RAMP: A Research Accelerator for Multiple Processors

John Wawrzynek
Mark Oskin
Christoforos Kozyrakis
Derek Chiou
David A. Patterson
Shih-Lien Lu
James C. Hoe
Krste Asanovic

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-158
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-158.html

November 24, 2006

RAMP: A Research Accelerator for Multiple Processors

John Wawrzynek (UC Berkeley) David Patterson (UC Berkeley)
Mark Oskin (U Washington) Shih-Lien Lu (Intel)
Christoforos Kozyrakis (Stanford) James C. Hoe (CMU)
Derek Chiou (UT Austin) Krste Asanovic (MIT)

I. Introduction

In 2005 there was a historic change of direction in the computer hardware industry: All microprocessor companies announced that their future products would be single-chip multiprocessors, and that future performance improvements would rely on software-specified parallelism rather than additional software-transparent parallelism extracted automatically by the microarchitecture. Several of us discussed this milestone at the ISCA conference in June 2005. We were struck that a multibillion-dollar industry would bet their future on solving the general-purpose parallel computing problem, when so many have previously attempted but failed to provide a satisfactory approach.

To tackle the parallel processing, our industry urgently needs innovative solutions that requires extensive co-development of hardware and software. However, the rate of such innovation is currently slowed by the following traditional development cycle:

1) It takes approximately four years and many millions of dollars to prototype a new architecture in hardware, even at only research quality.
2) Software engineers are ineffective until the new hardware actually shows up since simulators are too slow to support serious software development activities. Software engineers tend to innovate only *after* hardware arrives.
3) Feedback from software engineers based on the current production hardware cannot impact the immediate next generation due to overlapped hardware development cycles. Instead, the feedback loop can take several hardware generations to close fully.

Hence, we conspired on how to create an inexpensive, reconfigurable, highly-parallel platform that would attract researchers from many disciplines (architecture, compilers, operating systems, applications, etc.) to work together on perhaps the biggest challenge facing computing in the past fifty years. Our goal was a platform that would allow far more rapid evolution than traditional approaches as solutions are desperately needed.

II. RAMP Vision

Our hallway conversations led us to the idea of using Field-Programmable Gate Arrays (FPGAs) to emulate highly parallel architectures at hardware speeds. FPGAs enable very rapid turnaround for new hardware. You can ”tape out” a FPGA design every day, with a new system ”fabricated” overnight. Another key advantage of FPGAs is that they easily exploit Moore’s Law. As the number of cores per microprocessor die grows, FPGA density will grow at about the same rate. Today we can map about 16 simple processors onto a single FPGA, which means we can construct a 1000-processor system in just 64 FPGAs. Such a system is cheaper and lower power than a custom multiprocessor at about $100 and about 1 Watt per processor.

We named this project RAMP (*R*esearch *A*ccelerator for *M*ultiple *P*rocessors) since its goal is to ramp up the rate of innovation in hardware and software multiprocessor research. RAMP is an open-source project to develop and share the hardware and software necessary to create parallel architectures. RAMP is not just

a hardware architecture project. Perhaps our most important goal is to support the software community as it struggles to take advantage of the potential capabilities of parallel microprocessors, by providing a malleable platform through which the software community can collaborate with the hardware community.

Unlike commercial multiprocessor hardware, RAMP is designed as a research platform. We can include research features that would be impractical or impossible to include in real hardware systems due to speed, cost or practicality issues. For example, additional hardware to monitor any event in the system can be incorporated into the FPGA design. Being able to add arbitrary event probes, including arbitrary computation on those events, provides visibility formerly only available in software simulators, but without the inevitable slowdown faced by software simulators when introducing such visibility.

A second example of how RAMP is different than real hardware is reproducibility. Using the RDL framework described in Section III, different researchers can construct the same deterministic parallel computing system that will perform exactly the same way every time, clock cycle for clock cycle. By using processor designs donated by industry, RAMP users can start with familiar architectures and operating systems, providing much more credibility than software simulations that model idealized processors or that ignore operating system effects. RDL is designed to make constructing a full computer out of RDL-compatible modules easy. Our target speeds of 100 to 200 MHz are slower than real hardware, but are fast enough to run standard operating systems and large scale applications orders of magnitude faster than software simulators. Finally, due to the similarities in the design flow of logic for FPGAs and custom hardware, we believe RAMP is realistic enough to convince software developers to start aggressive development on innovative architectures and programming models and to convince hardware and software companies that RAMP results are relevant.

