Project Summary for NSF Solicitation 04-588

CRI: RAMP: Research Accelerator for Multiple Processors - A Community Vision for a Shared Experimental Parallel HW/SW Platform

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

Desktop processor architectures have crossed a critical threshold. Manufactures have given up attempting to extract ever more performance from a single core and instead have turned to multi-core designs. While straightforward approaches to the architecture of multi-core processors are sufficient for small designs (2–4 cores), little is really known how to build, program, or manage systems of 64 to 1024 processors. Unfortunately, the computer architecture community lacks the basic infrastructure tools required to carry out this research. While simulation has been adequate for single-processor research, significant use of simplified modeling and statistical sampling is required to work in the 2–16 processing core space. Invention is required for architecture research at the level of 64–1024 cores.

Fortunately, Moore's law has not only enabled these dense multi-core chips, it has also enabled extremely dense FPGAs. Today, for a few hundred dollars, undergraduates can work with an FPGA prototype board with almost as many gates as a Pentium. Given the right support, the research community can capitalize on this opportunity too. Today, one to two dozen cores can be programmed into a single FPGA. With multiple FPGAs on a board and multiple boards in a system, large complex architectures can be explored. To make this happen, however, requires a significant amount of infrastructure in hardware, software, and what we call "gateware", the register-transfer level models that fill the FGPAs. While it is possible for each research group to create these boards, design the gateware, and create this infrastructure in isolation, significant benefits can be had by pooling our collective resources.

Such a system would not just invigorate multiprocessors research in the architecture community. Since processors cores can run at 100 to 200 MHz, a large scale multiprocessor would be fast enough to run operating systems and large programs at speeds sufficient to support software research. Moreover, there is a new generation of FPGAs every 18 months that is roughly twice as fast and with capacity for twice as many cores, so future multiboard FPGA systems are even more attractive. Hence, we believe such a system would accelerate research across all the fields that touch multiple processors: operating systems, compilers, debuggers, programming languages, scientific libraries, and so on. Thus the acronymn RAMP, for Research Accelerator for Multiple Processors.

This proposal seeks funding for just such a community endeavor. By leveraging the work that each of us was going to do anyway in isolation, we can create a shared infrastructure for architecture research. Furthermore, by pooling our resources in hardware design we can reduce the risk each of us undertakes by designing hardware prototypes ourselves. Finally, by creating shared and *supported* baseline platforms for multi-core architectures we can jump start the required architecture and critical software research that the field needs.

Intellectual Merit

The intellectual merit of this proposal is embodied in the following contribution. First, we intend to create a set of RTL and software design standards that facilitate easy adaptability and integration of hardware and software components in the multi-core architecture. Second, we intend to use these design standards to create a baseline architecture of 1024+ nodes. Third, we will investigate architectures for fast emulation of large scale multiprocessors. For example, what type of memory controller and caches will speedup emulation of 1024 processors. Fourth, we will create systems to observe the MPP behavior without disturbing the computation. This design will be created, distributed for free on the Internet, and supported through full-time staff.

Broader Impact

The broader impact goal of this proposal is nothing less than the transformation of the parallel computing community in computer science from a simulation-driven to a prototype-driven discipline. The impacts of this will be rapid iteration across interfaces of the many fields of multiple processors, thereby moving much more quickly to a parallel foundation for large-scale computer systems research in the 21st century.

CRI: RAMP: Research Accelerator for Multiple Processors - A Community Vision for a Shared Experimental Parallel HW/SW Platform

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

1 Overview

We propose to build and support a large-scale, FPGA-based emulation platform to accelerate research in multiprocessors. Advances in FPGA capacity and speed now enable a 1024-processor system to be emulated with just a few dozen FPGAs at very low cost per processor. By creating a research community around a common hardware and software infrastructure, we will pool our resources to build a far more productive environment than we could have achieved individually. Hence the name of the system, Research Accelerator for Multiple Processors (RAMP), as it should ramp up the rate of research in hardware and software for multiple processors.

We expect RAMP to provide multiple orders of magnitude speedup over software-based simulation. This speedup will provide a qualitative leap in the quality and range of future computer architecture research. By running a full RTL model at high-speed, researchers will have much higher confidence in the accuracy of their results and the feasibility of their ideas.

In fact, the resulting prototype systems will have sufficient speed to interest a much wider research community, including compiler researchers, operating system developers, and distributed system designers. Previous experience has shown that software researchers are only inspired to work with new computer architectures when a hardware prototype is available. Furthermore, RAMP allows rapid changes in the hardware based on feedback from software developers; unlike conventional chip fabrication, we can tape out every day.

Because RAMP is a research platform, we can offer capabilities not found in commercial multiprocessors. For example, RAMP could have reproducible behavior, where every processor's memory references and interrupts would happen at exactly the same clock cycle on every run. RAMP would be valuable for many forms of software experiments, as well as aiding in software debugging.

Rather than having many institutions build their own RAMP-like boards and accompanying software with NSF funding, we propose that NSF fund the development of a common platform aided by industrial donations. The goal would be that any site could obtain a copy of the hardware at modest cost, and download a fully-functioning large-scale multiprocessor including hardware design and software environment. This will lower the barrier to entry for access to a multiprocessor research capability, and will dramatically expand the set of individuals and departments who can participate in this new wave of architecture and software research. We believe we only need NSF to fund two full-time engineering staff to realize this bold vision, as we have gathered together a large community of academic and industrial researchers who are willing to share in the development of RAMP, and we have also successfully solicited industrial donations.

Both parallel software and multiprocessor architecture are critical research areas, and RAMP will provide a platform to support much richer interaction between these communities. We believe RAMP will offer an unprecedented combination of speed, accuracy, repeatability, and adaptability in a large-scale multiprocessor platform with reduced cost and risk to both NSF and individual researchers.

1.1 Background

Most computer architects believe the future of the microprocessor is a large number, perhaps hundreds or even thousands, of processors on a chip. Given such widespread agreement, it is surprising how much research remains to be done in computer architecture, networks, operating systems, file systems, compilers, programming languages, applications, and so on to realize this vision.

The members of a panel on the topic at ISCA in June 2005 articulated how far we have to go. As Wen Mei Hwu of Illinois pointed out, one of the challenges of making progress is that software people generally become engaged only after the hardware is available. His group used VLIW/EPIC simulators for years, but made far more progress in the first six months after the Itanium hardware arrived. From those results, they were able to quickly propose architectural improvements that were incorporated into next generation Itaniums, which then took years to realize. The "edit-compile-run-debug" loop that takes minutes in software takes years when custom hardware is involved and is the real bottleneck in advancing that hardware and the software around it.

The good news for computer designers is that FPGAs have made it plausible to create sophisticated prototypes of new architectures that can be fully functional, running a full software stack. Rapid iteration is the foundation of FPGA; you can "tape out" every day. Moreover, FPGAs are large and still growing at Moore's Law rates. Today's FPGAs can hold one or two dozen "soft-core" RISC processors and we expect this to double roughly every 18 months. For example, Altera reported that the companies FPGA could hold 100 Nios-II 32-bit RISC soft cores [1]. Although a 64-bit commercial architecture, like PowerPC or SPARC, would take much more space than Nios, we still believe we can place at least one or two dozen on an FPGA.

Hence, we could plausibly build a 1024-processor system from just 40 to 80 FPGAs. Although these processors would be factors of 10 to 20 times slower than custom hardware, 100 to 200 MHz processors are still fast enough to run real applications and operating systems. In addition to their flexibility, FPGA-based systems enable a level of detailed probing and monitoring that has not been available since the days of breadboarding with SSI logic. Depending on the number of boards desired, the relatively low cost—about as expensive as an SMP server—makes such systems affordable to many who would otherwise be excluded from this important research area.

Because of this potential, many groups have independently been investigating, or actually building, FPGA-based hardware systems to investigate their hardware and software ideas. Rather than simply duplicating the cost and effort of each project in coming up with their own homebrew FPGA emulation system, a group of us has been investigating whether we could create a common infrastructure that the community could share.

We are inspired by SimpleScalar, which was a recent example of a community resource whose development was sponsored by NSF and which considerably accelerated research and development in the architecture community.

