to doubts about the relevance of the prototype. But it is primarily the act of prototyping that yields valuable implementation insights, and these are often largely independent of implementation technology. Simulator models can be constructed using the prototype design as a guide, but then extrapolating to model the effect of a more realistic implementation or an advanced future technology.

Given that most of the implementation insights are developed during the prototype design phase and that most of the cost is incurred during fabrication, one viable intermediate approach is to complete a detailed design using VLSI CAD tools without proceeding to fabrication. Simulator models can then be calibrated using timing, area, and power data extracted from the tools. This approach still requires significant engineering skill and design effort using a large and complex tool set, but is ultimately not as credible (or as satisfying) as a working prototype.

Given the high cost and long turnaround of prototype chip fabrication, some researchers are using FPGAs to construct relatively inexpensive and malleable working prototypes. The timing, area, and power of an FPGA prototype are very different from a production chip implementation, but the hardware design process is similar enough to yield many of the same important insights. A promising direction is to combine FPGA prototyping with detailed VLSI CAD design to provide both a working prototype and believable implementation metrics.

While we believe FPGA prototyping can be a valuable architecture research tool, our approach and the subject of this report is a very different technology: Using FPGAs to accelerate the execution of highly parameterized and thoroughly instrumented architecture simulators. We have found the difference between FPGA prototypes and FPGA simulators to be one of the main sources of confusion when discussing RAMP with those outside the project, and our goal in this report is to clarify the difference and provide taxonomy of FPGA simulation approaches.

4.0 SIMULATION TECHNIQUES

A modern processor running an application workload is a complex system that is difficult to model analytically; yet building a prototype for each design point is prohibitively expensive, so software simulators have become the primary method used to evaluate architecture design choices. We call the machine being simulated the target and the machine on which the simulation runs, the host. In this section, we present a brief review of the evolution of software simulator technology. This technology largely followed the sophistication in computer designs that were enabled by Moore's Law.

4.1. Uniprocessor Simulators

Much of the architecture research in the 1980s involved in-order processors, and popular topics were instruction set architectures (RISC vs. CISC), pipelining, and memory hierarchies. Instruction-set simulators were very slow and so pipelining and memory hierarchy studies relied on address traces to drive simulators that only simulated the portions of the computer of interest. Given the common needs of researchers, some shared their trace-driven simulators for others to use, such as the popular Dinero cache simulator [18]. Although it was time consuming to collect traces, some generous researchers would share them with the community. Combining free traces with free simulators helped many architects undertake memory hierarchy research.

Enthusiasm shifted to out-of-order processors in the 1990s. Popular topics included alternative microarchitectures, branch predictors, and memory hierarchy studies. Address traces were no longer adequate for microarchitecture or memory hierarchy studies due to the complex out-of-order nature of the processors. This complication led to the popularity of execution-driven simulators, which model the internal microarchitecture of the processor on a cycle-by-cycle basis. Alas, the speedup in host uniprocessor performance did not match the increase in target architectural complexity, so the simulation of each target instruction took more host clock cycles. In addition, Moore's Law enabled much larger and more elaborate memory hierarchies, so architects needed to simulate more instructions to warm up caches properly and to gather statistically significant results. Moreover, as computers got faster and had more memory, programs of interest became bigger and ran longer, which increased the time to simulate benchmarks [29].

One approach to reduce simulation time is simply to reduce the size of the input data set so that the program runs in less time. For example, SPEC CPU 2006 offers two input sizes per benchmark: test run and reference run. The test run of the full suite took 6 minutes to run natively on the SPEC test bed machine while the reference run took 600 minutes [24]. Although the test run is 100 times shorter, it is designed only to check the program is working and is not representative of the behavior of the full reference run. By analyzing the program it is possible to construct a reduced input set that is more representative of the application behavior. For example, MinneSPEC used this approach for the SPEC CPU 2000 suite [25]. However, this manual approach is time consuming and it is unclear how representative it is of the original workload.

Another way to reduce the simulation time is to only simulate a small sample of the full application run in detail. A simple scheme that was widely used was to skip some number of instructions at the start of a program run, assuming these represented initialization code, and then simulate the next N million instructions in detail, assuming these represented the steady state execution in the program. To quickly skip over the initialization code, a fast functional simulator is used that only maintains architectural state (ISA registers plus memory). At the start of the detailed sample, the architectural state is copied into a detailed simulator, which is then run for a period to warm up the microarchitectural state (caches and predictors) before gathering evaluation data. For large caches, the warm up time can be large and so several researchers proposed techniques to bound the amount of warmup time required at the start of each sample [28], [16], [22].

Different programs have very different execution profiles, but surprisingly many studies used the same initialization time and sample length for all programs in their evaluation. An obvious question emerged "if you can only afford to simulate X% of the program, which segment is the most representative to simulate?". SimPoints [42] finds the most representative segments to model in detail given a constraint on how many separate segments and total instructions to use. SMARTS [51] models enough small samples of the execution in detail to provide a statistical confidence bound that the benchmark behavior was faithfully captured. The SMARTS approach "fast forwards" between samples using a functional simulator that also updates large microarchitectural state components, such as cache tags and predictor state. At each sample point, the microarchitectural state is also copied into the detailed simulator, reducing the warm up time for each detailed sample point.

To get even faster speed up for functional simulation, dynamic binary translation can be employed to translate target instructions into host instructions with added instrumentation code. Pixie [2] and Shade [15] instrumented program binaries for MIPS and SPARC to generate the address traces on the fly, and this technique was later adapted for functional execution.

The complexity of building an execution-driven simulator, together with the increasing use of commercial instruction sets, common benchmarks, and a common research agenda led to the development of shared execution-driven simulator models, of which SimpleScalar is surely the most widely used example [8]. Architects would either use the standard options provided by the simulators or make modifications to the simulator to explore their inventions, and use the precompiled benchmark binaries as a workload. Once again, interesting free simulators let many architects perform the type of research that the simulator supported.

4.2. Multiprocessor Simulators

Another popular topic in the late 1980s and early 1990s was multiprocessors, with popular research topics including memory hierarchies, consistency models, coherency techniques (bus snooping vs. directories), and uniform versus non-uniform memory accesses.

Simulating parallel target machines is considerably 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 run multithreaded or multiprogrammed across the multiple cores in the simulated target machine.

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 [35]). The SimOS project demonstrated how to run an operating system on top of a fast software simulator [38]. 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 [53]. 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 others [32], Simics has become a popular research tool in the architecture community.

Although techniques such as dynamic binary translation and trace-driven simulation help with the interpretation of the functional behavior of each instruction, they do not help with the considerably more compute-intensive task of modeling microarchitecture details of the processor and memory hierarchy. Surprisingly, 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.

As with uniprocessors, researchers have considered using reduced input sets or sampling to reduce multiprocessor simulation time. Alas, mixed mode simulation and sampling does not work well for multiprocessor simulations [9]. For uniprocessors, the program execution should be the same no matter what 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 software such as databases are also amenable to sample-based simulation, since any sample is representative of some legal overlap of independent transactions [52].