We believe this combination of cost, power, speed, flexibility, observability, reproducibility, and credibility will make the platform attractive to software and hardware researchers interested in the parallel challenge. In particular, it allows the research community to revive the 1980's culture of building experimental hardware and software systems, which has been mostly lost due to the higher cost and difficulty of building hardware today.

Table I compares alternatives for pursuing parallel systems research in academia. The four options are a conventional shared-memory multiprocessor (SMP), a cluster, a simulator, building a custom chip and system, and RAMP. The rows are the features of interest. Cost rules out a large SMP for most academics. The cost of purchase and cost of ownership makes a large cluster too expensive for most academics as well. Our only alternative to date has been software simulation, and indeed that has been the vehicle for most architecture research in the last decade. As mentioned above, software developers rarely use software simulators since they run too slowly and results might not be credible. In particular, it's unclear how credible results will be to industry, when based on simulations of 1000 processors running small snippets of applications. The RAMP option is a compromise between these alternatives. It is so much cheaper than custom hardware that academics can afford highly scalable systems; it is as flexible as simulators so that we can rapidly evolve the state of the art in parallel computing; and it is so much faster than simulators that software people can be tempted to actually try out a new hardware idea.

This speed also allows architects to explore a much larger space in their research, and then to do a more thorough evaluation of their proposals. Although architects can achieve high "batch" simulation throughput using multiple independent software simulations distributed over a large compute cluster, this does not reduce the latency to obtain a single key result that can move the research forward. It also does not help an application developer trying to debug the port of an application to the new target system (the emulated machine is called the *target*, and underlying FPGA hardware is the *host*.) Worse, for multiprocessor targets, simulation speed in both instructions per second per core and total instructions per second drops as more cores are simulated and as operating system effects are included, and the amount of memory required for each node in the host compute cluster rises rapidly.

We believe the upside potential is so compelling that RAMP could create a "watering hole" effect in academic departments, as people from many disciplines would use RAMP in their research. RAMP is obviously attractive to hardware and software researchers in parallelism, but it's also excellent for those

TABLE I

RELATIVE COMPARISON OF FOUR OPTIONS FOR PARALLEL RESEARCH. FROM THE ARCHITECT'S PERSPECTIVE, THE MOST SURPRISING ASPECT OF THIS TABLE IS THAT PERFORMANCE IS NOT ONLY THE TOP CONCERN; IT IS LAST IN THIS LIST. IT JUST NEEDS TO BE FAST ENOUGH TO RUN THE WHOLE SOFTWARE STACK.

	SMP	Cluster	Simulate	Custom	RAMP
Scalability (1k CPUs)	C	A	A	A	A
Cost (1k CPUs)	F ($40M)	C ($2-3M)	A+ ($0M)	F ($20M)	A ($0.1-0.2M)
Cost of ownership	A	D	A	D	A
Power/Space (kilowatts, racks)	D (120, 12)	D (120, 12)	A+ (.1, 0.1)	A	B (1.5, 0.3)
Development Community	D	A	A	F	B
Observability	D	C	A+	A	B+
Reproducibility	B	D	A+	A	A+
Reconfigurability	D	C	A+	C	A+
Credibility of Result	A+	A+	D	A+	B+/A-
Performance (clock)	A (2 GHz)	A (3 GHz)	F (0 GHz)	B (0.4 GHz)	C (0.1 GHz)
Modeling Flexibility	D	D	A	B	A
GPA	C	C+	B	B-	A-

interested in topics as varied as security, dependability, and even data center emulation. Such a watering hole would lead to conversations between disciplines that rarely talk to each other, helping us to more quickly develop multiprocessor systems that are easy-to-program efficiently. Indeed, to help industry win its bet on parallelism, we will need the help of many people, for the parallel future is not just an architecture change, but likely a change to the entire software ecosystem.

The rest of this paper highlights the key current developments in RAMP project. Section III describes the RAMP Design Framework which enables the assembly of disparate hardware components on top of the underlying FPGA emulation substrate. Section IV introduces the three reference designs, RAMP Red (transactional memory), RAMP Blue (distributed systems) and RAMP White (distributed shared memory). They are representative of complete designs as well as vehicles for demonstrating the various reusable modules under development. Section V provides a summary and our conclusions.

III. RAMP DESIGN FRAMEWORK

From the earliest stages of the RAMP project, it was clear that a standardized design framework was needed to enable a large community of users to cooperate and build a useful library of inter-operable hardware models. This design framework has a number of challenging goals. It must support both cycle-accurate emulation of detailed parameterized machine models and rapid functional-only emulations. The design framework should hide the details in the underlying FPGA emulation substrate from the module designer as much as possible, to allow groups with different FPGA emulation hardware to share designs and to allow RAMP modules to be reused after FPGA emulation hardware upgrades. In addition, the design framework should not dictate the hardware design language (HDL) chosen by developers. Our approach was to develop a decoupled machine model and design discipline. This discipline is enforced by the RAMP Description Language (RDL) and a compiler to automate the difficult task of providing cycle-accurate emulation of distributed communicating components [8].