1.2 Vision

The vision is a scalable, multiboard FPGA-based system that would allow construction of up to a 1024-CPU multiprocessor. This size is an interesting target because many problems that are invisible at 32 processors and awkward at 128 become glaring at 1024. This scale challenge is true across the architecture, network, operating system, and applications disciplines. This shared artifact would consist of hardware and a collection of "gateware" (RTL) and software that members of the community would help create. We would seed the cooperative using the IP from the "open source gateware" found at `www.opencores.org` and open source software such as Linux, gcc, ORC, and Jikes. By working together, we could create something more interesting than any of us would have the energy to create on our own.

For example, some of us would want a system that faithfully emulated each clock cycle of a prototype machine. Clock-cycle accurate performance measurements at high-speed would help software development and benchmark sensitivity studies, as we vary parameters such as clock rate, local memory latency and bandwidth, switch latency and bandwidth, and so on. Creating a system that could have reproducible behavior, where every interrupt happened at exactly the same clock cycle on every run, would be useful for debugging hardware and software. It would be wonderful to have this done well once and then shared across the architecture and related communities.

We imagine that this infrastructure could be valuable to many projects, but here are a few representative examples:

- Testing the robustness of multiprocessor HW/SW under fault insertion workloads

- Developing thread scheduling and data allocation/migration techniques for large scale multiprocessors
- Developing and evaluating instruction set architectures for large scale multiprocessors
- Creating an environment to emulate a geographically distributed computer, with realistic delays, packet loss, and so on
- Evaluating the impact of 128-bit floating point on convergence of parallel programs
- Developing and field testing of 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 (a la Tensilica)
- Compiling DSP software directly into FPGAs

The goal is that our initial distribution of gateware and software would contain a complete configuration of a scalable multiprocessor populated with standard processor cores, switches, operating systems, and compilers. A user should be able to load binaries for the default cores and run them without modification. Hence, the default cores would implement a commercially available ISA that already has an open source implementation available, such as IBM Power or Sun SPARC. Our goal would be that once the base system is assembled and software installed, users could 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. Users would release back to the community any enhancements and new hardware/software modules. A similar model has lead to the proliferation of the SimpleScalar framework which now covers a range of instruction sets and processor designs.

RAMP has the potential of offering scalable multiprocessors that are attractive to both architecture and software researchers. It would provide architects with an economical, easily modified, large-scale platform with tremendous tracing facilities, and software researchers with an economical, large-scale multiprocessor with extensive measurement and debugging support. It would provide software developers with a fast and malleable system to use with real applications, large datasets, and full-scale operating systems.

Our goal is to create an infrastructure to attract experts in architecture, operating systems, compilers, programming languages, and applications at many locations, and creating a community that could share tools and techniques since they share a common platform. This shared infrastructure could lead to greater interaction and research productivity across communities, as well as stretch NSF research funding. We could imagine trying to duplicate interesting results from other projects locally. Imagine the impact on research of FTPing a novel computer overnight and then test driving it.

1.3 Anticipated Utility and Impact

RAMP has the potential to accelerate research in parallel processing, but that potential will only be realized if it becomes popular. We offer three pieces of evidence for the potential utility and impact of RAMP.

- The first evidence is the speed with which the RAMP vision came together. RAMP started as a series of hallway discussions at the three-day computer architecture conference on June 6, and the more we talked over the ideas, the more excited we became. By the Friday after the Wednesday close of ISCA, we had recruited the 10 people on this proposal, and everyone else we asked agreed to help make RAMP happen. All of us have a lot on our plate already, but the potential benefit of RAMP was so important that we all made time to help make it happen.

Table 1: List of Industry Supporters

Name	Affiliation	Areas of Expertise
Gordon Bell	Microsoft	computer architecture
Ivo Bolsens	Xilinx, CTO	DSP, EDA
Norm Jouppi	HP Labs	computer architecture
Shih-Lien Lu	Intel	computer architecture
Craig Mundie	Microsoft CTO	software
Greg Papadopoulos	SUN, CTO	parallel processing, computer architecture
Justin Rattner	Intel, CTO	parallel processors, computer architecture
Ivan Sutherland	Sun	graphics, computer design
Chuck Thacker	Microsoft	hardware, computer design
Kees Vissers	Xilinx	embedded Computing

Table 2: List of Academic Supporters

Name	Affiliation	Areas of Expertise
Doug Burger	UT Austin	computer architecture, system implementation, simulation
Bill Dally	Stanford	computer architecture
Carl Ebeling	U Washington	computer architecture
Susan Eggers	U Washington	parallel computers, compilers, architecture
Steve Keckler	UT Austin	computer architecture
Bill Kramer	NERSC/LBL	Head, high-performance Computing
Greg Morrisett	Harvard Univ	programming languages, compilers, security
Scott Shenker	UC Berkeley	networking
Ion Stoica	UC Berkeley	networking
Kathy Yelick	UC Berkeley	parallel languages, compilers, algorithms

- The second piece of evidence is the number of people (Tables 1 and 2) and the quality of the people who sent supporting letters, and how quickly they agreed. Everyone we asked agreed to write a letter, and most were just people we casually talked to about RAMP at the various meetings we attended over the summer in the two months between the ISCA conference and the NSF proposal deadline. Although support letters are always positive, we urge the reviewers to read the letters as we find their words very encouraging, especially when combined with the reputation of those writing the letters. Note the list includes several luminaries from industry who will not benefit directly from RAMP, in addition to direct academic participants. We believe such independent endorsements help validate the popularity of the RAMP vision.

- The third piece of evidence is that we have at least one FPGA partner willing to 1:1 match NSF funding with up to $800,000 in parts and cash to make RAMP successful. FPGA companies are frequently asked to support academic projects, but cannot afford to fund them all. This level of funding would be one of the largest for a single academic project from an FPGA company. Once again, such valuable support helps convince us that RAMP can be successful.

1.4 Plan

By standing on each other's shoulders rather than on our collective toes, the community might ramp up research in multiprocessor systems, which is sorely needed.

Based on discussions with many people, we propose the following plan:

- **Phase 1:** Start developing the "community gateware" using the Berkeley Emulation Engine 2 (BEE2), developed at the Berkeley Wireless Research Center. This second-generation multiboard FPGA system is close enough to our idea of what RAMP should look like that we think it would be better to start now with BEE2 rather than spend a year figuring out a better design for our starting point. BEE2 uses Xilinx Virtex 2 FPGAs, and has been operational since January 2005. We would build one 8-board system and give 12 single-board systems to the initial set of gateware developers. We call this version of the HW/SW system RAMP 1.

- **Phase 2:** Based on the experience gained with RAMP 1 and working with our FPGA partner to use the latest generation of FPGAs, we would build RAMP 2. We would construct two 10-board systems, and then distribute 30 boards in various configurations to help grow the RAMP community of researchers.

Traditionally, each FPGA generation roughly doubles the number of processors and doubles the speed of the logic. RAMP 2 development would then be timed to use an FPGA two generations after Virtex 2, so it should have four times the number of processors running at roughly four times the clock rate. Such an FPGA should be on the market in 2006.

1.5 Industry Support

Our plan is to work with the FPGA manufacturer, Xilinx, to create boards that the research community could use to accelerate research in multiprocessor systems. The boards currently available from FPGA companies are oriented towards developers of single FPGA systems, rather than for researchers who want to construct large scalable infrastructures.

Based on a visit to the CTO of Xilinx, we propose the following support plan:

- **RAMP 1:** The FPGA partner would donate the FPGAs and pay for the construction of 20 BEE2 boards.
- **RAMP 2:** The FPGA partner would donate the FPGAs and pay for the construction of 50 new boards, and help find a company that could sell the boards, as FPGA companies are not generally in the business of making and selling such complex boards. The advantage of such a plan is that it will be easy for anyone interested to purchase copies of the hardware. In our view, the alternative approach where a university makes the designs available "at cost" is unlikely to result in a less expensive solution. Hence, the proposed budget does not include the cost to design and fabricate boards.