4.3. The Static Software Premise

Implicit in many of the techniques used to reduce simulation time (traces, reduced inputs sets, sampling) was the assumption that software changes slowly and is independent of the target architecture. Thus, suites like SPEC with two-dozen old programs were reasonable benchmarks for architectures of the future, as long as SPEC suite changed every four to five years to reduce gamesmanship by compiler writers and architects. Similarly, research groups would create their

own long-lasting benchmark suites, and some of these became popular, for example, SPLASH and SPLASH 2 [54] for multithreaded programs. Parsec and STAMP are modern examples of such parallel benchmark suites [10] [13].

New programs, new programming models, and new programming languages have been largely ignored. The argument was either that architects should wait for them to prove their popularity and become well- optimized before paying attention to novel systems–which could take a decade–or that there was nothing you would do differently for them than you would for old programs written in the old programming languages. Since benchmarking rules prohibited changing the program, architects generally treated the programs as static artifacts to measure without understanding either the problems being solved or the algorithms and data structures being used. In fact, many architectural studies have simply used precompiled binaries distributed with the shared simulator infrastructure.

5.0 THE MULTICORE REVOLUTION

As has been widely reported, the end of ideal technology scaling together with the practical power limit for air-cooled sockets forced all microprocessor manufacturers to switch to multiple processors per chip [7]. The path to more performance for such multicore designs is increasing the number of cores every technology generation, with the cores themselves essentially going no faster.

There are four subtle impacts of the multicore era on architecture simulation. First, simulators will no longer get faster every 18 months without effort. Like all other programmers, simulator developers will need to discover how to get more performance from more cores. Second, the design space of possible future multicore architectures is vast, so it's much harder to develop a standard simulator that many can use to do architecture investigation. Third, parallel application performance can be very non-deterministic and multiple different runs of the same code might be required to gain confidence in reported performance numbers [5]. Fourth, to cope with power limits, microprocessors now use many techniques to improve power-performance that will require extended simulation time. For example, dynamic voltage scaling and frequency scaling per core means that clock cycles and instructions per clock cycle are no longer accurate measures of performance. Recent Intel microprocessors even offer "Turbo" modes that will allow some cores to run at much higher clock rates temporarily if current operating temperature allows. These modes will be turned on opportunistically by the hardware, without even the operating system being aware. The time constants for studying power and temperature will require very long simulations: from 1 to 100 seconds of target time.

Given the multicore revolution, we claim that the biggest architectural research challenges now deal with multiple processors rather than increasingly sophisticated single processors. Indeed, there is even a commercial movement towards "manycore" architectures that use many simpler processors, such as the IBM Cell, Intel Larrabee, and Sun Niagara [21], [40], [26]. Hence, architecture investigations now need to be able to look at many processors in addition to memory hierarchy and processor design. The number of cores per chip, sophistication of these cores, and even the instruction sets of the cores are all open to debate. Issues that have received little recent attention, like on-chip interconnect, are vital. Moreover, power is at least as important a resource to conserve today as chip area was in the past.

Perhaps the greatest changes will actually be above the hardware, in the new software stack. Software researchers and practitioners are trying to address the grand challenge of the multicore revolution: to make it as easy to write programs that are efficient, portable, correct, and scale as the number of cores per microprocessor increases, as it has been to write programs for sequential computers [7]. Hence, old programs and operating systems are being rewritten to be compatible with increasingly parallel microprocessors. New programming models, new programming languages, and new applications are being invented, and given the urgency of the multicore challenge, architects cannot ignore them. Creating portable parallel programs that maintain high-performance has long been a challenge, so techniques like autotuning are becoming popular [17]. Rather than thinking of the program as a static object, autotuning adapts the program to the features of the computer on which it is running and perhaps even to the input data to the program. Such self-adaptation can happen at the time the

computer is announced, at the time software is installed onto a particular example of that computer–depending on the number of cores, clock rate, amount of memory, and types of compilers installed–or even during the execution of the program.

In these revolutionary times, we draw four conclusions about the future of simulation:

1. Given software churn, new techniques like autotuning, new issues like temperature and power, and the increasing number of cores, architecture research in the multicore era requires simulation of many more target instructions than it did for the uniprocessor era.
2. Given the natural non-determinism of parallel programs running on a parallel operating system, to be sure of the significance of any results, architects need to run programs many times and then use averages with confidence intervals, which again increases the number of instructions that should be simulated [5].
3. Given our lack of intuition about how new parallel programs will behave on novel hardware and new metrics like power and absolute time (instead of clock cycles), we'll need to simulate this greater number of instructions at a greater level of detail than in the past.
4. Given this lack of understanding, we also need to run experiments for all portions of the computer and with added instrumentation, since we don't yet know what components or metrics can be safely ignored in novel multicore computers running novel applications.

Multiplying these four conclusions together suggests an upsurge in the demand for simulation over what can be done with traditional software simulators today by many orders of magnitude.

To put into perspective how many instructions are actually being simulated per processor using software simulators today, Table 1 compares the number of instructions simulated per benchmark run per paper from the 2008 International Symposium on Computer Architecture (ISCA) to the same conference a decade earlier. Recent papers simulate many more instructions and more cores if you compare medians, but the number of instructions per core per benchmark run was just 100 million in 2008 vs. 267 million in 1998. We assume that the authors didn't simulate more instructions because they didn't need more to generate accurate conclusions from their experiments. However, we see no evidence of the dramatic rise in simulation time that we argue is needed for the multicore challenge. In fact, it is heading in the other direction: these numbers correspond to about 0.05 seconds of target execution time in 2008 vs. about 0.50 seconds in 1998.

Table 1. Number of instructions simulated per core per benchmark

ISCA 2008				ISCA 1998			
Total Instructions	Cores	Instructions per Core	Programs	Total Instructions	Cores	Instructions per Core	Programs
150M	16	9M	SPEC2006	100M	1	100M	SPEC95
50M	4	12M	SPEC2000	100M	1	100M	SPEC95
240M	16	15M	SPLASH-2	100M	1	100M	SPEC95, NAS, CMU
500M	16	31M	Traffic Patterns	100M	1	100M	SPEC95
650M	16	40M	SPEC2000	171M	1	171M	SPEC95, SPLASH2
2300M	32	72M	STAMP+SPLASH	200M	1	200M	SPEC95
100M	1	100M	SPEC2000	236M	1	236M	SPEC95
100M	1	100M	SPEC2000	267M	1	267M	SPEC95
1600M	16	100M	SPEC2000	267M	1	267M	SPEC95
1000M	8	125M	SPLASH-2, SPECJBB	267M	1	267M	SPEC95
				267M	1	267M	SPEC95
2500M	16	160M	Hashtable, Rbtree	325M	1	325M	SPEC95
1000M	4	250M	MinneSPEC	860M	1	860M	OLTP/DB, SPEC95
2000M	8	250M	SPEC2000	900M	1	900M	OLTP/DB
1000M	1	1000M	MinneSPEC	1000M	1	1000M	Synthetic
12000M	1	12000M	SPEC2000	84000M	8	10500M	SPLASH2, Pmake
825M	16	100M	MEDIAN	267M	1	267M	MEDIAN

6.0 SOFTWARE ARCHITECTURE MODEL EXECUTION (SAME)

As mentioned above, the performance challenge for software simulators is turning the increasing number of host cores into higher simulated target instructions per second. We believe the challenge will be far harder for detailed simulation than for functional simulation, as there is naturally much more communication between components in a target cycle.