The RAMP design framework is based on a few central concepts. A RAMP target model is a collection of loosely coupled target *units* communicating with latency-insensitive protocols over well-defined target *channels*. Figure 1 gives a simple schematic example of two connected units. In practice, a unit will be a large component corresponding to tens of thousands of gates of emulated hardware, e.g., a processor with

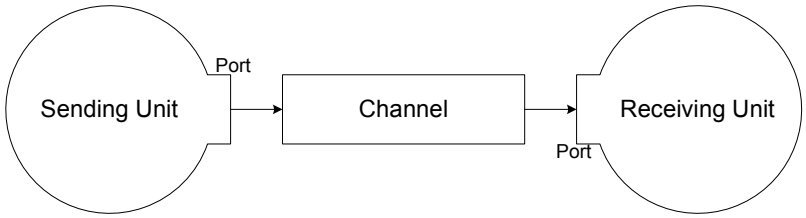

Fig. 1. Basic RAMP communication model.

L1 cache, a DRAM controller, or a network router stage. All communication between units is via messages sent over unidirectional point-to-point inter-unit *channels*, where each channel is buffered to allow units to execute decoupled from each other.

Each unit faithfully models the behavior of each target clock cycle in the component. The target unit models can be developed either as RTL code in a standard HDL (currently Verilog, VHDL, and Bluespec are supported) for compilation onto the FPGA fabric, or as software models that execute either on attached workstations or on hard or soft processor cores embedded within the FPGA fabric. Many target units taken from existing RTL code will execute a single target clock cycle in one FPGA physical clock cycle, giving a high simulation speed. However, to save FPGA resources, a unit model can be designed to take multiple physical host clock cycles on the FPGA to emulate one target clock cycle, or might even use a varying number of physical clock cycles. Initially, the whole RAMP host system uses the same physical clock rate (nominally around 100 MHz), with some higher physical clock rates in off-chip I/O drivers.

Unit models are only synchronized via the point-to-point channels. The basic principle is that a unit cannot advance by a target clock cycle until it has received a target clock cycle's worth of activity on each input channel and the output channels are ready to receive another target cycle's worth of activity. This scheme forms a distributed concurrent event simulator, where the buffering in the channels allows units to run at varying physical speeds on the host while remaining logically synchronized in terms of target clock cycles.

Unit model designers must produce the RTL code (or "gateware") of each unit in their chosen HDL, and specify the range of message sizes that each input or output channel can carry. For each supported HDL, the RAMP design framework provides tools to automatically generate a unit wrapper that interfaces to the channels and provides target cycle synchronization. The RTL code for the channels is generated automatically by the RDL compiler from a RDL description, which includes a structural netlist specifying the instances of each unit and how they are connected by channels.

The benefit of enforcing a standard channel-based communication strategy between units is that many features can be provided automatically. Users can vary the target latency, target bandwidth, and target buffering on each channel at configuration time. The RAMP configuration tools will also provide the option to have channels run as fast as the underlying physical hardware will allow to support fast functional-only emulation. We are also exploring the option of allowing these parameters to be changed dynamically at target system boot time to avoid re-running the FPGA synthesis flow when varying parameters for performance studies.

The configuration tool will include support for inter-unit channels to be tapped and controlled to provide monitoring and debugging facilities. For example, by controlling stall signals from the channels, a unit can be single stepped. Using a separate automatically-inserted debugging network, invisible to target system software, messages can be inserted and read out from the channels entering and leaving any unit, and all significant events can be logged. We believe these monitoring and debugging facilities will provide significant advantages compared with running applications on commercial hardware.

IV. RAMP Prototypes

Though RAMPants are all volunteers, the project is on a fairly aggressive schedule. Figure IV shows the timeline for the RAMP project. We began RAMP development using pre-existing FPGA boards—see

June 6, 2005	Hallway discussions lead to RAMP vision
June 13, 2005	The name "RAMP" coined; BEE2 [6] selected as RAMP-1; a dozen people identified to develop RAMP
January 2006	RAMP retreat and RDL tutorial at Berkeley
March 2006	NSF infrastructure grant awarded
June 2006	RAMP retreat at MIT; RAMP Red running with 8 processors on RAMP-1 boards
January 2007	RAMP Blue running with 64 to 128 processors on 8 RAMP-1 boards
June 2007	RAMP Red, White, and Blue running with 128 to 256 processors on 16 RAMP-1 boards; accurate clock cycle accounting and I/O model
December 2007	RAMP-2 boards redesigned based on Virtex-5 and available for purchase; RAMP web site has downloadable designs