The attached letter from Ivo Bolston and Kees Vesser confirms Xilinx's participation in RAMP, provided NSF funds this proposal.

1.6 Request for NSF Support

We are asking for two full-time engineers with skills in software, RTL, and hardware development to make sure we have a fully coherent and documented design, and to build and debug the scalable prototype and make sure the design works. Details of the responsibilities of the engineers is provided in Section 3.2. If our RAMP vision succeeds, the community will have such momentum by the end of this proposal that it should no longer need to be sustained by centralized NSF financial support for infrustructure development. We expect that, by the time that RAMP 3 is needed either the FPGA partner or the board vendor will design and sell the board for a small profit, since it will be a commercially viable system.

1.7 Proposal Outline

We present the details of this proposal in the following sections. Section 2 presents the key components of the proposed RAMP infrastructure. Section 3 outlines our plan of attack for executing the proposed effort. Section 4 highlights several examples of how RAMP can be used to extend the quality and variety of computer architecture research and expands on the broader impacts of this work. Section 5 describes how the RAMP effort relates to the work of others. Section 6 briefly presents the results of recent prior NSF support for the PI and co-PI.

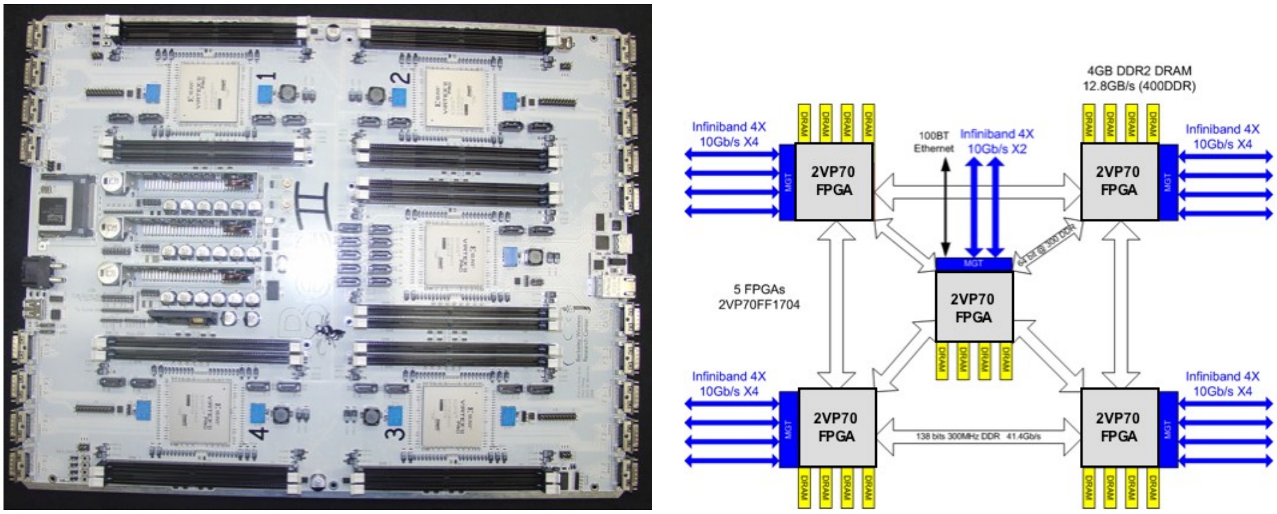

Figure 1: BEE2 Module Photograph and Architecture Diagram.

2 Infrastructure

This section presents the three key components of the RAMP infrastructure: FPGA prototyping substrate, the gateware modular design methodology, and the supported system models that will be created and distributed. We first present BEE, an existing design that will serve as the FPGA emulation platform in RAMP1. We next discuss our gateware design philosophy to ensure the flexibility, reusability and integrateability of design modules contributed by the community. Next, we describe the system models we intend to create and distribute; these models naturally progress from the simple uniprocessors to complex full-system MPP architectures. This section concludes with a survey of closely related prior efforts in FPGA prototyping.

2.1 Berkeley Emulation Engines

The Berkeley Wireless Research Center (BWRC) at UC Berkeley has developed two successful generations of FPGA-based emulation platforms. The first platform, the Berkeley Emulation Engine (BEE), evolved from the need for a development environment that enabled rapid prototyping of DSP systems. The complexity of wireless baseband designs lead to increased simulation times, prompting the need for a custom platform that could emulate a design in real-time.

The success of our first FPGA platform, BEE [5], inspired our development of a more powerful and scalable FPGA-based emulation platform. In recent years, the computational capabilities of commercial reconfigurable devices from Xilinx and Altera, have increased tremendously both in terms of gate count as well as circuit speed. By embedding dedicated arithmetic units (i.e., multipliers and accumulators) and general purpose processor cores (i.e., PowerPC 405) directly in the reconfigurable fabric, high-end FPGAs have evolved into a system-on-chip (SoC) solution to a variety of applications that require high throughput integer or floating point computations.

We have developed the next generation emulation platform, BEE2. The BEE2 Compute Module is shown in Figure 1. 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 4GB per FPGA. The four DIMMs are organized into four independent DRAM channels, each running at 200MHz (400DDR) with a 72-bit data interface (for non-ECC 64-bit data width). Therefore, peak aggregate memory bandwidth is 12.8 GBps per FPGA.

The five FPGAs on the same module are organized into four compute FPGA and one control FPGA. The control FPGA has additional global interconnect interfaces and controls 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 Transceivers (MGTs) 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 channels is configured in software to run at 2.5Gbps or 3.125Gbps with 8B/10B encoding, and every 4 MGTs are channel-bonded into a physical Infiniband 4X (IB4X) electrical connector, to form a 10Gbps full duplex (20Gbps total) interface. The IB4X connections are AC coupled on the receiving end to comply with the Infiniband and 10GBase-CX4 specification.

Each BEE2 compute module can be used as a global communication tree node, connecting up to 16 other compute modules as its leaves, and up to two independent parent nodes. Using the 4X Infiniband physical connections, the compute modules can also form many other types of network topology, such as 3D-mesh. For applications require high bisection bandwidth random communication among many compute modules, the BEE2 system is designed to take advantage of the commercial network switch technology, such as Infiniband or 10G Ethernet. The crossbar switch network provides the highest throughput connection in the BEE2 system, with commercial switches currently reaching 244 4X ports in a single physical 9U chassis, aggregating to 4.88 Tbps bisection bandwidth. The regular 10/100Base-T Ethernet connection, available only 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 has been designed to run Linux OS on the control FPGA with full IP network stack.

Each BEE2 module is hosted in one of the 10 blades in a custom 8U rack-mount chassis. With 2 out the 10 blades reserved for AC/DC power supplies, each chassis packs 8 BEE2 modules, hence a standard 42 48U rack can host up to 40 BEE2 modules. With each BEE2 module designed to dissipate a maximum of 250 Watts—similar to a standard 1U dual-processor computer server.

Two BEE2 modules were manufactured and fully tested in early 2005. Since then we have developed rudimentary software and gateware to support several Radio Astronomy applications. One module is currently in production use at the JPL/Barstow Deep-space-network radio telescope installation by the UC Berkeley SETI project. More details of the BEE2 platform and its current use may be found in [6].

Rather than begin the RAMP project by designing yet-another-FPGA-board, we intend to use the BEE2 design for the RAMP 1 system. We will use the BEE2 board to figure out what the wish list of features for the RAMP 2 board will be. We will begin the design of RAMP 2 a year into this project. RAMP 2 will be based on a new design employing the Virtex 5 FPGA architecture (expected to be available beginning in 2006).

2.2 Emulation Methodologies

Microarchitectural techniques are typically evaluated by modifying an existing software cycle-level simulator, such as Simplescalar, to model the interaction of the proposed new hardware with a conventional superscalar out-of-order processor. Such a simulator is slow; RAMP is not. RAMP is also able to collect and process a tremendous number of measurements with little to no performance impact. Virtually all statistics can be gathered by dedicated logic and state within the FPGA and thus will generally not slow down the simulation. Questions such as "when does the number of active functional units drop below 1" can be asked and monitored across the entire program run. RAMP FPGAs will be surrounded by a large amount of high-speed RAM where statistics can be stored for latter perusal.