Figure 1 demonstrates the impact on simulation speed as the number of cores and level of detail increases: functional only, functional plus caches, and functional plus caches plus detailed memory system model on a log-log scale. The MIPS gap between simulating 1 core at functional level and 64 cores at detailed memory is 150,000. We used an interleaving factor of 3, meaning we simulate 3 instructions of one target core before switching to the next to improve host cache locality (there was up to a 10% increase in simulation speed between 3 and 1, and larger interleaves gave diminishing returns). The parallel program run on the target computer was a Cholesky Factorization kernel. Simics was run on a 2.33 GHz Clovertown host processor. Note that the data point for cache level simulation for one core (5.4) seems low by factors of 3 or 4; we believe this is due to the constant overhead of simulating some shared components of the target cache hierarchy.

The slowdown is almost a factor of 100 when adding caches and almost 2000 when adding caches and the detailed memory model. Another critical factor, however, is the number of cores simulated, with 64 cores simulating about 80 times slower than a single core. The total slowdown per core of functional simulation of 1 core versus detailed memory simulation of 64 cores is about 150,000.

The opportunity for a SAME simulator is to leverage the natural parallelism in the target machines to run the simulator faster on the multicore hardware of the host machine. The difficulty of this challenge is demonstrated by the Electronic CAD industry. Despite the availability of parallel servers for more than a decade, and even though there is massive potential parallelism in the system being simulated, most of the state-of-the-art register-transfer language (RTL) simulators are still not parallelized. Note that the market for RTL simulators is likely much larger than for architecture simulators, so it’s hard to be hopeful for commercial progress on this topic.

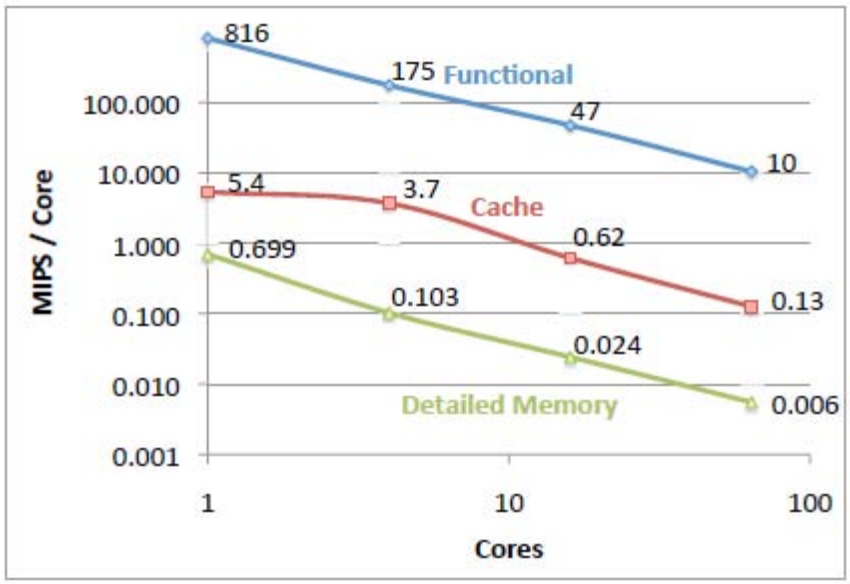

Figure 1: Simulated target MIPS/core on Simics under varied numbers of cores

An alternative approach would be to use a cluster to emulate a multicore computer, perhaps with one node of cluster for each processor, and then use the cluster network for communication between cores. As most clusters typically interconnect via hierarchies of Ethernet switches, a major challenge will be to prevent the network from becoming a bottleneck. We believe the issue is more the latency of synchronization rather than data bandwidth. For example, the Wisconsin Wind Tunnel project spent 40% of the time in the network in their simulations [33]. Since they were using the low-latency Myrinet network and much slower computers, we expect it to be much worse today given higher latency Ethernet networks and multi-gigahertz processors. Once again, we expect the challenge will be greater as the level of detail required increases.

While we encourage others to make progress on this important but difficult problem, we are more excited by an alternative approach.

7.0 FPGA ARCHITECTURE MODEL EXECUTION (FAME)

As observed by the Research Accelerator for Multiple Processors (RAMP) Project [49], FPGA technology offers a promising vehicle for architectural investigation.

The first advantage is that a program written in a Hardware Description Languages (HDL) like Verilog or VHDL naturally runs in parallel on an FPGA, while a program written in languages like C or Java naturally runs sequentially on a computer. Indeed, the programmer needs to do extra work to prevent parallelism in an HDL on an FPGA. A second advantage is relative to hardware development, changing an FPGA is very fast. You only need go through the sequence of CAD programs and download the contents into a FPGA, which can take less than one hour for a small design. Hence, you can "tape out" a new design every day, as opposed to every three to six months with real hardware. (You also don't have to pay the millions of dollars for new masks and to fabricate chips.) This flexibility also makes it easy to add runtime measurements without slowing down simulation since FPGAs are naturally parallel. Third, the FPGA hardware is relatively inexpensive. At the low end is a $750 board with a small FPGA (Virtex 5 XUP board) to a $15,000 board with four medium-sized FPGAs (BEE3). Fourth, relative to software simulators, FPGAs are fast: today's FPGA-based simulators run at about 100 target MIPS with timing models. Finally, like microprocessors and other CMOS-based devices, FPGAs also take advantage of any advances in device geometry and the resulting increased transistor count and on-chip functionality and thus are roughly doubling in capacity every 2 years. Hence, rather than a processor taking many FPGAs as in the past, depending on processor complexity and size of FPGA, now many processors can fit on a single FPGA. Looking forward, we would expect the number of cores per FPGA to double roughly every two years. This trend is a great match to the multicore challenge.

There are downsides to using FPGAs as well. Simulator developers need to learn a HDL and FPGA CAD tools. Moreover, FPGA CAD tools are often of poorer quality than software tools (or even ASIC tools) and typically have longer run times, so they can be frustrating to use. While such problems in the past led universities to make their own CAD tools, the FPGA industry keeps internal formats proprietary, thereby preventing third parties from developing their own CAD tools. Finally, compared to hardware, logic is inefficient in FPGAs unless you can find equivalent hardware primitives on FPGAs, such as DSPs and RAMs. In practice, FPGAs are bad at logic such as pipeline forwarding but good at state storage such as register files and caches.

Given the multicore revolution and the need to simulate many more instructions at a finer level of detail, FPGA Architecture Model Execution (FAME) simulators offer a promising direction. In our discussion with other architects, there has been confusion about the terms, how our ideas relate to prior work using FPGAs for architecture prototyping and chip simulation, and what can, and cannot be done using FAME. This section and the following try to address these issues.

Just as in software simulation, there are various options in implementing a FAME simulator that have a big impact on cost and performance.

1) Direct vs. Decoupled. One of the first uses of FPGAs was to simulate the logic of a new chip design, to help find bugs in the logical design before the chip was taped out. Quickturn was one of the early products to offer this service. By running at about 1 MHz to 2 MHz, Quickturn boxes could run much larger test programs than could be simulated by ECAD logic programs. The current version of this product is the Cadence Palladium. We describe this modeling style as a *Direct* implementation, where the host is directly implementing the RTL of the target machine with a one-to-one mapping between target clock cycles and host clock cycles.