Fig. 2. RAMP Timeline

side-bar on RAMP hardware. To seed the collaborative effort, we are developing three prototype systems, named *RAMP Red*, *RAMP Blue*, and *RAMP White*. Each of our initial prototypes contains a complete gateware/software configuration of a scalable multiprocessor populated with standard processor cores, switches, and operating systems. Once the base system is assembled and software installed, users will be able to easily run complex system benchmarks, and then modify this working system as desired or start from the ground up using the basic components to build a new system. We expect that users will release back to the community any enhancements and new gateware/software modules. A similar usage model has led to the proliferation of the SimpleScalar framework which now covers a range of instruction sets and processor designs.

A. *RAMP Red*

RAMP Red is the first multiprocessor system with hardware support for transactional memory (TM). Transactional memory transfers the responsibility for concurrency control from the programmer to the system [1]. It introduces database semantics to the shared memory in a parallel system, which allows software tasks (transactions) to execute atomically and in isolation without the use of locks. Hardware support for TM reduces the overhead of detecting and enforcing atomicity violations between concurrently executing transactions and guarantees correct execution under all cases.

RAMP Red implements the Stanford TCC architecture for transactional memory [9]. The design uses 9 PowerPC 405 hardcores (embedded in the Xilinx Virtex-II-Pro FPGAs) connected to a shared main memory system through a packet-switched network. The built-in data cache in each PowerPC 405 core is disabled and replaced by a custom cache (emulated in FPGA) with transactional memory support. Each 32-KByte cache buffers the memory locations read and written by a transaction during its execution and detects atomicity violations with other on-going transactions. An interesting feature of RAMP Red is the use of a transaction completion mechanism that eliminates the need for a conventional cache coherence protocol.

From an application's point of view, RAMP Red is a fully featured Linux workstation. The operating system is actually running on just one of the cores while the remaining 8 cores are used for application execution. A light-weight kernel in each application core forwards exceptions and system calls to the OS core. The programming model is multithreaded C or Java with locks replaced by transactional constructs. RAMP Red includes an extensive hardware/software framework for debugging, bottleneck identification, and performance tuning.

The RAMP Red design has been fully operational since June 2006. It runs at 100 MHz on RAMP-1, which is 100 times faster than the same architecture simulated in software on a 2 GHz workstation. Early experiments with enterprise, scientific, and artificial intelligence applications have demonstrated the simplicity of parallel programming with transactional memory and that RAMP Red achieves scalable performance. In the future, RAMP Red will be the basis for further research in transactional memory, focusing mostly on software productivity and system software support [5].

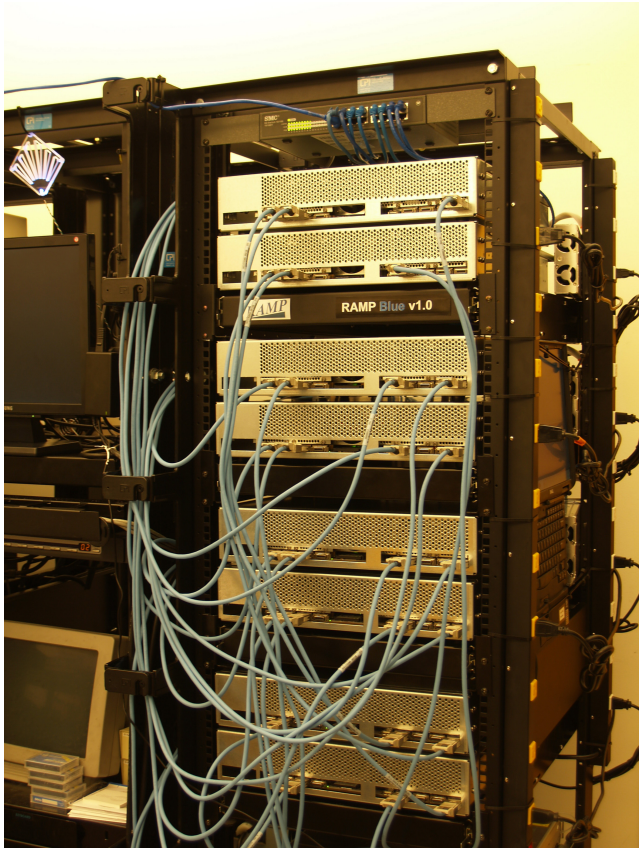

Fig. 3. Photograph of RAMP Blue Prototype.