Rather than put disks on the same boards as the FPGAs, our plan is to use a seperate, network attached storage device (NASD) using either the Infiniband links or the 10 Gbit Ethernet links. Using such an off-the-shelf storage array should dramtically simplify the design and management of RAMP. We would use a virtual machine such as Xen running on each processor to present a simple, single, large virtual disk to each operating system. The NASD could then act as a block server to each VM, likely running on top of a file system in order to simplify management. A parameterized disk model determining the capacity and performance characteristics of the virtual disks would be supplied with this interface. Thus, our approach to disk storage is similar to our approach for main memory.

Detailed microarchitecture research can be accomplished at several different levels on RAMP.

Direct Implementation One use of the RAMP infrastructure would be to enable microarchitectural research with detailed RTL models of processors by providing a baseline synthesizable RTL model of a processor capable of running a full operating system, together with a full suite of test applications. Researchers proposing a new mechanism can test that mechanism in this infrastructure. This approach should provide a qualitative jump in the confidence the community can place in new ideas and evaluations. A full RTL model ensures that important pipeline interactions cannot be ignored, and effects on design area and cycle time can be readily observed. Crude area and delay estimates can be obtained by comparing FPGA implementations, but processor implementations on FPGA substrates scale quite differently than full-custom processor designs in terms of area, delay, and power. However, completion of a fully functional RTL design provides a much better basis for accurate area and delay estimation than a pencil and paper block diagram. The same RTL design can also be pushed through standard-cell synthesis tools to obtain more accurate estimates of area, delay, and power.

Although full RTL design is more complex than simply modifying a software simulator the RAMP team intends to provide two crucial components to make it tractable for a wide range of researchers. The first component is the open source "reference" gateware that implements a complete, functional, and *supported* RTL model for a complete MPP system. The second component RAMP provides is a clearly defined design standard, outlined in the next section, that makes it easy to compartmentalize hardware designs as to minimize their global impact on the RTL model.

Emulation Unfortunately, there will be some structures that cannot be implemented directly within the FPGAs. For example, modern microprocessors have well over 2MB of L2 cache memory. Since the largest FPGAs today have less than 1.25MB of block memory total, it will be impossible to place the L2 cache in its entirety on an FPGA. Such an L2 would have to be emulated using off-chip memory. If the L2 cache is associative in any way, multiple reads to external memory may also be required. Such structures will obviously incur more latency and thus will require processor core stalls to accurately simulate. By permitting such emulation, however, the space of architectures that can be evaluated using RAMP becomes much larger. The design standard proposed in the next section makes this type of emulation straightforward.

Another alternative is to implement the functionality of a structure in the FPGA and maintain the timing accuracy relative to the whole system. For example, one may model nodes in a parallel system with an FPGA-based functional model augmented with timing information and the interconnection network accurately. Transactions made on the interconnection network returns not only the result but also timing information which can be used by the system to calculate performance.

Simulation Some architectures, such as the ubiquitous Intel IA-32, do not have a freely available reference implementation and/or are too large/complicated to implement a full microarchitecture. In such cases, simulation is probably more appropriate. Most software-based cycle-level simulators are divided into an instruction set interpreter that models the effect of each instruction on architectural state, and parameterizable timing models that calculate the cycles required for each stage of execution. This division of responsibility reduces the effort required to model changes to the microarchitecture as only the timing models are modified.

Such a partioning can be used on RAMP as well[7], where the timing model is implemented in hardware on the FPGA and the instruction set interpreter can be implemented in software (enabling the use of off-the-shelf instruction set interpreters) or in hardware. By implementing each model half in the technology that best suits it, much higher simulation performance with very low resource usage is achievable. Such a system would be able to immediately implement a full system (a prototype runs the IA-32 instruction set and already boots Linux) and can more easily model complex microarchitectures with large amounts of state. For example, such a simulator could model large L2 caches since only the tags and not the data are stored in the FPGA. Of course, the timing models of such simulators require explicit verification for accuracy.

2.3 RAMP Gateware Design

A configured RAMP system models a collection of CPUs connected to form a cache-coherent multiprocessor. The emulated machine is called the *target*, and underlying FPGA hardware (e.g. BEE) is the *host*. In this section, we describe our proposal for a RAMP design framework to allow flexible, modular, cycle-accurate emulation of target machines. The framework must support both cycle-accurate emulation of detailed parameterized machine models and rapid functional-only emulations. The framework should hide changes in the underlying RAMP hardware from the module designer as much as possible, to allow groups with different hardware configurations to share designs and to allow RAMP modules to be reused in subsequent hardware revisions. We also require that the framework does not dictate the hardware design language chosen by developers.

The RAMP design framework is based on a few central concepts. A RAMP configuration is a collection of communicating *units*, where a unit is a relatively large number of gates. Example units include CPUs, coherence engines, DRAM controllers, I/O device interfaces, and network router stages. 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 has a single clock domain. The target clock rate of a unit is the relative rate at which it runs in the target system. For example, the CPUs will usually have the highest target clock rate and all the other units will have some rational divisor of the target CPU clock rate (e.g., the L2 cache might run at half the CPU clock rate). The physical clock rate of a unit is the rate at which the FPGA host implementation is clocked. In some cases, a unit might use multiple physical clock cycles to emulate one target clock cycle, or even require a varying number of physical clock cycles to emulate one target clock cycle. At least initially, we expect that the whole RAMP system will have the same physical clock rate (nominally around 100 MHz), perhaps with some higher physical clock rates in I/O drivers.

All channels are unidirectional and strictly point-to-point between two units. The two units at each end of a channel can have different target clock rates, but, at least for the initial RAMP standard, must have the same physical clock rate. Units 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 either a real message (more precisely, one target cycle's worth of a real message) or an idle token on each of its input channels. The basic execution model for a unit is: 1) wait for all input channels to have produced values for this unit's target cycle, and for all output channels to have space for a new message, 2) execute for one target cycle, 3) for each output, either send a new message or an idle token.

The channels have full flow control. At reset time, each channel is filled with a number of idle tokens equal to the channel latency. This scheme allows units to run at varying speeds while remaining logically synchronized.

Unit designers must produce the RTL code of each unit in their chosen hardware design language or RTL generation framework, and specify the range of message sizes that each input or output channel can carry. For each supported hardware design language, we provide 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 RAMP configuration tool from a RAMP configuration specification file, 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 latency, bandwidth, and 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 will also explore the option of allowing these parameters to be changed dynamically at target system boot time to avoid re-running the FPGA tools when varying parameters for performance studies. The configuration tool will build in support to allow 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.

A higher-level aspect of the RAMP design discipline is that the operation of a unit cannot depend on the absolute or relative latency of messages between units (i.e., all inter-unit communication is latency insensitive). We do not believe this unnaturally constricts unit design, as latency-insensitive protocols are commonly used in real hardware systems. We emphasize that units are intended to represent large pieces of a design and that it is not intended that channels will be used at a fine-grain, such as within a CPU. Any rigid pipeline timing dependencies must be contained within a unit. For example, primary caches will usually be implemented as part of a CPU unit.

Providing a modular design strategy within a unit is more difficult because of the dependence on each module on the detailed timing of other modules. Our approach in RAMP is to allow each class of unit to provide a design template within which various parameterized modules can be inserted. For example, CPU design templates could include one for a seven-stage pipelined in-order scalar processor and one for a nine-stage pipelined out-of-order super-

Table 3: RAMP project components and sources of development

Hardware	
Lab facilities	BWRC
RAMP 1 (BEE2)	Donation
RAMP 2 (BEE3)	Staff-designed, fabbed via donation
Network attached disk array	Donation
Gateware	
Bring-up, integration, validation & support	Staff
Performance monitor	Christos Kozyrakis
Cache coherency	Mark Oskin & James Hoe
Network model	Staff
CPU Model (PPC)	Arvind
CPU Model (x86)	Derek Chiou & James Hoe
Memory Hierarchy	Shih-Lien Lu
PCI Interface	Staff
Ethernet Interface	Staff
Diagnostic processor	Staff
Software	
Compiler (gcc port)	Staff
OS (Linux port)	Krste Asanovic & Christos Kozyrakis
Virtual Machine (Xen port)	Staff
Test suits & benchmarks	Staff

scalar processor with speculative execution and register renaming. There might also be similar design templates for pipelined cache-coherent directory controllers. We expect that various unit design templates will usually be primarily co-ordinated by different research groups, providing a natural distribution of effort.