   The most powerful FAME option, which improves efficiency and enables the other options, is to adopt a *Decoupled* design, where a single target clock cycle can be implemented in many or even a variable number of host clock cycles. Models now have to use additional host logic to model target time correctly, and a method is needed to exchange target timing information between modules, as different modules in a design may then run with different target to host clock cycle ratios. One example of this approach is in the Green Flash climate supercomputer project [50], which requires software to be tuned for each candidate architecture design point. The processor core is generated by the Tensilica processor generator tools and mapped to FPGA boards directly, but a decoupled timing model is used to model the memory system to make the target DRAM performance more realistic.

2) Full RTL vs. Abstract. For the direct mapping case, the full RTL of the target machine is required.

   Instead of implementing the full RTL of a real hardware design, an *Abstract* simulator uses higher- level descriptions of pieces of the design. HASIM [20] 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.
   This FAME option clearly reduces the difficulty of performing an architectural experiment, since the full RTL design is not required. More importantly, it allows parameterized components and timing models, such as memory hierarchy models. Since these parameters can be set at runtime in the FAME implementation without resynthesizing the design, this upgrade dramatically increases the number of architecture experiments that can be performed per day, since in practice successfully synthesizing a design and loading it into onto an FPGA board needs a skilled person involved to make sure every step happens correctly.

3) Single-Threaded vs. Multi-Threaded. A standard approach to obtain greater performance from a processor is to switch threads every clock cycle so that all dependencies are resolved by the next time that thread is executed [6], [11]. This FAME option introduces the concept of host *Multi-threading* as a powerful technique to improve utilization of FPGA resources by the simulator. When the target system contains multiple instances of the same component (e.g., multiple processors in a manycore design), the host model can be designed so that one physical FPGA pipeline can model multiple target components by interleaving their execution using multi-

threading. For example, a single FPGA processor pipeline might model 64 target processor cores. Or, a single FPGA router pipeline might model 16 on-chip routers. One advantage of this approach is that it can hide emulation system communication latencies that may otherwise slow the emulation. 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 see a stall. Multi-threaded emulation adds additional design complexity, but can provide a significant improvement in emulator throughput.

Many of these choices make sense in combination. Inspired by the five levels of RAID, the next four sections present a four-level taxonomy of FAME that improves cost, performance, or flexibility at each new level. 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. The least-significant bit represents Direct (0) vs. Decoupled (1), the middle bit represents Full-RTL (0) vs. Abstract (1), and the most-significant bit represents Single-Threaded (0) vs. Multi-Threaded (1). Table 2 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.

Table 2. Summary of four levels of FAME

Level	Name	Example	Strength	Experiments/ day / $1000
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	60
111	Multi-threaded FAME	RAMP Gold	Lower cost; may improve clock rate	1080

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 comments above about power, dynamic frequency per core, increasing number of cores and so on, 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 coherency. You can think of 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. 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 2 estimates this metric for 2009 prices.

7.1. Direct FAME (Level 000): (e.g., Quickturn/Palladium)

The common characteristic of Direct FAME systems 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. The current examples of Direct FAME systems include Cadence Palladium and Mentor Graphics Veloce, which cost millions of dollars. They are no longer based on commercial FPGAs because the extra features like DSPs and block RAMs are not useful at this level, so the companies design their own custom simpler, denser FPGAs[1].

We consider Direct FAME a type of simulator model, because the RTL is for a target machine designed to be fabricated in some custom chip technology. This is in contrast to the FPGA architecture described above, where the FPGA prototype is itself the final target implementation and hence where the RTL will be tuned to work well on the FPGA. For the same reason, we do not consider FPGA computers such as Xilinx Microblaze or Convey HC-1 as FAME systems.

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 there are no parameters, we would have to go through 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 takes up to 30 hours to set up a new design[2]. Let's assume Direct FAME can do one experiment per day. The number of experiments per day per $1000 is then 1/1000 or 0.001. Note that in addition to low cost-performance, Direct FAME takes a great deal of effort for an architect to change the RTL for another experiment, unlike some of the later FAME Levels.

Although helpful in debugging the designs of a complete microprocessor intended for fabrication, Direct FAME is too expensive and time consuming to use for early-stage architectural investigations.

7.2. Decoupled FAME (Level 001) (e.g., Green Flash)

When a custom microprocessor RTL design is directly synthesized to an FPGA, certain ASIC features, such as associative structures or multiport register files, can consume considerable FPGA resources. For example, Green Flash [50] can fit two Tensilica cores with floating-point units per medium-sized FPGA, and it runs at 50 MHz[3]. The system memory is implemented in DRAM connected to the FPGAs which runs much faster, however, and so decoupling is used to make the DRAM timing match the intended target machine DRAM timing.

To perform a 16-core experiment, it would take two BEE3 boards, which cost academics about $15,000 per board, plus the FPGAs and DRAMs, which cost about $3000 per board. Commercial pricing is higher, perhaps two to three times, and is negotiated on a per-customer basis. It would take about 2 GHz/50 MHz or 40 seconds to run the experiment. It takes 8

[1] Kees Vissers. Private communication, June 2009.
[2] Chuck Thacker. Private communication, May 2009.
[3] John Shalf. Private communication, June 2009.

hours to synthesize and place and route the design. Since this level has a few timing parameters, such as DRAM latency and bandwidth, Green Flash can run about 24 experiments per synthesis [41]. 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 the fact that the processor cores fit on a single FPGA so they don't incur the FPGA partitioning costs that slows Direct FAME systems that model designs at the gate level and because 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 [43]. 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.

Hence, Decoupled FAME will take a great deal of effort to perform a wider range of experiments compared to the higher levels of 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 binary numbering system).

7.3. Abstract FAME (Level 011) (e.g., HASIM)

Abstract FAME allows high-level descriptions for that early-stage exploration, which in turn simplifies the design and thereby reduces the synthesis time to less than 1 hour, and it fits onto a single BEE3 board. More importantly, it allows the exploration of many design parameters without having to resysthesize at all, which dramatically improves cost-performance. Often in this level of FAME a processor model is divided into separate functional and timing models that do not correspond to structural components in the target machine. In some cases, the timing and functional models can be implemented on different platforms completely. For instance, a complex timing model could be mapped to an FPGA with the relatively simpler functional model running in software on a general purpose processor.

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 perform the snooping on the addresses. Let's assume the average number of host cycles per target instruction is 4 and the host clock rate is 100 MHz. The time for one experiment is then 2 GHz /100 MHz * 4 = 80 seconds. Since a person does not have to be involved to run the CAD tools and load the FPGAs, the number of experiments per day is 24 hours / 80 seconds = 1080. The number of experiments per day per $1000 is then about 1080/18, or 60. Abstract FAME (Level 011) makes a dramatic improvement in this metric over lower FAME levels: by a factor of almost 100 over Decoupled FAME (Level 001) and a factor of 60,000 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 greatly lowers the effort for experiments and greatly increases the number of people who can perform architecture experiments.

Once again, the advantages of abstract designs and decoupled designs are so great then we assume that any further level is both Abstract and Decoupled.

7.4. 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. 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 that could be on the critical path, such as forwarding logic for pipelined execution.