B. RAMP Blue

RAMP Blue is a family of emulated message-passing machines, which can be used to run parallel applications written for the Message-Passing Interface (MPI) standard, or for partitioned global address space languages such as Unified Parallel C (UPC). RAMP Blue can also be used to model a networked server cluster.

The first RAMP Blue prototype is currently under development at UC Berkeley. The hardware platform for this prototype is shown in Figure 3. It comprises a collection of RAMP-1 boards housed in 2U chassis and assembled in a standard 19" rack. Physical connection among the eight boards is through 10 Gbps Infiniband cables (light blue cables in the photograph). The RAMP-1 boards are wired in an all-to-all configuration with a direct connection from each board to all others through 10 Gbps links. System configuration, debugging, and monitoring is through a 100 Mbps Ethernet switch with connection to the control FPGA of each board (dark blue wires in the photograph). For system management and control, each board runs a full-featured Linux kernel on one PowerPC 405 hardcore embedded in the control FPGA. Our initial target applications are the UPC version of the NAS Parallel Benchmarks.

The four user FPGAs per RAMP-1 board are configured to hold a collection of 100 MHz Xilinx MicroBlaze soft processor cores running uCLinux. We have mapped eight processor cores per FPGA. The first prototype, with 32 user FPGAs, will emulate a 256-way cluster system. In the future, the number of processor cores can still be scaled up through several means. More RAMP-1 boards will be added—the simple all-to-all wiring configuration will accommodate up to 17 boards. More cores will be added per FPGA—the current configuration of eight processor cores per FPGA only consumes 40% of the FPGA's logic resources. Overall, we expect to reach 16 MicroBlaze cores per FPGA, and 1024 cores in a system.

All necessary multiprocessor components are implemented within the user FPGAs. In addition to the soft processor cores, each FPGA holds a packet network switch (one for each core) for connection to cores on

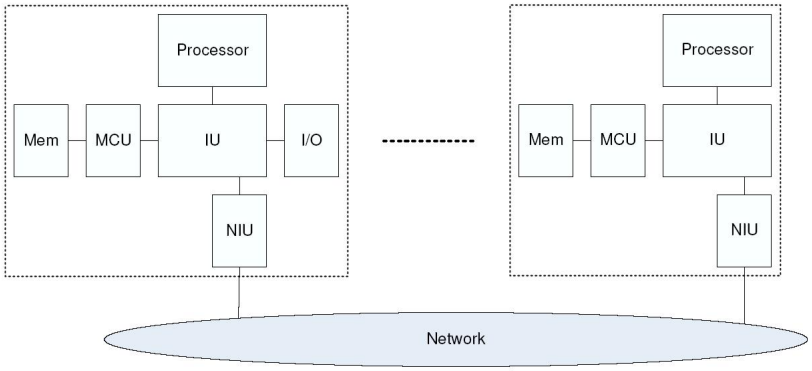

Fig. 4. A High-Level View of RAMP White

the same and other FPGAs, shared memory controllers, shared double-precision floating-point units (FPUs), and a shared "console" switch for connection to the control FPGA.

In RAMP Blue, each processor is assigned its own DRAM memory space (at least 250 MB per processor). The external memory interface of the MicroBlaze L1 cache connects to external DDR2 DRAM through a memory arbiter, as each DRAM channel is shared among a set of MicroBlaze cores. Since each RAMP-1 user FPGA has four independent DRAM memory channels, four processor cores would share one channel in the maximum-sized configuration (16 processor cores per FPGA). With each processor running at 100 MHz and each memory channel running a 200 MHz DDR 72-bit data interface, each processor can transfer 72 bits of data at 100 MHz, which is more than each processor core can consume even in our maximum-sized configuration. A simple round-robin scheme is used to arbitrate among the cores.

The processor–processor network switch currently uses a simple interrupt-driven programmed I/O approach. A Linux driver provides an Ethernet interface so applications can access the processor network via traditional socket interfaces. We are planning a next generation network interface with direct memory access through special ports into the memory controller.

A 256-core (8 per FPGA) version of the RAMP blue prototype is currently nearing completion. At present, all cores successfully run uCLinux and communicate over the "console" switch with the control FPGAs. *As of this writing, the processor–processor network switch is not fully functional, however, several UPC NAS Parallel Benchmark programs have been run successfully on a limited number of cores. We expect the system will be fully functional soon and will report on its operation running the benchmark suite in the final version of the paper.*

C. RAMP White

RAMP White is a distributed shared memory machine that will demonstrate the open component nature of RAMP by integrating modules from RAMP Red, RAMP Blue and individual RAMPants contributions. The initial version is being designed and integrated at the University of Texas at Austin. The RAMP White effort started in the summer of 2006, somewhat after Red and Blue, and will be implemented in the phases listed below.

