RAMP: Research Accelerator for Multiple Processors

John Wawrzynek
David Patterson
University of California, Berkeley

Mark Oskin
University of Washington

Shih-Lien Lu
Intel

Christoforos Kozyrakis
Stanford University

James C. Hoe
Carnegie Mellon University

Derek Chiou
University of Texas at Austin

Krste Asanović
Massachusetts Institute of Technology

THE RAMP PROJECT'S GOAL IS TO ENABLE THE INTENSIVE, MULTIDISCIPLINARY INNOVATION THAT THE COMPUTING INDUSTRY WILL NEED TO TACKLE THE PROBLEMS OF PARALLEL PROCESSING. RAMP ITSELF IS AN OPEN-SOURCE, COMMUNITY-DEVELOPED, FPGA-BASED EMULATOR OF PARALLEL ARCHITECTURES. ITS DESIGN FRAMEWORK LETS A LARGE, COLLABORATIVE COMMUNITY DEVELOP AND CONTRIBUTE REUSABLE, COMPOSABLE DESIGN MODULES. THREE COMPLETE DESIGNS—FOR TRANSACTIONAL MEMORY, DISTRIBUTED SYSTEMS, AND DISTRIBUTED-SHARED MEMORY—DEMONSTRATE THE PLATFORM'S POTENTIAL.

In 2005, the computer hardware industry took a historic change of direction: The major microprocessor companies all 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 International Symposium on Computer Architecture (ISCA) in June 2005. We were struck that a multibillion-dollar industry would bet its 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 problem, our industry urgently needs innovative solutions, which in turn require extensive codevelopment of hardware and software. However, this type of innovation currently gets bogged down in the traditional development cycle:

- Prototyping a new architecture in hardware takes approximately four years and many millions of dollars, even at only research quality.
- Software engineers are ineffective until the new hardware actually shows up, because simulators are too slow to support serious software development activities. Software engineers tend to innovate only after hardware arrives.
- Feedback from software engineers on the current production hardware cannot help the immediate next generation because of 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—architectures, compilers, operating systems, applications, and others—to work together on perhaps the greatest challenge facing computing in the past

Published by the IEEE Computer Society

50 years. Because our industry desperately needs solutions, our goal is to develop a platform that would allow far more rapid evolution than traditional approaches.

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, and have 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 1,000-processor system in just 64 FPGAs. Such a system is cheaper and consumes less power than a custom multiprocessor, at about $100 and 1 W per processor.

Because our goal is to ramp up the rate of innovation in hardware and software multiprocessor research, we named this project RAMP (Research Accelerator for Multiple Processors). 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 plan to include research features that are impossible to include in real hardware systems owing to speed, cost, or practicality issues. For example, the FPGA design can incorporate additional hardware to monitor any event in the system. 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 differs from real hardware is reproducibility. Using the RAMP Description Language (RDL) framework, 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 will start with familiar architectures and operating systems, which will provide far 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 fast enough to run standard operating systems and large-scale applications that are orders of magnitude faster than software simulators. Finally, because of 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.

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 1980s culture of building experimental hardware and software systems, which today has been almost entirely lost because of the higher cost and difficulty of building hardware.

Table 1 compares alternatives for pursuing parallel-systems research in academia. The four options are a conventional shared-memory multiprocessor (SMP), a cluster, a simulator, a custom-built chip and system, and RAMP. The rows are the features of interest, with a grade for each alternative and quantification where appropriate. Cost rules out a large SMP for most academics. The costs of both purchase and ownership make a large cluster too expensive for most academics as well. Our only alternative thus far has been software simulation, and indeed that has been the vehicle for most architecture research in

Table 1. Relative comparison of four options for parallel research. From the architect's perspective, the most surprising aspect of this table is that not only is performance *not* the top concern, it is at the bottom of this list. The platform just needs to be fast enough to run the entire software stack.

Feature	SMP	Cluster	Simulator	Custom	RAMP
Scalability (1,000 CPUs)	C	A	A	A	A
Cost (1,000 CPUs)	F ($40M)	C ($2M-$3M)	A+ ($0M)	F ($20M)	A ($0.1M-$0.2M)
Cost of ownership	A	D	A	D	A
Power/space (kW, 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
Overall grade	C	C+	B	B−	A

the past decade. As mentioned, software developers rarely use software simulators, because they run too slowly, and results might not be credible. In particular, it's unclear how credible results will be to industry when they are based on simulations of 1,000 processors running small snippets of applications. The RAMP option is a compromise among these alternatives: It is so much cheaper than custom hardware that it will make highly scalable systems affordable to academics. It is as flexible as simulators, allowing rapid evolution of the state of the art in parallel computing. And it is so much faster than simulators that it could actually tempt software people to try out a new hardware idea.