Since we are multiplexing a single FPGA, we can use the much less expensive XUP board, which costs $750. Due to the multithreaded design, we assume that each instruction takes 1 host clock cycle per core, or 16 host clock cycles per multiprocessor instruction per host pipeline. We believe we can include three pipelines on such an FPGA, so we would need to add host cache coherence for 3 pipelines, which is not too hard to do. The time for one experiment is then 2 GHz /100 MHz * 16 / 3 = 107 seconds. The number of experiments per day is 24 hours / 107 seconds = 810. The number of experiments per day per $1000 is then about 810 / .75 or 1080. Multithreaded FAME (Level 111) improves this metric by a factor of almost 20 over Abstract FAME (Level 011), by a factor of about 1500 over Decoupled FAME (Level 001), and by a factor of more than 1,000,000 over Direct FAME (Level 000).

In addition, Multithreaded FAME lowers the entry point cost for people who want to do experiments by a factor of 24 to 48 versus Abstract or Decoupled FAME, making it possible for many more researchers to do parallel architecture research, which is just what we need in these demanding times.

7.5. Hybrid FAME Simulators

Although these layers are presented as completely separate approaches for pedagogic reasons, real systems will combine modules at different levels together, or even use hybrid designs with some portions in FPGA and others 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, where the RTL mapped to the FPGA is the same as will be mapped to the final ASIC implementation (FAME Level 001), but the rest of the system is described at an abstract level (FAME Level 011).

An example of a mixed FAME/SAME system is the Protoflex system [14]. It uses a FPGA-system to simulate the state of the memory hierarchy throughout the run of the program, and then uses a software simulator to investigate performance of short but interesting stretches of the program. Note that the FPGA portion is abstract, in that it does not use full RTL of the

design, yet it does not offer a timing model, so it does not fit cleanly into the FAME taxonomy above.

8.0 DESCRIPTION OF RAMP GOLD

An example of Multithreaded FAME is RAMP Gold [45]. RAMP Gold is a prototyping platform developed in collaboration with the Berkeley Parallel Computing Laboratory for experimental development of next generation single-socket multicore architectures. It will be used to validate micro-architecture mechanisms and to provide an early platform for operating system, library, and application development. RAMP Gold v1.0 is implemented on a single Xilinx Virtex5 LX110T FPGA. The FPGA board connects to a PC server through a 1 Gbps Ethernet link. This front-end server is responsible for all simulation controls, such as loading executable binaries and dumping simulation statistics. The front-end machine also serves complex system calls whose functionality is not implemented in the simulated target software kernel, such as file I/O.

Inside the FPGA, we currently use a single-channel 233 MHz DDR2 memory controller based on the BEE3 memory controller [1]. It supports up to a 2GB dual-rank SODIMM. We use the 2GB DRAM for both target memory and some simulated microarchitectural state, such as target cache tags and data. On top of the memory controller, there is a host cache whose purpose is only to accelerate the simulation it does not affect the timing of the target memory system.

The simulation engine includes two basic models: a processor model and memory system model. Each processor model emulates a single in-order issue 32-bit SPARC V8 CPU. The functional model is built on our previous work in [44], which is highly optimized for the Xilinx Virtex 5 family of FPGAs and runs at over 100 MHz. Every functional model simulates up to 64 target processors using host multithreading. The functional model implements the full SPARC V8 ISA, including floating-point and precise exceptions. It also has been verified against the SPARC V8 verification suite. All integer instructions, double precision floating-point multiply, add/subtract and conversions are implemented in hardware. Complex floating-point operations, such as FDIV, cause traps and are emulated in the simulated supervisor. The timing model emulates a classic five-stage pipeline. For instance, it models pipeline stalls such as the load-use delay and the branch delay slot. The number of cores being simulated can be configured at runtime without resynthesizing the simulator.

The processor-timing model is connected to a configurable model of the memory hierarchy. The target system has split first-level instruction and data caches connected to a unified L2 cache, which can be configured as private or shared. Many cache parameters, including cache size, line size, associatively, and hit latency, are configured at runtime; within reasonable limits, varying them does not require resynthesis. In the current version of the model, the write-back and write-allocate policies are fixed, and replacement is pseudorandom; these restrictions are only for design simplicity and are not the result of inherent limitations in RAMP Gold.

In RAMP Gold v1, the target caches are automatically kept coherent because the underlying host is coherent; protocol transitions are not modeled, nor is contention for the interconnect. Constructing a cycle- accurate model of a coherence protocol is among our planned future work. Note that one of the advantages of the higher FAME levels is that a complete working RTL design is not required before beginning architectural exploration. The underlying memory

functional model keeps our shared memory design functionally correct, even if timing is incorrect. As with software simulators, timing model validation will be required to ensure reasonable accuracy is achieved.

In addition to hardware simulation models, RAMP Gold also provides a systematic design and verification environment. Our target compiler tool chain is directly built from the latest GCC without any modification. Newlib provides a lightweight C library that is ABI-compatible with OpenSolaris, so the same single-threaded application binary can run on RAMP Gold and Sun servers. Multi-threaded binaries are object-code compatible but must be linked against a different implementation of POSIX threads.

9.0 HANDLING DYNAMIC CLOCK FREQUENCY, POWER, AND TEMPERATURE IN FAME

Modern simulators must now offer the ability for each portion of the computer to run at different clock frequencies, to estimate power usage, and to model temperature. Given the naturally parallel nature of FPGAs, we believe these will be much easier to handle in FAME than in SAME. To make this point, we describe our plan for them in RAMP Gold.

To model dynamic clock frequency changes, we'll use a much faster target timebase–say, 16 GHz–and track time via that master clock. The control portions of each RAMP Gold component will then simply use the very fast clock to coordinate and account for events. Note that even if a component had a clock that is not a multiple of the master clock, we would simply round up to the next master clock cycle. Hence, the master clock rate just needs to be fast enough that such "quantization errors" are too small to appreciably affect the results. Fortunately, quantization errors don't accumulate.

Like others, our approach to power is to record which components are active over time and then plug them into a formula that estimates the power of a component given its activity [12]. These component power models can come from CAD tools if the components are actually designed. If not, they are estimated by higher-level models of components [12], although there is concern about the accuracy of these higher- level models. One advantage of FAME is that there is little simulation time penalty to supply the lower level activity parameters that are needed by the more accurate, lower-level power models. Hence, it may prove easier to have more accurate power estimates for FAME than for SAME.

We can use a standard approach to simulating temperature, by taking the estimates of energy consumed for every, say, 10 milliseconds of target time and plug these into a thermal simulation. This floating-point intensive calculation can be offloaded to the host workstation rather than on FAME itself [34]. We believe the relatively slow change of temperature will allow us to retain fast simulation with the decoupled but high performance nature of GPU computing.

10.0 FAME VS. SAME: PRELIMINARY PERFORMANCE COMPARISON

Simics is a popular architecture simulator which has often been used by architects to prototype and evaluate new microarchitectural ideas. Figure 2 compares performance of Simics to RAMP Gold for 16 processors for functional and timing variations for a functional single pipeline per FGPA. The RAMP Gold detailed memory model is an estimate, and the parallel program run on the target computer was a Cholesky Factorization kernel for this data. When running on a high-end workstation and simulating 16 SPARC processor cores in a purely functional mode, Simics can achieve approximately 42 MIPS per core of simulated performance. When simulating the cache, the rate is 0.56 MIPS per core, or a slowdown of nearly 100. Moreover, the addition of the GEMS (General Execution-driven Multiprocessor Simulator) modules that accurately model timing drops performance to 0.024 MIPS per core, a slowdown of 2000 over functional simulation. For RAMP Gold, it is 4.0 MIPS per core for the functional simulator and 3.2 MIPS per core for the version with timing and an (estimated) MIPS per core of about 2.0 with detailed memory, a slowdown of about 1.2 and 2, respectively.