1) Global distributed shared memory without caches. All requests to remote global memory will be serviced directly from remote memory. Communication will be performed over a ring network.
2) Ring-based snoopy coherency. The basic infrastructure of the cache-less system will be expanded to include a snoopy cache that will snoop the ring.
3) Directory-based coherency. A directory-based coherence engine eliminates the need for each cache to snoop all transactions but will use the same snoopy cache.

RAMP White will eventually be composed of processor units from the University of Washington and RAMP Blue teams that will be connected by a simple ring network (Figure 4.) For expediency, the initial

RAMP White will use the embedded PowerPC processors. Each processor unit contains one processor that is connected to an Intersection Unit (IU) that provides connections to a memory controller (MCU), a network interface unit (NIU), and I/O if the processor unit supports it. The NIU will be connected to a simple ring network.

The IU switches requests and replies between the processor, local memory, I/O and the network. The initial IU is very simple. Memory requests from the processor are divided into local memory requests, global memory requests (both handled by memory), I/O requests (handled by the I/O module) and remote requests (handled by the NIU). Remote requests from the NIU are forwarded to the memory. Since the initial version of RAMP White does not cache global locations, incoming remote requests do not need to be snooped by the processor.

I/O will be handled by a centralized I/O subsystem mapped into the global address space. Each processor will run a separate SMP-capable Linux that will take locks to access I/O. The global memory support then transparently handles shared I/O. Later versions will add a coherency support using a soft cache (emulated in FPGA). RAMP White's first snoopy cache will be based on RAMP Red's snoopy cache. It is possible that some or all of the data in the emulated cache will actually reside in DRAM if there is not sufficient space in the FPGA itself. In the coherent versions of RAMP White, the IU passes all incoming remote requests to the coherent cache for snooping before allowing the remote request to proceed to the next stage.

V. Conclusions

This paper presented RAMP, an open-source, community developed, FPGA-based emulator of parallel architectures. We presented the RAMP design framework to enable an emulator comprising reusable and composable design modules to be maintained and developed by a large collaborative community. We highlighted the on-going development of three reference full-system designs. We are planning a full public release of the RAMP infrastructure described in this paper in 2007.

RAMP is a return to building hardware/software systems by the research community. Hardware architecture, operating system, compiler, application and programming model research will all benefit. RAMP is designed to be the right cost/performance/density/visibility tradeoffs for system research. Moreover, since the system is not frozen, we can use it to both rapidly evolve and spread successful ideas across the community.

Acknowledgments

This work was 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. 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 paper are the efforts by the RAMP students and staff: Hari Angepat, Dan Burke, Jared Casper, Chen Chang, Pierre-Yves Droz, Greg Gibeling, Alex Krasnov, Martha Mercaldi, Nju Njoroge, Andrew Putnam, Andrew Schutlz, and Sewook Wee.

References

[1] Ali-Reza Adl-Tabatabai, Christos Kozyrakis, and Bratin Saha. Tutorial: Transactional programming in a multi-core environment. Tutorial at the 15th International Conference on Parallel Architecture and Compilation Techniques (PACT), September 2006.

[2] Nathan L. Binkert, Ronald G. Dreslinski, Lisa R. Hsu, Kevin T. Lim, Ali G. Saidi, and Steven K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

[3] Nathan L. Binkert, Erik G. Hallnor, and Steven K. Reinhardt. Network-oriented full-system simulation using M5. In *Proceedings of Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, February 2003.

[4] Doug Burger and Todd M. Austin. The simplescalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin, Madison, June 1997.