This speed also lets architects explore a much larger space in their research and thus 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 computing cluster, this does not reduce the latency of obtaining a single key result that can move the research forward. Nor does it 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.

RAMP is obviously attractive to a broad set of hardware and software researchers in parallelism. Some representative research projects that we believe could benefit from using RAMP are

- testing the robustness of multiprocessor hardware and software under fault insertion;
- developing thread scheduling and data allocation and migration techniques for large-scale multiprocessors;
- developing and evaluating ISAs for large-scale multiprocessors;
- creating an environment to emulate a geographically distributed computer, with realistic delays, packet loss, and so on (Internet in a box);
- evaluating the impact of 128-bit and other floating-point representations on convergence of parallel programs;
- developing and testing hardware and software schemes for improved security;
- recording traces of complex programs running on a large-scale multiprocessor;
- evaluating the design of multiprocessor switches (serial point-to-point, distributed torus, fat trees);

- developing data-flow architectures for conventional programming languages;
- developing parallel file systems;
- testing dedicated enhancements to standard processors; and
- compiling software directly into FPGAs.

We believe that RAMP's upside potential is so compelling that the platform will create a "watering hole" effect in academic departments as people from many disciplines use RAMP in their research. As researchers from such diverse fields begin using RAMP, conversations between disciplines that rarely communicate may result, ultimately, in helping 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.

RAMP design framework

From the earliest stages of the RAMP project, it was clear that we needed a standardized design framework to enable a large community of users to cooperate and build a useful library of interoperable hardware models. This design framework has several 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, so that groups with different FPGA emulation hardware can share designs and RAMP modules for reuse after FPGA emulation hardware upgrades. In addition, the design framework should not dictate the hardware design language (HDL) that the developers choose. Our approach was to develop a decoupled machine model and design discipline. This discipline is enforced by the RDL and a compiler to automate the difficult task of providing cycle-accurate emulation of distributed communicating components.[1]

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—for example, a processor with an L1 cache, a DRAM controller, or a network router stage. All communication between units is via messages sent over unidirectional point-to-point interunit channels, where each channel is buffered to allow units to execute decoupled from one another.

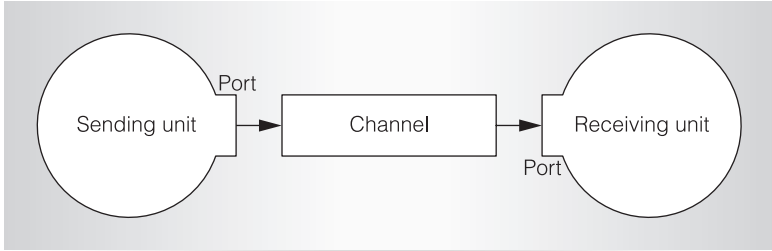

Figure 1. Basic RAMP communication model.

Partitioning of target models is far simpler than the classic circuit-partitioning problem associated with traditional FPGA-based circuit emulation. Although units will be large, we expect them to be relatively small compared to the FPGA capacity, so they will never be partitioned across multiple FPGAs. A target model is only partitioned at the channel interfaces, leaving units intact. Channels connecting units that map to separate FPGAs are implemented using FPGA-to-FPGA physical links. Currently, partitioning is driven by user annotations in RDL, but eventually we expect to build automatic partitioning tools.

Each unit faithfully models the behavior of each target clock cycle in the component. The target unit models can be developed either as register-transfer-level (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

Table 2. RAMP timeline.

Date	Milestone
6 June 2005	Hallway discussions lead to RAMP vision
13 June 2005	The name ''RAMP'' coined; BEE2[2] 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 Massachusetts Institute of Technology; RAMP Red running with eight processors on RAMP-1 boards
January 2007	RAMP Blue running with 256 processors on eight RAMP-1 boards
August 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

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 synchronized only through 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 until 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 lets units run at various 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 an 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 by the RDL compiler and runtime system. 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 of having channels run as fast as the underlying physical hardware will allow, thus supporting 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 rerunning the FPGA synthesis flow when varying parameters for performance studies.