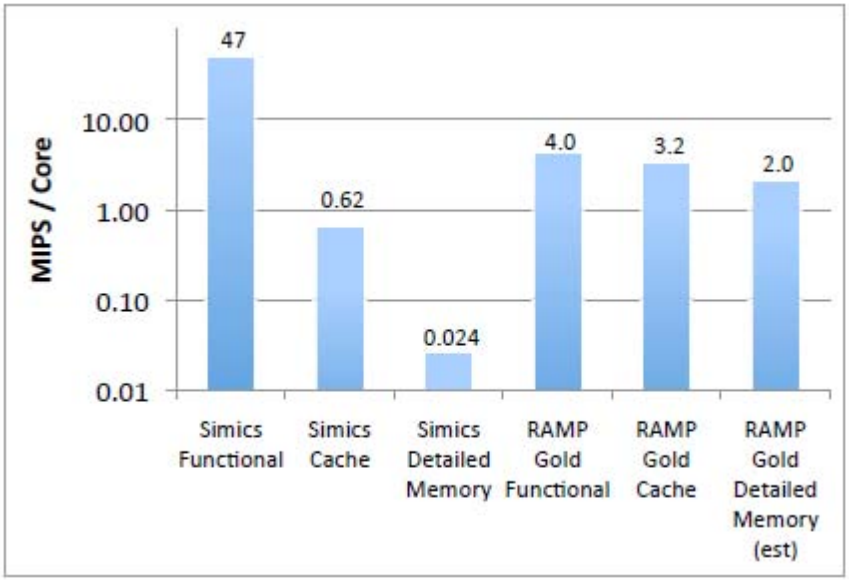

Figure 2: Three levels of functionality for Simics vs. RAMP

Note that this is the current performance of RAMP Gold for a single functional pipeline per FGPA, and we expect future versions of RAMP Gold will have 2 to 4 piplines per FPGA, which should improve performance considerably.

Figure 2 serves to illustrate the basic argument for FAME: Although functional simulation can be very fast, even faster than FAME simulation, the huge slowdown for the detailed level of SAME simulation allows FAME simulation to be a factor of 100 faster since it doesn't slow down nearly as quickly as the level of detail is increased.

11.0 FALLACIES AND PITFALLS

In discussions with others, we have heard some common misconceptions about FAME. We list them below as fallacies or pitfalls.

Fallacy: Because logic is relatively slow in FPGAs, which leads to a slow clock cycle, yet DRAMs run at full speed, you can't believe performance measurements from FAME simulators.

Although this criticism does apply to FPGA prototypes and Direct FAME, the primary reason for Decoupled and higher FAME levels is to overcome this exact weakness by separating timing simulation from functional simulation. Just as the speed of the host computer running the SAME simulator is obviously distinct from the modeled performance of the target machine being simulated, the clock rates of the host FPGA board and DRAMs are also obviously independent from target machine performance for Decoupled and higher-level FAME simulators.

Fallacy: You can't simulate out-of-order processors using FAME.

In fact, HASIM has a uniprocessor Out-Of-Order architectural model that runs the full SPEC2006 benchmark suite[4].

Fallacy: You can't run the operating system natively on FAME.

RAMP Gold is fully compatible with the SPARC V8 specification. It implements all types of instructions and traps required to boot an OS. The functional model design of RAMP Gold passes the SPARC V8 verification test suite required by the SPARC certification test. Currently, we are running a thin OS layer called the Proxy Kernel on RAMP Gold. The Proxy Kernel supports all system calls required by Newlib, a lightweight C library that is ABI-compatible with OpenSolaris. User-mode applications running on RAMP Gold can make I/O requests by calling standard C functions such as fread, fwrite, and printf. In the future, we plan to enhance the current RAMP Gold MMU to boot the Linux 2.6 kernel and bring up Debian user applications.

Fallacy: FAME is too expensive for most researchers to use.

At $750 for an XUP board, Multithreaded FAME costs less than a laptop, which makes it highly affordable, so almost anyone can afford to do parallel architecture research.

Fallacy: Surely industry must have solved this problem already.

Industry largely does evolutionary design. Hence, they heavily instrument current microarchitectures so that engineers can stress test real chips at high speeds to look for opportunities for improvements in subsequent versions. Note that industry's track record is not

[4] Joel Emer. Private communication, June 2009.

as good when creating new microarchitectures, such as the cancelled Intel Prescott, or new instruction sets, such as the HP/Intel Itanium. Given the multicore revolution, industry may not be able to rely on evolutionary design as much as it has in the past.

Pitfall: Use an FPGA prototype as the simulated target machine.

For FPGA prototypes or Direct FAME, people often map 'simplified' or 'FPGA optimized' RTLs due to implementation efficiency and target RTL availability. Further, people may use the implementation on FPGA as their simulated target architecture. However, those FPGA implementations are quite different from the real target. For example, a cache-coherent SMP target machine would usually employ a more advanced interconnect than a snooping bus, which is easiest to implement on FPGAs. In addition, the memory access latencies are significantly lower on FPGAs, because the processors on FPGAs run much slower than the host memory interface.

12.0 CONCLUSION AND RECOMMENDATIONS

Although Software Architectural Model Execution (SAME) simulators are important tools for computer systems research, we believe Multithreaded FPGA Architectural Model Execution (FAME) simulators offer the most cost-effective approach to exploring the multicore design space, as they provide the best combination of simulation speed and architectural flexibility [49]. As a specific example, RAMP Gold simulates 16 cores with a detailed memory model about 100 times faster than Simics+GEMS. Using architecture experiments per day per $1000 as a cost-performance metric within the FAME taxonomy, a Multithreaded FAME (Level 111) such as RAMP Gold is about 1,000,000 times better than Direct FAME (Level 000) such as Cadence Palladium, 1500 times better than Decoupled FAME (Level 001) such as Green Flash, and 20 times better than Abstract FAME such as HASIM (Level 011).

Like times past when sharing of simulators and useful simulation artifacts, such as traces or tools, lead to an increase in the number of researchers working on a problem, we hope that Multithreaded FAME simulators like RAMP Gold will increase the number working on the multicore challenge, for the IT industry certainly could use more help with the multicore challenge.

There are many opportunities and remaining obstacles for FAME to become useful. We conclude by listing them here:

As it is usually the case that the timing model circuitry is slower than functional simulation, and the functional model is underutilized, we would recommend arranging multiple timing models to share one functional pipeline.

As FPGAs get larger, to improve host simulation time without increasing the number of cores of the target architecture, we recommend adding multiple functional pipelines per FPGA. This change implies a cache coherency scheme for the host.

Given the relatively high bandwidth of host DRAM compared to the low host clock rates of FPGAs, we recommend simplifying the host memory hierarchy, to increase the number of pipelines per FPGA, thus reducing the difficulty of building coherency.

Both simulators and instruction-set-level virtual machines, such as those from Xen and VMware, run whole software stacks without the software being any the wiser. What techniques developed for Virtual Machines can be borrowed by FAME to improve cost performance or software compatibility?

While FPGAs allow emulation of many processors, there is no equivalent to the emulation of lots of memory. One idea to consider is using Flash memory as main memory. Another approach would be to borrow the separation of physical memory from machine memory from virtual machines to share identical memory pages between cores.