2.4 RAMP Gateware models

Through the combined efforts of the Co-PI's, senior personal, paid professional staff, and students at participating universities, a set of baseline RAMP gateware configurations will be constructed. The goal of these designs is two fold: First, much like *sim-outorder* [4] served as a baseline simulator for hundreds of architecture research projects, these models should serve as the starting point for an entire community of architecture researchers. Second, the models will be supported, through the support of the paid professional staff listed on this grant proposal's budget. The list of infrastructure components and the groups responsible for each are shown in Table 3.

We intend to create four basic models, fulfilling six project steps. We will briefly describe each of these models below:

Uniprocessor/proxy kernel: Our first model is a simple single-processor model utilizing a proxy-kernel. Conceptually, this is precisely the gateware equivalent of *sim-outorder* [4]. The uniprocessor will be a configurable superscalar design. The memory system will be a straightforward multilevel cache hierarchy. For application support a widely utilized commercial ISA will be implemented, with the most likely candidates being SPARC, or PPC. For system-call support a "proxy-kernel" will be utilized. In the proxy-kernel configuration all system calls are trapped by the processor and passed to a controlling host processor for operation. Experience with software simulators suggests only two dozen system calls are required to execute the SPEC2000 benchmark suite. The host processor will implement these system calls, modify the targets memory state, and return the system call results to the simulated processor. The goal of this model is to have something up and running quickly on the RAMP hardware and to have something easily modifiable for those researchers interested in uniprocessor performance.

Multiprocessor/proxy kernel: The next phase of our gateware design is to extend the uniprocessor/proxy kernel configuration to support multiple processors. Our goal here is a straightforward extension: a simple MESI cache coherency protocol will be utilized; all processors will continue to call into the proxy kernel for system call support. The host processor will sequentialize all system calls to ease any interfacing issues. This baseline design should be useful for most basic CMP studies.

Uniprocessor/full kernel: In the past, architecture researchers have not often performed full system simulation. The reason is the complexity involved in booting a kernel on a simulator. It is widely accepted, however, that only simulating user-applications has skewed our communities results, particularly with such things as TLB miss and branch-prediction behavior. Hence our goal of booting a full kernel on the RAMP system. As a starting point we intend to implement straightforward hardware elements, i.e., those you find in software full-system simulators such as Bochs [9]. These are simple interfaces for a NIC, PCI controller, VGA display, interrupt controller, and so forth. For a kernel our goal is to boot Linux. When this model is completed the host will be become just another device this full system executing on RAMP is able to communicate with.

Multiprocessor/full kernel: Our final baseline model is a multiprocessor executing a full operating system kernel. This system is clearly the most complex of the four we intend to create. Starting from the uniprocessor we intend to add the required interprocessor interrupt dispatching and synchronization facilities required to boot a complex MP system.

Starting from the multiprocessor full kernel configuration, two further away mile-posts are included in our plan. The first is a multiprocessor system of at least 64 cores. This number was chosen because we expect the complexity of the system of 64 nodes to be significantly more than 4 or 8. Things such as the memory interface, interprocessor coordination bus, and operating system kernel will likely require extensive debugging and tweaks to operate well with 64 nodes. Another reason that 64 nodes will be significantly more complex than 4 or 8 is that will require multiple RAMP hardware boards. Hence the issues of partitioning the design across FPGAs and boards will need to be addressed in this step.

Once the 64 node version of the multiprocessor is complete we intend to apply our experience to the design of a baseline 1024+ core version. This is the pinnacle of the RAMP project. The main challenge here, we believe, we will be architectural. Producing a high-quality baseline 1024 node design for other researchers to use is going to require extensive architectural research. No one really has a concept yet of what a desktop 1024 node processor should look like. We expect extensive modifications to the memory, I/O, and coherence protocols will be required to produce a quality baseline model.

3 Project Management

The success of the RAMP projects hinges on the collaboration of a large number of participants. In this section, we first discuss our arrangements to maximize the efficiency of PI, co-PI, and senior personnel participation and interactions. We next justify our requirements for two full-time engineers to supplement the voluntary efforts of the senior personnel. Finally, we propose a 3-year schedule for key milestones in RAMP 1 and RAMP 2 development.

Although proposals are often cut, we would like to point out that this proposal has no padding in anticipation of a cut. This project cannot be accomplished without two full-time engineers. It is an $800,000 proposal only because (i) we were able to raise $800,000 from an industrial partner, (ii) many senior personnel are planning on building the gateware and software infrastructure without specific financial support from this proposal and (iii) the BWRC overhead is low. Please consider funding this proposal as budgeted.

3.1 Senior Personnel Participation

The management of this project is composed of two pieces. First, there is the overall effort being conducted by the PI, co-PI, and senior personnel. This effort is spread out across seven institutions and three time zones. Second there is board development and engineering staff management. This is centralized at the BWRC.

To facilitate this geographically diverse effort our team meets regularly via a conference call. Furthermore, we intend to meet at least twice yearly, once at the Workshop on Architecture Research with FPGA-based platforms held in conjunction with the High Performance Computer Architecture (HPCA) conference (held in February), and once the week of the Hot Chips conference (held in August).

In addition, we intend to organize a yearly retreat for RAMP participants. These retreats will be held in geographically diverse areas (the bay area, Boston, Seattle) that also have a significant local high-tech industry. Outside academics, industry leaders, RAMP students and senior personnel will gather for three days of intense discussion on the current state and future progress of RAMP. In the experience of the PIs, these retreats have proven enormously valuable to research projects. The main benefit is they allow project outsiders to comment and provide helpful feedback to project participants. Another benefit is the intense focus on the research project from so many participants focuses enormous creative energy on the problems at hand. While these retreats are expensive to organize and run, no additional funds are requested in the budget to support them. Our intention is to solicit additional industry support to operate them.

Finally, we will also establish a web-based project repository at `www.sourceforge.net` that allows the senior personnel initially and the whole community later on to contribute fixes, enhancements, and extensions to the RAMP infrastructure.

3.2 Personnel

We are asking for two full-time engineers with skills in software and hardware to build and debug the scalable prototype and make sure we have a fully coherent implementation. One of the keys to success will be assembling an attractive collection of software and gateware (FPGA configuration bitstreams) that can run on the system out of the box. We expect that the majority of the RAMP software and gateware will be provided by our collaborating developers, however, full-time engineers are necessary to create a successful system and drive the various volunteers to do their part. They will also design and debug RAMP 2 boards and systems.

Roughly speaking, the Hardware Platform Engineer will be responsible for the host system design and the software/gateware specific to the host, with the RAMP Software Engineer being responsible for the target machine specific software/gateware. Their specific responsibilities are listed below.