The configuration tool will include support for interunit 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. These monitoring and debugging facilities will provide significant advantages over running applications on commercial hardware.

RAMP prototypes

Although most of the participants in the project are volunteers, we are on a fairly aggressive schedule. Table 2 shows the RAMP project timeline. We began RAMP development using preexisting FPGA boards—see the ''RAMP hardware'' sidebar. To seed the collaborative effort, we are developing three prototype systems: RAMP Red, RAMP Blue, and RAMP White. Each of our initial

RAMP hardware

Rather than begin the RAMP project by designing yet another FPGA board, for the RAMP-1 system we adopted the Berkeley Emulation Engine.[1] 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 next generation RAMP hardware platform, currently in design, will be based on a new board design employing the recently announced Virtex-5 FPGA architecture.

Figure A shows the BEE2 compute module. Each compute module consists of five Xilinx Virtex-2 Pro-70 FPGA chips, each directly connected to four DDR2 240-pin DRAM dual in-line memory modules (DIMMs), with a maximum capacity of 4 Gbytes per FPGA. The four DIMMs are organized into four independent DRAM channels, each running at 200 MHz (400 DDR) with a 72-bit data interface. Therefore, peak aggregate memory bandwidth is 12.8 Gbytes per second for each 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 falls into two classes: on-board LVCMOS connections and off-board multigigabit transceiver (MGT) connections. The local mesh connects the four compute FPGAs on a 2×2 2D grid. Each link between the adjacent FPGAs on the grid provides over 40 Gbps of 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 using 8B/10B encoding. Every four 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, available on the control FPGA, provides an out-of-band communication network for user interface, low-speed system control, monitoring, and data archival. 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 University Program (XUP) Virtex-II Pro Development System (http://www.xilinx.com/univ/xupv2p.html). As with the BEE2 board, the XUP board uses Xilinx Virtex-II Pro FPGA technology—in this case, a single XC2VP30 instead of five XC2VP70s. The XUP board also includes an FPGA-SDRAM interface (DDR instead of DDR2) and several I/O interfaces, such as video, USB2, and Ethernet. Despite its reduced capacity, the XUP board has been a convenient development platform for key gateware blocks before moving them to the BEE2 system.

References

1. C. Chang, J. Wawrzynek, and R.W. Brodersen, ''BEE2: A High-End Reconfigurable Computing System,'' *IEEE Design & Test*, vol. 22, no. 2, Mar.-Apr. 2005, pp. 114-125.

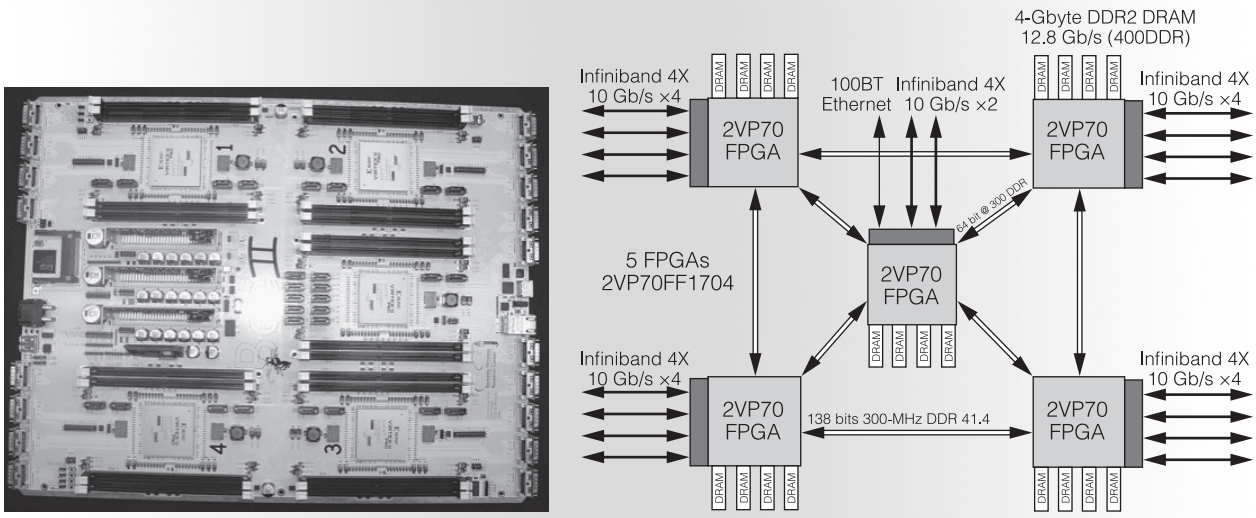

Figure A. BEE2 module photograph (a) and architecture diagram (b).

prototypes contains a complete gateware and 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 they can start from the ground up, using the basic components to build a new system. We expect users to release back to the community any enhancements and new gateware and 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.