[5] Brian D. Calrstrom, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. The software stack for transactional memory: Challenges and opportunities. In *Proc. 1st Workshop on Software Tools for Multicore Systems (STMCS) at CGO-4*, March 2006.
[6] C. Chang, J. Wawrzynek, and R. W. Brodersen. BEE2: A High-End Reconfigurable Computing System. *IEEE Design and Test of Computers*, 22(2):114–125, Mar/Apr 2005.
[7] Joel Emer, Pritpal Ahuja, Eric Borch, Artur Klauser, Chi-Keung Luk, Srilatha Manne, Shubbendu S. Mukherjee, Harish Patil, Steven Wallace, Nathan Binkert, Roger Espasa, and Toni Juan. Asim: A performance model framework. *IEEE Micro*, Feb 2002.
[8] G. Gibeling, A. Schultz, and K. Asanović. RAMP architecture and description language. In *2nd Workshop on Architecture Research using FPGA Platforms, HPCA-12*, February 2006.
[9] Lance Hammond, Brian Carlstrom, Vicky Wong, Ben Hertzberg, Mike Chen, Christos Kozyrakis, and Kunle Olukotun. Programming with transactional coherence and consistency (tcc). In *Proc. of the 11th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, June 2006.
[10] A. KleinOsowski and D. Lilja. Minnespec: A new spec benchmark workload for simulation-based computer architecture research, 2002.
[11] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, 2005.
[12] K. Oner, L. A. Barroso, S. Iman, J. Jeong, K. Ramamurthy, and M. Dubois. The Design of RPM: An FPGA-based Multiprocessor Emulator. In *Proc. 3rd ACM International Symposium onField-Programmable Gate Arrays (FPGA'95)*, February 1995.
[13] Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve. Rsim reference manual. version 1.0. Technical Report 9705, Electrical and Computer Engineering Department, Rice University, July 1997.
[14] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta. Complete computer system simulation: The SimOS approach. *IEEE parallel and distributed technology: systems and applications*, 3(4):34–43, Winter 1995.
[15] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications, 2001.
[16] Simflex: Fast, accurate & flexible computer architecture simulation. http://www.ece.cmu.edu/ simflex/flexus.html.
[17] Virtutech. Simics.
[18] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling, 2003.

VI. SIDEBAR: RAMP HARDWARE

Rather than begin the RAMP project by designing yet another FPGA board, we have adopted the Berkeley Emulation Engine (BEE2 [6]) for the RAMP-1 system. BEE2 boards serve as a platform of the first RAMP machine prototypes and to help us understand our wish list of features for the next generation board. The RAMP-2 system, currently in design, will be based on a new board design employing the recently announced Virtex-5 FPGA architecture.

The BEE2 Compute Module is shown in Figure 5. Each compute module consists of five Xilinx Virtex-2 Pro-70 FPGA chips each directly connected to four DDR2 240-pin DRAM DIMMs, with a maximum capacity of 4 GB per FPGA. The four DIMMs are organized into four independent DRAM channels, each running at 200 MHz (400DDR) with a 72-bit data interface. Therefore, peak aggregate memory bandwidth is 12.8 GBps per FPGA.

The five FPGAs on the same module are organized into four compute FPGAs and one control FPGA. The control FPGA has additional global interconnect interfaces and control signals to the secondary system components. The connectivity on the compute module can be classified into two classes: on-board LVCMOS connections and off-board Multi Gigabit Transceiver (MGT) connections. The local mesh connects the four compute FPGAs on a two-by-two 2D grid. Each link between the adjacent FPGAs on the grid provides over 40 Gbps data throughput per link. The four down links from the control FPGA to each of the computing FPGAs provide up to 20 Gbps per link. These direct FPGA-to-FPGA mesh links form a high-bandwidth low-latency mesh network for the FPGAs on the same compute module, so all five FPGAs can be aggregated to form a virtual FPGA with five times the capacity.

All off-module connections use the MGTs on the FPGA. Each individual MGT channel is configured in software to run at 2.5 Gbps or 3.125 Gbps with 8B/10B encoding, and every 4 MGTs are channel-bonded into a physical Infiniband 4X (IB4X) electrical connector, to form a 10 Gbps full duplex (20 Gbps total) interface. The IB4X connections are AC coupled on the receiving end to comply with the Infiniband and 10GBase-CX4 specification.

Using the 4X Infiniband physical connections, the compute modules can be wired into many network topologies, such as a 3D mesh. For applications requiring high bisection bandwidth random communication among many compute modules, the BEE2 system is designed to take advantage of commercial network switch technology, such as Infiniband or 10G Ethernet. The regular 10/100Base-T Ethernet connection,