	Hardware Platform Engineer	**Software Engineer**
Year 1	Manage RAMP 1 (BEE2) hardware fab. RAMP-1 hardware manufacturing testing Design physical packaging for RAMP 1 Distribution of RAMP 1 platforms Porting and maintenance of Linux on RAMP 1 host Generic memory and network interfaces (gateware)	Tools to generate inter-unit channels Tools to generate standard unit wrappers Development of LAN based access control software Uniprocessor proxy kernel and libraries Multiprocessor proxy kernel and libraries
Year 2	RAMP 2 FPGA board design and layout Interaction with manufacturers for prototype fab. Prototype board- and system-testing Design physical packaging for RAMP 2 Develop Diagnostic manufacturing test suite	Uniprocessor full OS kernel and libraries First multiprocessor full OS kernel and libraries Integration of modules from contributing developers First "out-of-the-box" multiprocessor implementation
Year 3	Physical packaging for RAMP 2 systems Porting of Linux to RAMP 2 Low-level modules for memory and network Diagnostic test suite for preventive maintenance Handoff to third-party for fab. and distribution	Mantenance of multiprocessor OS kernel Development of interactive debug and monitoring Second "out-of-the-box" multiprocessor implementation
All Years	System documentation Website content creation and maintenance Point of contact for user/developer questions	

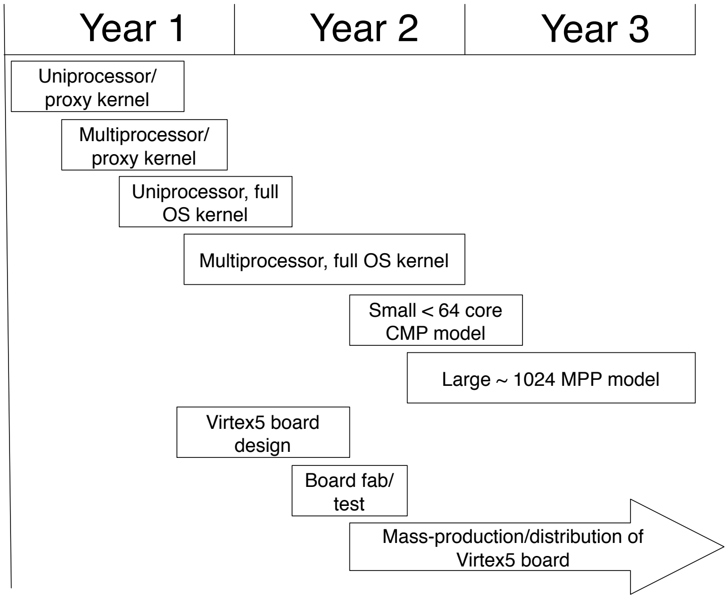
Year 1
Year 2
Year 3
Uniprocessor/ proxy kernel
Multiprocessor/ proxy kernel
Uniprocessor, full OS kernel
Multiprocessor, full OS kernel
Small < 64 core CMP model
Large ~ 1024 MPP model
Virtex5 board design
Board fab/ test
Mass-production/distribution of Virtex5 board

Figure 2: RAMP project timeline

3.3 Schedule

Figure 2 depicts the proposed schedule of work for the RAMP project. The project encompasses two major components: (1) construction of base machine configurations capable of supporting architecture and systems research; and (2) development, fabrication and, through partnership with a commercial vendor, widespread distribution of a low-cost hardware platform.

Development of the modeling infrastructure is broken down into six major steps, each of which builds upon the prior. Naturally some amount of overlap in the development will occur and this is depicted in the time line. These six steps are: (1) uniprocessor "proxy kernel" configuration; (2) multiprocessor "proxy kernel" configuration; (3) uniprocessor full-system; (4) multiprocessor full-system; (5) scaling of the multiprocessor system up to 64 nodes; (6) scaling of the multiprocessor system to 1024+ nodes.

Our expectation is that steps (1) and (3) can leverage significant quantities of existing IP already freely available. The projects value-add to this existing IP is fitting it within the design standards discussed earlier and providing the proper hooks into the modules for debugging, logging, and educational support (easy parameterization and design space exploration).

Steps (2) and (4) will leverage our efforts from (1) and (3), but focus on multiprocessor support. Significant effort will be spent on implementing the interconnect and coherence protocols that are required to make these systems function correctly.

Steps (5) and (6) naturally build on (4), yet adapt them to work on our custom designed Virtex 5 prototype board. The ultimate goal of this project—a simulation infrastructure for MPP systems of 1024+ nodes requires the use of several of these boards and we expect significant design challenges to appear in the IP when scaling up to this level. For this purpose over a year of time is allocated to the proper development, debugging, and supported deployment of the final system.

Concurrent with the IP development our team intends to design, fabricate, and in partnership with Xilinx, deploy a custom Virtex5-based emulation board. The goals of this board are: low cost for educational and research use, high-density to support a large number of emulated processing cores, extreme flexibility to support the wide variety of research projects being undertaken in computer architecture, and easy debugging and bootstrapping to ease the transition of our community away from software-based simulators and towards hardware based prototypes.

4 Projects and Outreach

In this section, we describe a few of the possible research projects that will be made possible by the RAMP framework. We also describe a few of the possible uses of RAMP in engineering education and its broader impact on the research community.

4.1 Project Summaries

The collaborators and supporters of this proposal plan to initiate the following projects as soon as RAMP is available. Additional details are provided in the letters of support of the corresponding authors.

4.1.1 Internet In A Box (Patterson, Shenker – Berkeley)

Distributed systems such as the Internet are at the core of information technology. However, it is extremely difficult to systematically study such large-scale systems in a controlled environment with realistic workloads and repeatable behavior. The RAMP infrastructure can provide a highly controllable environment in which to test distributed system development. Although a large collection of commercial PCs connected by off-the-shelf switches can provide high throughput, the resulting distributed system is difficult to observe and control. With a RAMP configuration, a 1000 node distributed system can be single-stepped, a message at a time, to allow a detailed view of the effects of distributed protocols. Checkpoint facilities can save the intermediate state of the parallel system, to allow developers to repeatedly replay an interesting event. Such an environment will allow significant advances in areas such as new Internet architectures, distributed hash tables, routing algorithms, distributed middleware, etc.

4.1.2 Research on Multiprocessor Software (Kozyrakis – Stanford)

A critical limitation for large scale multiprocessors is the difficulty of parallel programming. Similarly, there is limited understanding on efficient compilation and runtime techniques for systems with hundreds of nodes. RAMP would provide an excellent platform for research on programming models, compilation, and operating systems for large scale multiprocessors. The size, clock frequency, and observability of RAMP allows researchers to analyze applications with large datasets and identify the real research challenges in these areas. The open source software that runs on the baseline RAMP configuration reduces the engineering effort necessary to experiment with new compilation and runtime optimizations. We can also easily experiment with instruction set and microarchitecture modifications that provide direct support for innovative programming and scheduling techniques.

Moreover, RAMP provides a wide range of researchers in application areas such as graphics, databases, and machine learning with the infrastructure to develop, tune, and demonstrate new algorithms that scale to hundreds of processors. The compute power available in systems of such scale can enable major in these fields and new collaborative efforts between architects and software developers.

4.1.3 Transactional Memory Systems (Kozyrakis – Stanford, Lu – Intel)

Transactional memory provides a new model for shared-memory multiprocessors that simplifies parallel programming. Transactions provide a high-level, intuitive abstraction for specifying non-blocking synchronization, speculative parallelism, and failure atomicity in parallel programs. We are investigating architecture and system software mechanisms that support pervasive use of transactions in parallel programs. Moreover, we are exploring transaction-based programming techniques that allow simple parallel programs to scale from small-scale to large-scale multiprocessors. The RAMP infrastructure will allow us to prototype transactional memory architectures and provide a fast platform for experimentation with programming models and compilation techniques with large-scale applications. Moreover, RAMP will enable thorough investigation of hardware implementation for transactional memory.

4.1.4 Fault-tolerant Systems (Hoe – CMU)

The Computer Architecture Lab at Carnegie Mellon (CALCM) is engaged in research to develop a practical fault-tolerant architecture for distributed shared-memory multiprocessor servers (`http://www.ece.cmu.edu/ truss`, NSF ACI-0325802). The target architecture is based on a distributed system design to achieve cost and performance scalability. We are investigating mechanisms such as computation fingerprinting, distributed lockstepping and distributed memory parity to support distributed redundancy of processing, memory and I/O throughout the system to tolerate any single component failure (whether permanent or transient). The RAMP prototyping platform will deliver the necessary speedup (over a software-simulated system) to enable us to evaluate the proposed mechanisms in a complete system executing real applications. Moreover RAMP provides a full-system platform ideal for fault-injection studies.