RAMP Red

RAMP Red is the first multiprocessor system with hardware support for transactional memory. Transactional memory transfers the responsibility for concurrency control from the programmer to the system.[3] 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 transactional memory 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 Transactional Coherence and Consistency (TCC) architecture for transactional memory.[4] The design uses nine PowerPC 405 hard cores (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 that are read and written by a transaction during its execution and detects atomicity violations with other ongoing 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 perspective, RAMP Red is a fully featured Linux workstation. The operating system actually runs on just one of the cores, while the remaining eight cores execute applications. A light-weight kernel in each application core forwards exceptions and system calls to the operating system core. The programming model is multithreaded C or Java with locks replaced by transactional constructs. RAMP Red includes an extensive hardware and 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. Wee et al. provide more details of the RAMP Red design.[5] 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.[6]

RAMP Blue

RAMP Blue is a family of emulated message-passing machines that can 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 model a networked server cluster.

The first RAMP Blue prototype was developed at the University of California, Berkeley; Figure 2 shows its hardware platform. It comprises a collection of BEE2 boards housed in 2 U chassis and assembled in a standard 19-inch rack. Physical connection among the eight boards is through 10-Gbps Infiniband cables (light-colored cables in Figure 2). The BEE2 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 take place through a 100-Mbps Ethernet switch with connection to each board's control FPGA (dark wires at the top of Figure 2). 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 versions of the NASA Advanced Supercomputing (NAS) Parallel Benchmarks.

The four user FPGAs per BEE2 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, emulates a 256-way cluster system. In the future, the number of processor cores can be scaled up through several means. We will add more BEE2 boards—the simple all-to-all wiring configuration will accommodate up to 17 boards. We will also add more cores per FPGA—the current configuration of eight processor cores per FPGA only consumes 40 percent of the FPGA's logic resources. RAMP Blue implements all necessary multiprocessor components within the user FPGAs. In addition to the soft processor cores, each FPGA holds a packet network switch (one per core) for connection to cores on the same and other FPGAs, shared memory controllers, shared double-precision floating-point units, 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 Mbytes per processor). The external memory interface of the MicroBlaze L1 cache connects to external DDR2 (double-data-rate 2) DRAM through a memory arbiter, as each DRAM channel is shared among a set of MicroBlaze cores. Because each BEE2 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 that 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 in the memory controller.

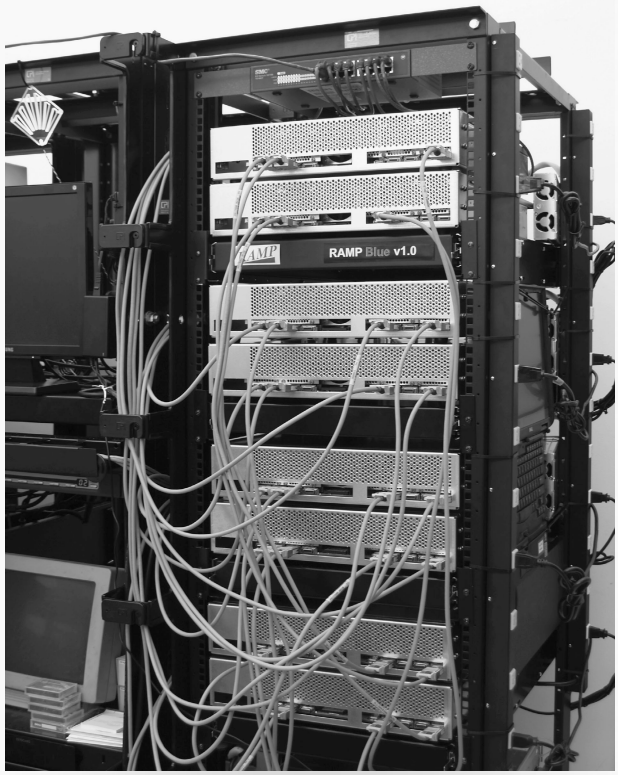

Figure 2. Photograph of RAMP Blue prototype.