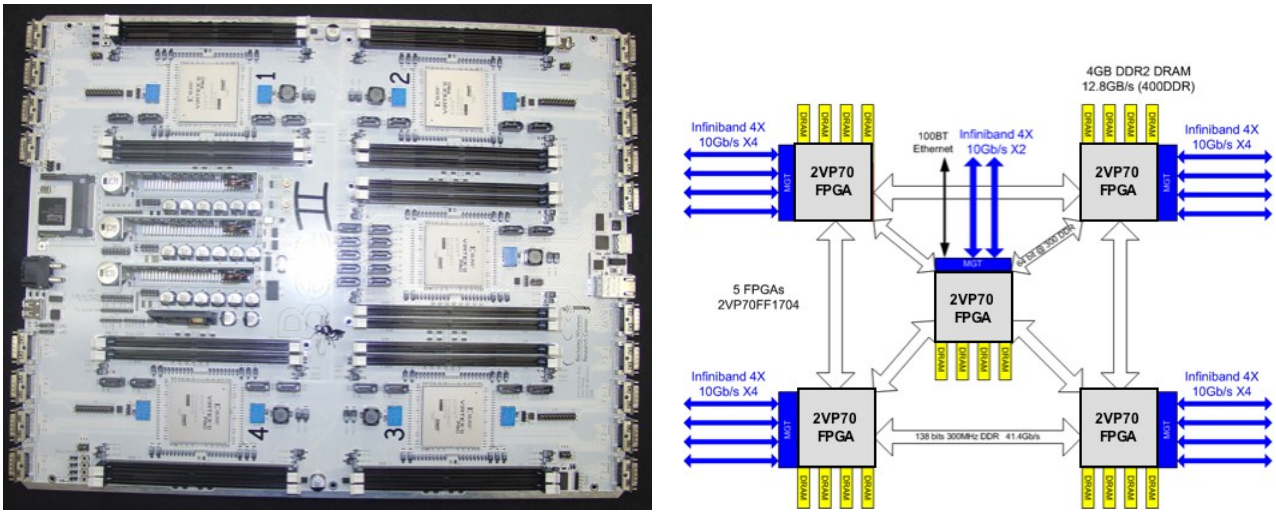

Fig. 5. BEE2 module photograph and architecture diagram.

available on the control FPGA, provides an out-of-band communication network for user interface, low speed system control, monitoring, and data archiving purposes. The compute module runs the Linux OS on the control FPGA with a full IP network stack.

In our preliminary work developing the first RAMP prototypes, we have made extensive use of the Xilinx XUP Virtex-II Pro Development System. As with the BEE2 board, the XUP uses Xilinx Virtex-II Pro FPGA technology; In this case, a single XC2VP30 instead of five XC2VP70s. It also includes an FPGA–SDRAM interface (DDR instead of DDR2) and includes a number of I/O interfaces, such as video, USB2, and Ethernet. In spite of its reduced capacity, the XUP has been a convenient development platform for key gateware blocks before moving them to the BEE2 system.

VII. SIDEBAR: Simulation and Emulation Technologies

Early computer architecture research relied upon convincing argument or simple analytical models to justify design decisions. Beginning in the early 1980's computers became fast enough that simple simulations of architectural ideas could be performed. By the 1990's, and onward to today, computer architecture research has come to rely extensively on software simulation. Many sophisticated software simulation frameworks exist, including SimpleScalar [4], SimOS [14], RSIM [13], Simics [17], ASIM [7] and M5 [2]. As our field's research focus shifts to multi-core, multi-threading systems, a new crop of multiprocessor full-system simulators—with accurate OS and I/O support—have more recently emerged (e.g., [16], [11], [3]). Software simulation has significantly changed the computer architecture research field because it is comparably easy to use, and it can be parallelized effectively by using separate program instances to simultaneously explore the design space of architectural choices.

Nevertheless, even for studying single core architectures, software simulation is slow to generate a single datapoint. Detailed simulations of out-of-order microprocessors typically execute in the kilo-instructions per second range. In the case of the multiprocessor simulation, the performance bottleneck is magnified since the simulators slow down commensurably as the number of cores studied continues to rise. A number of researchers have explored mechanisms to speedup simulation. The first of these techniques relied upon modifying the inputs used to benchmarks in order to reduce their total running time [10]. Later, researchers recognized that the repetitive nature of program execution could be exploited to subset the amount of time on which a detailed microarchitectural model is exercised. The first technique to exploit this was basic block vectors [15]. Later researchers proposed techniques that continuously sample program execution to find demonstrably accurate subsets [18].

As previously discussed, it is now widely accepted that the challenges facing our field will find solutions only by innovating both hardware *and* software. In order to engage software researchers, proposed new

architectures must be usable for real software development. The possibility of FPGA prototyping and simulation acceleration has garnered the interest of computer architects for as long as the technology existed. Unfortunately, until recently, this avenue has met only limited success due to the restrictive capacity of earlier generation FPGAs and the relative ease of simulating uniprocessor systems in software. An example of a large-scale FPGA prototyping effort is RPM [12]. The RPM system enabled flexible evaluation of the memory subsystem, but it was limited in scalability (8 processors) and did not execute OS code. With current FPGA capacity, RAMP and other like-minded efforts stand to provide a much needed, scalable research vehicle for full-system multiprocessor research.