4.1.5 Data-Flow Architectures for C (Oskin – U. Washington)

WaveScalar is a highly distributed data-flow architecture being developed at the University of Washington. The basic idea is that instead of building a complex monolithic processing core like the Intel Pentium4, one designs a small processing tile. These tiles are then replicated across the silicon die. Instructions from an application are mapped to these processing tiles for execution. Once an instruction executes it communicates its result to its dependents using an on-chip dynamic packet network. After execution the instruction stays at the processing tile for possible subsequent execution, for instance if the instruction is in a loop. Through careful design of the instruction set this substrate of processing elements is able to efficiently execute applications. The basics of this architecture were developed two years ago and we are now exploring all of the unanswered questions raised from its design. The Wavescalar team will port the WaveScalar RTL to that board and future development will occur on it. The reason is two fold. First, there is the reduced cost and risk transfer. Second is once the RTL is ported we can share WaveScalar more easily with other researchers.

4.1.6 Network processors (Chiou – UT Austin)

One class of network processors is essentially a multi-core parallel computer with special purpose memories, memory interfaces, datapaths and functional units. Such processors are quite difficult to simulate quickly and thus, like other systems we have described, the simulators tend to be only useful until hardware is available. Generally, simulation is only used until hardware is available, or when tracking down a problem difficult to duplicate or observe within the hardware.

RAMP would be an ideal platform to simulate such processors. The processor cores/microengines tend to be very simple and thus easy to implement on the FPGA. Such processors almost always have slower clock rates than general-purpose microprocessors; thus, the RAMP-based systems will tend to be far closer in performance to real hardware. Coupled with the fact that most network processors rely on extensive parallelism, RAMP-based network processors could potentially have the same or even better performance than real network processors. The platform would encourage more scalable architectures that would seamlessly allow the addition of additional resources that would automatically result in a correspondingly higher performance.

4.1.7 Scalable Multithreaded Architectures (Asanovic – MIT)

Merely connecting a thousand existing processors is unlikely to lead to an efficient large-scale system. It is important to revisit the computing, memory, and interconnect components of future multiprocessors and engineer the feature necessary for performance and energy efficiency at that scale. The Infini-T project investigates techniques that allow a thousand-node system to support a million threads. The techniques include support for threads in local memories, word-level memory protection, support for rendezvous synchronization and transactions. RAMP provides a prototyping framework for Infini-T that allows the development of the software stack and the exploration of hardware implementation alternatives.

4.1.8 Application Specific Accelerators and Architectures (Dally, Meng, Kozyrakis – Stanford)

Application specific accelerators and architectures provide an energy and area efficient mechanism to use the exponentially increasing transistor budgets to improve the performance and extend the functionality of important computing tasks. We are investigating application specific architectures for applications such as bio-computing (alignment of multiple DNA sequences, statistical signal processing for neural decoding), machine learning (Bayesian and neural networks), and software security. RAMP allow us to directly implement and measure the efficiency of these architectures.

4.1.9 Novel Architectures for Scientific Computing (Yelick, Kramer – NERSC)

Most modern computers are optimized for low-end commercial applications or medium-scale enterprise systems. Consequently, scientific disciplines such as biology, chemistry, and geophysics that require real-time processing of huge datasets are currently limited by computational capabilities. RAMP can provide a platform to evaluate new architectural features for scientific computing ranging from extensions of conventional organizations to radical ideas such as direct FPGA computing. Moreover, RAMP enables the use of full applications within days of the feature being develop. It allows to "edit-compile-run-debug" the entire computing system along with detailed performance measuring features.

4.1.10 UNUM: A tinker-toy approach to building PowerPC microarchitectures. (Arvind-MIT)

We will demonstrate that it is possible to synthesize many different PowerPC models (both existing and new variants) quickly by using a library of microarchitectural IP blocks. The IP blocks and modules that we propose to develop include instruction decoding, branch prediction, speculative execution structures, ALUs, L1 and L2 cache structures, and cache-coherence engines. This research will not only provide PowerPC "gateware" for others to use, but will also shed light on how IP blocks should be written to be easily modifiable and reusable.

4.2 Educational Advancement

We expect that a fully developed RAMP system will significantly enhance the educational environment at participating universities. Education occurs both within the classroom and, particularly for graduate students, outside in less formal settings. In this section we will elaborate on how we expect RAMP to be used in both of these environments.

Student training. Easy access to appropriate tools do more than just enable research, they serve as the genesis of it. Witness simplescalar [4]. Citeseer reports over 351 scholarly publications referencing its use. Simplescalar spawned an entire generation of graduate students trained on superscalar processor research.

With the coming age of chip multiprocessors a new generation of graduate students must be trained. This new generation of fresh PhD's are needed by the computing industry to solve the programming, architecture, and systems challenges desktop MPP systems will have. Studying large scale MP systems purely in simulation is intractable. By being both fast and accurate, RAMP is a platform that will enable this training and spawn new research.

In addition to graduate student research, if we are funded we will apply for REU grants so that undergraduates can participate in the research to realize the RAMP vision.

Classes. While individual student training and spawning research is an important goal of the RAMP project, facilitating classroom education is equally important. Classes serve to educate beyond the specialists in the area and create a broad group of professionals experienced with MPP systems. Here in this section we'll limit ourselves to discussing the impact on only the three most related classes taught at undergraduate and graduate levels. These are architecture, parallel systems, and digital systems design.

- Architecture: Current undergraduate and graduate architecture classes are limited by the available infrastructure. Those that make use of FPGAs lack the ability to teach about the "whole system" including architecture /

operating system interaction. Those taught using only simulation tools lack the ability to impart to students the true complexity, design challenges, and hardware opportunities available.

RAMP would change this. Starting from the base RAMP system the PIs included on this proposal will develop classroom projects designed to teach about the interaction of the operating system and architecture as well as the design and tuning of real microprocessors. Since a community of educators will be built that utilize the same base hardware and software platform this community can share classroom exercises and thereby gain leverage.

- Parallel systems: If parallel computing classes are taught at all in universities, they are not using the systems of the future, but make do with the computers of the past. Parallel architecture is often taught as a "history lesson", focusing on reading papers about significant systems. Parallel programming is forced to make do with whatever make shift parallel system the educators have access to – a cluster of workstations and/or a small SMP. The scope of class examples, assignments, and projects is limited to small-scale uniprocessor systems.

  RAMP can make a significant impact on the teaching of parallel systems. First, for parallel architecture classes it will enable advanced class projects that explore the processor, bus, coherence protocol, and I/O storage system of MPP systems. For parallel programming classes the availability of canned "model" RAMP systems with complete infrastructure: Linux operating system, GNU tools, etc., facilitates class room projects built around software development for future instead of historic parallel computers. Moreover, RAMP will allow both architecture and programming courses to focus on large-scale systems and provide students with hands-on experience on the real challenges with tens, hundreds, and thousands of nodes in a parallel system.

  Being educators as well as researchers, the PIs are keenly interested in facilitating classroom experience with MPP systems. To this end, we intend to make the base RAMP system components as parameterizable as possible. This will enable students to explore variations on the base system design through easy change of the parameters. Furthermore, the core design standard facilitates a compartmentalized design approach. This will enable student projects that alter and extend the base RAMP system with minimal impact on the entire system fragility.

- Digital systems design: Advanced undergraduate and graduate level digital systems design is taught with FPGAs. RAMP has the potential to provide a canned system with tested infrastructure to serve as a starting point for these classes. How RAMP can change these classes is best illustrated by example. Currently, we may ask students to design the processing and communication hardware for a sensor network node that works with input data embedded an on-chip RAM. With RAM, we would ask them to build the sensor network node, run a real network stack, and demonstrate it in a network configuration with hundreds of nodes. This presents a project that is much closer to a real-world, full-system experience. The PIs on this proposal teach digital systems design and will develop a number of integrated systems projects that can be shared with the entire RAMP community.

- Broader educational impact: Once RAMP is fully adopted in these "core" related areas, we see it achieving a broader impact throughout a department's curriculum. In particular, applications classes such as databases, graphics, and AI should be training undergraduates on how these areas will be impacted by future desktop MPP systems. If RAMP were available, these classes can use it to explore how their respective algorithms and software artifacts execute on models of future processing systems.

Such training of undergraduates could help answer the question during the Hot Chips keynote address of where are we going to find the programmers that understand parallel computing.

4.3 Research Outreach

Although the universities involved in the development in RAMP and the initial group of external supporters come from the usual leading research institutions, we believe the low cost and large scale will allow a much wider range of departments to have their own massively parallel processor for use in courses and in research.