13.0 REFERENCES

[1] DDR2 DRAM Controller for BEE3, online at http://research.microsoft.com/en-us/projects/BEE3/, 2008.

[2] Pixie: MIPS Computer Systems, Inc. Assembly Language Programmer's Guide, 1986.

[3] Agarwal, Anant, Bianchini, Ricardo, Chaiken, David, David Kranz, David, Kubiatowicz, John, Lim, Benghong, Mackenzie, Kenneth, and Yeung, Donald. "The MIT Alewife Machine: Architecture and Performance", *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 2–13.

[4] Ahn, Jung Ho, Dally, William J., Khailany, Bruce, Kapasi, Ujval J., and Das, Abhishek. "Evaluating the Imagine Stream Architecture", *Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.

[5] Alameldeen, Alaa R., and Wood, David A., "Addressing Workload Variability in Architectural Simulations", *IEEE Micro*, **23(6)**, 2003, pp. 94–98.

[6] Alverson, Robert, Callahan, Callahan, Cummings, Daniel, Koblenz, Brian, Porterfield, Allan, and Smith, Burton, "The Tera Computer System". *Proceedings of the 4th International Conference on Supercomputing*, ACM, pp. 1–6, New York, NY, USA, 1990.

[7] Asanovic, Krste, Bodki, Ras, Catanzaro, Bryan Christopher, Gebis, Joseph James, Husbands, Parry, Keutzer, Kurt, Patterson, David A., Plishker, William Lester, Shalf, John, Williams, Samuel Webb, and Yelick, Katherine A., *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[8] Austin, Todd, Larson, Eric, and Ernst, Dan. "Simplescalar: An infrastructure for computer system modeling", *Computer*, **35(2**), 2002**,** pp. 59–67.

[9] Barr, Ken. *Summarizing Multiprocessor Program Execution with Versatile, Microarchitecture-Independent Snapshots*, PhD thesis, MIT, Sep 2006.

[10] Bienia, Christian, Kumar, Sanjeev, Singh, Jaswinder Pal, and Li, Kai. "The PARSEC Benchmark Suite: Characterization and Architectural Implications", *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ACM, pp. 72–81, New York, NY, USA, 2008.

[11] Bokhari, Shahid H., Mavriplis, Dimitri J., and Elton, Bracy H., *The Cray MTA and Unstructured Meshes*, Sep 2000.

[12] Brooks, David, Tiwari, Vivek, and Martonosi, Margaret, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ACM, pp. 83–94, New York, NY, USA, 2000.

[13] Minh, Chi Cao, Chung, JaeWoong, Kozyrakis, Christos, and Olukotun, Kunle, "STAMP: Stanford Transactional Applications for Multi-Processing", *Proceedings of The IEEE International Symposium on Workload Characterization*, Sep 2008.

[14] Chung, Eric S., Nurvitadhi , Eriko, Hoe , James C., Falsafi , Babak, and Mai , Ken, *ProtoFlex: FPGA-accelerated Hybrid Functional Simulation*, Technical Report CALCM 2007-2, ECE Department, Carnegie Mellon University, Feb 2007.

[15] Cmelik, Robert, and Keppel, David, "Shade: A Fast Instruction-Set Simulator for Execution Profiling", *ACM SIGMETRICS Performance Evaluation Review*, **22(1)**, May 1994, pp.128–137.

[16] Conte, Thomas M., Hirsch, Mary Ann, and Menezes, Kishore N., "Reducing State Loss For Effective Trace Sampling of Superscalar Processors", *Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*, IEEE Computer Society, 1996, pp. 468–477, Washington, DC, USA.

[17] Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., R. Vuduc, R., Whaley, R.C., and Yelick, K., "Self-adapting Linear Algebra Algorithms and Software", *Proceedings of the IEEE*, **93(2)**, Feb 2005, pp. 293–312.

[18] Edler, Jan, and Hill, Mark. Dinero IV Cache Simulator, online at http://www.cs.wisc.edu/~markhill/DineroIV.

[19] Emer, Joel, Adler, Michael, Parashar, Angshuman, Pellauer, Michael, and Vijayaraghavan, Murali, *RAMP/HAsim Status Update*, online at http://ramp.eecs.berkeley.edu/.

[20] Gschwind, Michael, Hofstee, H. Peter, Flachs, Brian, Hopkins, Martin, Watanabe, Yukio, and Yamazaki, Takeshi, "Synergistic Processing in Cell's Multicore Architecture", *IEEE Micro*, **26(2)**, 2006, pp. 10–24.

[21] Haskins, J. R., and Skadron, K., *Memory Reference Reuse Latency: Accelerated Sampled Microarchitecture Simulation*, Technical report 2002, Charlottesville, VA, USA.

[22] Hennessy, John L., Jouppi, Norman P., Gill, John, Baskett, Forest, Strong, Alex, Gross, Thomas R., Rowen, Christopher, and Leonard, Judson, "The MIPS Machine" Proceedings *of the 24th IEEE Computer Society International Conference*, 1982, pp. 2–7.

[23] Henning, John L., "SPEC CPU2006 Benchmark Descriptions" *SIGARCH Computer Architecture News*, **34(4)**, 2006, pp. 1–17.

[24] KleinOsowski, A J, and Lilja, David J., "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research" *IEEE Computer Architecture Letters*, **1(1)**, 2002, pp. 7.

[25] Kongetira, Poonacha, Aingaran, Kathirgamar, and Olukotun, Kunle, "Niagara: A 32-Way Multithreaded SPARC Processor", *IEEE Micro*, **25(2)**, 2005, pp. 21–29.

[26] Krashinsky, Ronny, Batten, Christopher, and Asanović, Krste, "Implementing the Scale Vector-thread Processor" *ACM Transactions on Design Automation of Electronic Systems*, **13((3)):41**, Jul 2008, pp. 1–41:24.

[27] Laha, S., Patel, J. H., and Iyer, R. K., "Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems" *IEEE Transactions on Computers*, **37(11)**, 1988, pp. 1325–1336.

[28] Larus, James "Spending Moore's Dividend" *Communications of the ACM*, **52(5**), 2009, pp. 62–69.

[29] Lenoski, Daniel, Laudon, James, Gharachorloo, Kourosh, Gupta, Anoop, and Hennessy, John, "The directory-based cache coherence protocol for the dash multiprocessor" *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ACM, 1990, pp. 148–159, New York, NY, USA.

[30] Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A. and Werner, B., "Simics: A full system simulation platform" *IEEE Computer*, **35**, 2002.

[31] Martin, Milo M. K., Sorin, Daniel J., Beckmann, Bradford M., Marty, Michael R., Xu, Min, Alameldeen, Alaa R., Moore, Kevin E., v Mark D., and Wood , David A., "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset" *SIGARCH Computer Architecture News*, **33(4)**,2005, pp. 92–99.

[32] Mukherjee, Shubhendu S., Reinhardt, Steven K., Falsafi, Babak, Litzkow, Mike, Hill, Mark D., Wood, David A., Huss-Lederman, Steven, and Larus, James R., "Wisconsin Wind Tunnel II: A Fast, Portable Parallel Architecture Simulator", *IEEE Concurrency*, **8(4)**, 2000, pp. 12–20.