A 256-core (eight per FPGA) version of the RAMP Blue prototype has been fully operational, running the NAS Parallel Benchmark suite, since December 2006. This initial prototype was not implemented using RDL, but a newer version based on RDL has been operational since February 2007. We are currently measuring and tuning the prototype's performance.

RAMP White

RAMP White is a distributed-shared-memory machine that will demonstrate RAMP's open-component nature by integrating modules from RAMP Red, RAMP Blue, and contributions from other RAMP participants. The initial version is being designed and integrated at the University of

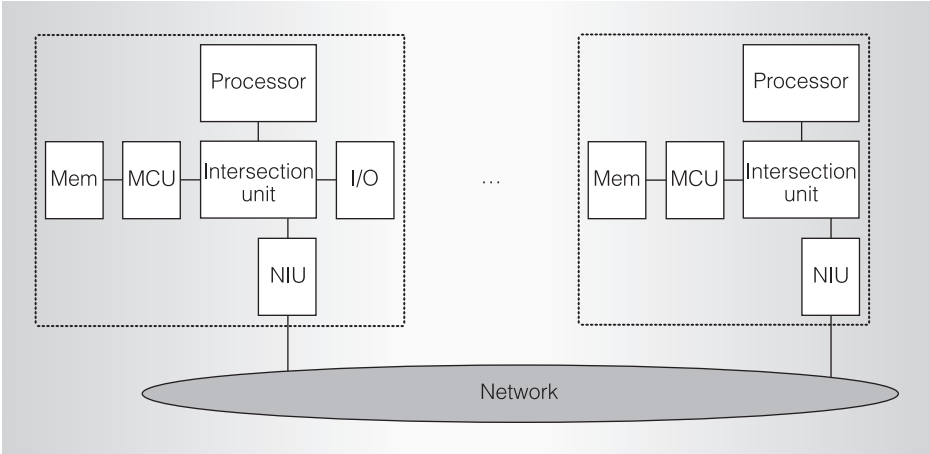

Figure 3. High-level view of RAMP White.

Texas at Austin. The RAMP White effort began in the summer of 2006, somewhat after Red and Blue, and will be implemented in the following phases.

1. *Global distributed-shared memory without caches.* All requests to remote global memory will be serviced directly from remote memory. Communication will take place 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 through a simple ring network (Figure 3). For expediency, the initial RAMP White will use embedded PowerPC processors. Each processor unit will contain one processor connected to an intersection unit 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 intersection unit switches requests and replies between the processor, local memory, I/O units, and the network. The initial intersection unit 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. Because the initial version of RAMP White does not cache global locations, incoming remote requests need not 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 intersection unit passes all incoming remote requests to the coherent cache for snooping before allowing the remote request to proceed to the next stage.

RAMP represents the research community's return to building hardware-software systems. RAMP is designed to embody the right trade-offs of cost, performance, density, and visibility for system

Simulation and emulation technologies

Early computer architecture research relied on convincing argument or simple analytical models to justify design decisions. Beginning in the early 1980s, computers became fast enough that simple simulations of architectural ideas could be performed. By the 1990s, and onward, computer architecture research relied extensively on software simulation. Many sophisticated software simulation frameworks exist, including SimpleScalar,[1] SimOS,[2] RSIM,[3] Simics (http://www.virtutech.com), ASIM,[4] and M5.[5] As our field's research focus shifts to multicore, multithreading systems, a new crop of multiprocessor full-system simulators with accurate operating-system and I/O support (for example, see http://www.ece.cmu.edu/simflex/flexus.html) have more recently emerged.[6,7] 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 data point. Detailed simulations of out-of-order microprocessors typically execute in thousands of instructions per second. Multiprocessor simulation tightens the performance bottleneck because the simulators slow down commensurably as the number of studied cores continues to rise. Several researchers have explored mechanisms to speed up simulation. The first of these techniques relied on modifying the inputs to benchmarks used, to reduce their total running time.[8] Later, researchers recognized that the repetitive nature of program execution could be exploited to reduce the amount of time on which a detailed microarchitectural model is exercised. The first technique to exploit this was basic block vectors.[9] Later researchers proposed techniques that continuously sample program execution to find demonstrably accurate subsets.[10]

But the challenges facing our field will find solutions only by innovating both hardware and software. 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 has existed. Unfortunately, until recently, this avenue has met only limited success, because of 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 the Rapid Prototype Engine for Multiprocessors (RPM).[11] The RPM system enabled flexible evaluation of the memory subsystem, but it was limited in scalability (eight processors) and did not execute operating system code. With current FPGA capacity, RAMP and similar efforts stand to provide a much needed, scalable research vehicle for full-system multiprocessor research.