[33] Nayfach-Battilana, and Renau, Jose, "SOI, Interconnect, Package, and Mainboard Thermal Characterization", *Proceedings of the 14th International Symposium on Low Power Electronics and Design*, August 2009.

[34] Pai, Vijay S., Ranganathan, Parthasarathy, and Adve, Sarita V., *RSIM Reference Manual. Version 1.0.* Technical Report 9705, Department of Electrical and Computer Engineering, Rice University, July 1997.

[35] Patterson, David, Gibson, Garth, and Katz, Randy, "A Case for Redundant Arrays of Inexpensive Disks (RAID)" *Proc. of the ACM SIGMOD International Conference on Management of Data*, Jun 1988.

[36] Patterson, David A. and Séquin, Carlo H, "RISC I: A Reduced Instruction Set VLSI Computer", *Proceedings of the 8th Annual International Symposium on Computer Architecture*, 1981, pp 443–458.

[37] Rosenblum, Mendel, Bugnion, Edouard, Devine, Scott, and Herrod, Stephen A., "Using the SimOS machine simulator to study complex computer systems", *ACM Transactions on Modeling and Computer Simulation*, 1997, **7(1)**, pp. 78–103.

[38] Sankaralingam, Karthikeyan, Nagarajan, Ramadass, McDonald, Robert, Desikan, Rajagopalan, Drolia, Saurabh, Govindan, M. S., Gratz, Paul, Gulati, Divya, Hanson, Heather, Kim, Changkyu, Liu, Haiming, Ranganathan, Nitya, Sethumadhavan, Simha, Sharif, Sadia, Shivakumar, Premkishore, Keckler, Stephen W., and Burger, Doug, "Distributed microarchitectural protocols in the TRIPS prototype processor" *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 480–491, Washington, DC, USA.

[39] Seiler, Larry, Carmean, Doug, Sprangle, Eric, Forsyth, Tom, Abrash, Michael, Dubey, Pradeep, Junkins, Stephen, Lake, Adam, Sugerman, Jeremy, Cavin, Robert, Espasa, Roger, Grochowski, Ed, Juan, Toni, and Hanrahan, Pat, "Larrabee: a Many-core x86 Architecture for Visual Computing", *ACM SIGGRAPH 2008*, ACM, 2008, pp. 1–15, New York, NY, USA.

[40] Sherwood, Timothy, Perelman, Erez, Hamerly, Greg, and Calder, Brad, "Automatically Characterizing Large Scale Program Behavior", *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ACM, 2002, pp. 45–57, New York, NY, USA.

[41] Swanson, Steven, Putnam, Andrew, Mercaldi, Martha, Michelson, Ken, Petersen, Andrew, Schwerin, Andrew, Oskin, Mark, and Eggers, Susan J., "Area-Performance Trade-offs in Tiled Dataflow Architectures", *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, IEEE Computer Society, 2006, pp. 314–326, Washington, DC, USA.

[42] Tan, Zhangxi, Asanovic, Krste, and Patterson, David, "An FPGA Host-Multithreaded Functional Model for SPARC V8", 3rd Workshop on Architectural Research Prototyping (WARP-2008), *Proceedings of the 35th International Symposium on Computer Architecture* , Jun 2008.

[43] Tan, Zhangxi, Waterman, Andrew, Avizienis, Rimas, Lee, Yunsup, Patterson, David, and Asanovic, Krste, "RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors", 4th Workshop on Architectural Research Prototyping (WARP-2009), *Proceedings of the 36th International Symposium on Computer Architecture* , Jun 2009.

[44] Taylor, M. B., Lee, Miller, W., Wentzlaff, J., D., . Bratt, I., Greenwald, B., Hoffmann, H., Johnson, P., Kim, J., Psota, J., Saraf, A., Shnidman, N., Strumpen, V., Frank M., Amarasinghe, S., and Agarwal, A., "Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and Streams". *Proceedings of the 31st International Symposium on Computer Architecture* , Jun 2004.

[45] Wawrzynek, John, Patterson, David A., Oskin, Mark, Lu, Shih-Lien, Kozyrakis, Christoforos E., Hoe, James C., Chiou, Derek, and Asanovic, Krste, "RAMP: Research Accelerator for Multiple Processors", *IEEE Micro*, 2007 **27(2)**, pp. 46–57.

[46] Wehner,, Michael, Oliker, Leonid, and Shalf, John, "Towards Ultra-High Resolution Models of Climate and Weather", *International Journal of High Performance Computing Applications*, 2008, **22(2)**, pp. 149–165.

[51] Wenisch, Thomas F., Wunderlich, Roland E., Falsafi, Babak, and Hoe, James C., "TurboSMARTS: Accurate Microarchitecture Simulation Sampling in Minutes", *SIGMETRICS Performance Evaluation Review*, 2005, **33(1)**: pp. 408–409.

[47] Wenisch, Thomas F., Wunderlich, Roland E., Ferdman, Michael, Ailamaki, Anastassia, Falsafi, Babak, and Hoe, James C., "SimFlex: Statistical Sampling of Computer System Simulation", *IEEE Micro*, 2006, **26(4)**, pp. 18–31.

[48] Witchel, Emmett, and Rosenblum, Mendel, "Embra: Fast and Flexible Machine Simulation" *SIGMETRICS Performance Evaluation Review*, 1996, **24(1)**, pp. 68–79.

[49] Woo, Steven Cameron, Ohara, Moriyoshi, Torrie, Evan, Singh, Jaswinder Pal, and Gupta, Anoop, "The SPLASH-2 programs: Characterization and Methodological Considerations", *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, ACM, 1995, pp. 24–36, New York, NY, USA.

14.0 ACKNOWLEDGEMENTS

We are grateful to the Department of Defense for supporting this study. It was funded through the Information Processing Techniques Office of DARPA, Dr. William Harrod, program manager, and administered by the Air Force Research Laboratory, Kerry L. Hill, project engineer. Thanks also to Jon Hiller and Sherman Karp for their feedback on our work.

The RAMP collaboration has been funded in part by the National Science Foundation, grant number CNS-0551739. Special thanks to Xilinx for their continuing financial support and donation of FPGAs, and development tools. We appreciate the financial support provided by the Gigascale Systems Research Center (GSRC). Thanks to IBM for their financial support through faculty fellowships and donation of processor cores, and to Sun Microsystems for processor cores. Much of the work on RAMP at UC Berkeley now takes place in the Par Lab and is supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227), and by donations from National Instruments, NEC, Nokia, NVIDIA, and Samsung.

This project has been a collaborative effort between universities and companies. The other senior investigators are David Patterson (UC Berkeley), Mark Oskin (U Washington), Shih-Lien Lu (Intel), Christoforos Kozyrakis (Stanford), James C. Hoe (CMU), Derek Chiou (UT Austin), and Joel Emer (Intel and MIT). There is an extensive list of industry and academic friends who have given valuable feedback and guidance. Here we especially give thanks to Arvind (MIT) and Jan Rabaey (UCB) for their advice. The work presented in this report is the effort of the RAMP students and staff: Dan Burke, Greg Gibeling, Zhangxi Tan, and Andrew Waterman.

We would like to thank the following people who provided feedback on drafts of this report: John Davis (Microsoft, who suggested the bit-vector to distinguish FAME levels), Kees Vissers (Xilinx), and Chen Chang (BEEcube).