References

1. D. Burger and T.M. Austin, *The SimpleScalar Tool Set, Version 2.0*, tech. report 1342, Computer Sciences Dept., Univ. of Wisconsin, Madison, 1997.
2. M. Rosenblum et al., ''Complete Computer System Simulation: The SimOS Approach,'' *IEEE Parallel and Distributed Technology*, vol. 3, no. 4, Winter 1995, pp. 34-43.
3. V.S. Pai, P. Ranganathan, and S.V. Adve, *RSIM Reference Manual, Version 1.0*, tech. report 9705, Electrical and Computer Engineering Dept., Rice Univ., July 1997.
4. J. Emer et al., ''ASIM: A Performance Model Framework,'' *Computer*, vol. 22, no. 2, Feb. 2002, pp. 68-76.
5. N.L. Binkert et al., ''The M5 Simulator: Modeling Networked Systems,'' *IEEE Micro*, vol. 26, no. 4, Jul.-Aug. 2006, pp. 52-60.
6. M.M.K. Martin et al., ''Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset,'' *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, Nov. 2005, pp. 92-99.
7. N.L. Binkert, E.G. Hallnor, and S.K. Reinhardt, ''Network-Oriented Full-System Simulation Using M5,'' *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads* (CAECW 03), Feb. 2003; http://www.eecs.umich.edu/~stever/pubs/caecw03.pdf.
8. A. KleinOsowski and D. Lilja, *MinneSpec: A New Spec Benchmark Workload for Simulation-Based Computer Architecture Research*, tech. report 02-08, ARCTiC Labs, Univ. of Minnesota, 2002; http://www.arctic.umn.edu/papers/minnespec-cal-v2.pdf.
9. T. Sherwood, E. Perelman, and B. Calder, ''Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques* (PACT 01), IEEE CS Press, 2001, pp. 3-14.
10. R. Wunderlich et al., ''Smarts: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling,'' *Proc. 30th Ann. Int'l Symp. Computer Architecture* (ISCA 03), IEEE CS Press, 2003, pp. 84-95.
11. K. Oner et al., ''The Design of RPM: An FPGA-Based Multiprocessor Emulator,'' *Proc. 3rd ACM Int'l Symp. Field-Programmable Gate Arrays* (FPGA 95), ACM Press, 1995, pp. 60-66.

research. Moreover, since the system is not frozen, we can use it to both rapidly evolve and spread successful ideas across the community. Research in hardware architecture, operating systems, compilers, applications, and programming models will all benefit. We are planning a full public release of the RAMP infrastructure in 2007. MICRO

Acknowledgments

This work was funded in part by the National Science Foundation, grant CNS-0551739. Special thanks to Xilinx for its continuing financial support and donation of FPGAs and development tools. We appreciate the financial support provided by the Gigascale Systems Research Center. Thanks to IBM for its 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. We especially thank Arvind (Massachusetts Institute of Technology) and Jan Rabaey (University of California, Berkeley) for their advice. The work presented in this article is the effort of the RAMP students and staff: Hari Angepat, Dan Burke, Jared Casper, Chen Chang, Pierre-Yves Droz, Greg Gibeling, Alex Krasnov, Ken Lutz, Martha Mercaldi, Nju Njoroge, Andrew Putnam, Andrew Schutlz, and Sewook Wee.

References

1. G. Gibeling, A. Schultz, and K. Asanovic, ''RAMP Architecture and Description Language,'' 2nd Workshop Architecture Research Using FPGA Platforms, 2006; http://cag.csail.mit.edu/~krste/papers/rampgateware-warfp2006.pdf.
2. C. Chang, J. Wawrzynek, and R.W. Brodersen, ''BEE2: A High-End Reconfigurable Computing System,'' *IEEE Design & Test*, vol. 22, no. 2, Mar.-Apr. 2005, pp. 114-125.
3. A.-R. Adl-Tabatabai, C. Kozyrakis, and B. Saha, ''Unlocking Concurrency: Multicore Programming with Transactional Memory,'' *ACM Queue*, vol. 4, no. 10, Dec. 2006; http://acmqueue.com/modules.php?name=Content&pa=showpage&pid=444.
4. L. Hammond et al., ''Programming with Transactional Coherence and Consistency (TCC),'' *Proc. 11th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 04), ACM Press, 2004, pp. 1-13.
5. S. Wee et al., ''A Practical FPGA-Based Framework for Novel CMP Research,'' *Proc. 15th ACM/SIGDA Int'l Symp. Field-Programmable Gate Arrays* (FPGA 07), ACM Press, 2007, pp. 116-125.
6. B.D. Carlstrom et al., ''The Software Stack for Transactional Memory: Challenges and Opportunities,'' First Workshop on Software Tools for Multicore Systems (STMCS 06), 2006; http://ogun.stanford.edu/~kunle/publications/tcc_stmcs2006.pdf.

John Wawrzynek is a professor of electrical engineering and computer sciences at the University of California, Berkeley. He currently teaches courses in computer architecture, VLSI system design, and reconfigurable computing. He is codirector of the Berkeley Wireless Research Center and principal investigator of the Research Accelerator for Multiple Processors (RAMP) project. He holds a PhD and MS in computer science from the California Institute of Technology and a MS in electrical engineering from the University of Illinois, Urbana-Champaign. He is a member of the IEEE and the ACM.

David Patterson is the Pardee Professor and Director of RAD Lab at the University of California, Berkeley. His research interests are in design and automatic management of datacenters and in hardware-software architectures of highly parallel microprocessors. He is a fellow of IEEE; recipient of the IEEE von Neumann and IEEE Mulligan Education medals; fellow and past president of the ACM; and a member of the American Academy of Arts and Sciences, the National Academy of Engineering, the National Academy of Sciences, and the Silicon Valley Engineering Hall of Fame.

Mark Oskin is an assistant professor at the University of Washington. His research interests include computer systems architecture, performance modeling, and architectures for emerging technologies. With a large team of graduate students, he is the coinventor of WaveScalar, the first dataflow machine capable of executing applications written in imperative languages, and brick and mortar, a low-cost technique for manufacturing SoC devices. He received his PhD in computer

science from the University of California, Davis.

Shih-Lien Lu has a BS in electrical engineering and computer sciences from the University of California, Berkeley, and an MS and a PhD, both in computer science and engineering, from the University of California, Los Angeles. He served on the faculty of the Electrical and Computer Sciences Department at Oregon State University from 1991 to 2001. In 1999, he took a two-year leave from OSU and joined Intel. He is currently a principal research scientist in the microarchitecture lab of Intel's Microprocessor Technology Lab in Oregon. His research interests include computer microarchitecture, circuits, and FPGA systems design.

Christoforos Kozyrakis is an assistant professor of electrical engineering and computer science at Stanford University. His research focuses on architectural support for parallel computing, system security, and energy management. He is currently working on transactional memory techniques that can greatly simplify parallel programming for the average developer. He has a PhD in computer science from the University of California, Berkeley.

James C. Hoe is an associate professor of electrical and computer engineering at Carnegie Mellon University. His research interests include computer architecture and high-level hardware description and synthesis. He has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a member of the IEEE and the ACM.

Derek Chiou is an assistant professor at the University of Texas at Austin. His research interests include computer system simulation, computer architecture, parallel computer architecture, and Internet router architecture. He received his PhD, SM, and SB degrees in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a senior member of the IEEE and a member of the ACM.

Krste Asanović is an associate professor in the Department of Electrical Engineering and Computer Science at MIT and a member of the MIT Computer Science and Artificial Intelligence Laboratory. His research interests include computer architecture and VLSI design. Asanović has a BA in electrical and information sciences from the University of Cambridge and a PhD in computer science from the University of California, Berkeley. He is a member of the IEEE and the ACM.

Direct questions and comments about this article to John Wawrzynek, 631 Soda Hall, Computer Science Division, University of California, Berkeley, CA 94720-1776; johnw@eecs.berkeley.edu.