If the RAMP 2.0 can be sold for $10,000 per board, including FPGAs and DRAMs, then departments can have a 128 to 256 parallel processor for the cost of a few fully loaded PCs. We believe that parallel processing can rapidly advance when most CS departments can afford to have their own MPP, where you don't have to ask permission from anyone to try crazy ideas and you can run programs as long as you want. Besides making boards and software directly available for purchase, remote users can sign up to run large experiments on an 1024 node or larger system at BWRC.

In addition to broadening the types of departments to have access to the large scale parallel processing, the RAMP project will also engage underrepresented groups. For example, some women and under-represented minorities wrote letters of support, so these supporters will surely be involved. Also, many grad students on the developers projects are from underrepresented groups. For example, most of the developers of Infini-T at MIT are female, while the main developers for the FPGA frameworks at Stanford are African-American.

5 Related Work

The proceedings of any recent computer architecture conference provides an unequivocal testament to the dominance of software simulation as a design evaluation vehicle. The dramatically lowered risk and expense incurred in simulation-based investigations has enabled many more ideas to be studied by a wider community. Nevertheless, a pressing concern for the continued effectiveness of simulation-based research is the ever growing simulation turn-around time. Due to the complexity of modern computer systems, the detail and complexity of research simulators have greatly outpaced even the exponential performance improvements of the microprocessors used to run the simulators. A "fast" cycle-accurate simulator of a microprocessor and its memory hierarchy (e.g., sim-outorder [4]) manages a throughput of a few hundred kilo-instructions per second on a 2005 workstation. To put this in perspective, the average simulation time of SPEC2000 benchmarks is over 5 days per benchmark—the longest SPEC2000 benchmark requires 24 days. Consequently, simulation-based computer architecture studies invariably incorporate some methodology to curtail simulation turn-around time, with varying degrees of impact on the quality of results.

Moving forward, new factors will collude to further exacerbate the already prohibitive simulation turn-around time. First, as computers become faster, benchmarks designed to measure them must also be lengthened to maintain a minimum measurement period on the real hardware. For example, the planned SPEC2005 benchmarks have a stated target of "no less than 600 seconds on a machine with a SPEC CPU2000 baseline metric of $\approx$700 for integer codes and $\approx$900 for floating point code." Given simulation performance available today, even the shortest SPEC2005 benchmark would require over a month of simulation time. Second, as computer designs migrate toward greater parallelism at both the system-level and the chip-level, simulation throughput will be proportionately reduced because the concurrency in hardware must be interpreted sequentially by software simulators. The latest generation of execution-driven timing simulators of cache-coherent multiprocessor systems attain only up to several tens of kilo-instructions per second aggregated over all simulated processors [14, 10, 3]. As such, any attempt to simulate realistic benchmarks running on non-trivially sized parallel systems—say greater than 100 nodes— would be an exercise in futility.

Recent efforts such as SMARTS/TurboSMARTS [16, 15] and SimPoint [13] have successfully applied simulation sampling to reduce the average simulation time per SPEC2000 benchmark to just hours, or even minutes, and yet still produce meaningful performance measurements for design evaluations. But the current success in accelerating simulation by sampling is limited to the simulation of uniprocessor systems executing fixed benchmarks. Efforts to extend simulation sampling to the simulation of multiprocessor systems are on-going [2]; early experiences would only suggest it is a much more challenging problem (due to non-deterministic executions when repeating the same benchmark on different simulated hardware). Moreover, sampling-based simulation does not help with the significant efforts in software development and debugging for the operating system, programming model, compiler, and applications that is necessary for large scale systems.

The RAMP project will overcome many of the above software simulation limitations. RAMP prototyping will deliver improved performance by emulating the intended hardware behavior in FPGAs rather than by indirect software interpretation. The more concurrency in the hardware design under investigation, the more exaggerated the performance difference between RAMP prototyping and software simulation. The level of benchmarking performance made possible by RAMP should encourage more diligent efforts in understanding a new hardware design as well as in tuning the software for the new hardware. Another important offshoot of RAMP's performance is of course the ability to evaluate full-scale system performance under real workloads and operating systems. Nevertheless, to make these new-found capabilities appealing to the computer architecture research community, the proposed RAMP effort must take care to limit the time, effort and expense required by each researcher to obtain results. We intend for RAMP to be the next computer architecture research vehicle of choice.

There are many other approaches to simulate parallel systems. All the approaches we are familiar with, however, suffer from one or more of the following problems.

- **Slowness.** RAMP promises very high performance. Many alternate schemes run much slower than RAMP, often because of software simulation speeds or software emulation of specific functionality.
- **Target Inflexibility.** When a scheme uses standard microprocessors as the basic computation engine, it can simulate that microprocessor's ISA and microarchitecture accurately and quickly, but any difference has substantial accuracy and speed implications. It becomes prohibitive to model processors much different than the host processor.
- **Inaccuracy.** RAMP is capable of implementing many systems in their full glory. Some other schemes are inherently inaccurate due to approximations that must be made to get the target system to run. There are often tradeoffs that have to be made. For example, target inflexibility can be reduced at the cost of greater inaccuracy and/or slowness.
- **Scalability.** RAMP is capable of scaling to at least several hundred processors. Many other systems have limited scalability.
- **Unbalanced computation/communication.** RAMP has tremendous communication bandwidth both within an FPGA and between FPGAs. Many other systems do not, making the ratio of computation to communication much lower and thus limiting the range of target systems that can be accurately implemented quickly.

5.1 Other Approaches

Several competing approaches are very briefly summarized and their issues relative to RAMP are discussed below.

Clusters Clusters of workstations, built from standard PCs connected by some sort of network, can provide significant processing power inexpensively, but almost always have unbalanced computation/communication. In addition, since such a system uses **standard microprocessors**, they suffer from some combination of target inflexibility, inaccuracy and slowness.

PlanetLab Existing distributed environments (like PlanetLab [8]) are very hard to use for development. The computers are live on the Internet and subject to all kinds of problems such as security, reliable connectivity, etc. and there is no reproducibility. The environment is not reservable, the operating system and routing are not changeable and they are very expensive to support limiting their size to a few hundred nodes. There is the standard combination of target inflexibility, speed and inaccuracy as well as unbalanced computation/communication.

Wisconsin Wind Tunnel The Wisconsin Wind Tunnel simulated shared memory on a Thinking Machines CM-5 by trapping to software on corrupted ECC bits [11]. Software implements the coherency protocol. Such a scheme runs at full speed on cache hits but slows down dramatically on cache misses since they are serviced by software creating inaccuracies and/or slowness problems. Such a scheme is also target inflexible, and time on a CM-5 is not broadly available to researchers.

Software Simulator Software simulators are three to four orders of magnitude slower than RAMP-based implementations of single processors. Simulating more processors will result in at least a linear slowdown in software (assuming a single host processor), but will not slow down in RAMP (as long as there are sufficient hardware resources.)

Quickturn, etc. Several companies such as Cadence (Quickturn and Palladium), Axis, IKOS and Tharas sell field-programmable gate array (FPGA) based hardware emulators or multiprocessor-based hardware emulators to improve register-transfer level (RTL) simulation performance. In general, synthesizable Verilog or VHDL RTL descriptions are compiled to an FPGA array that then emulates the cycle and bit-accurate hardware behavior. Large RTL designs must first be partitioned and then mapped across many FPGAs (sometimes numbered in the thousands). These platforms are intended to speed up RTL simulation (claimed speed of up to 100M cycles per second or more) for RTL functional and performance verification. The proposed RAMP 1 and RAMP 2 hardware would be comparable in capacity to the most capable commercial platforms. However, a fundamental difference is RAMP's emphasis on emulating high-level architectural behaviors rather than cycle-accurate RTL netlists. RAMP hardware and its supporting software provide useful abstractions, such as point-to-point channels and virtualized clock domains, to facilitate easy integration of architectural modules captured at a higher level of hardware abstraction. Overall, RAMP targets architecture and software research instead of simulation of a single RTL design.

RPM RPM developed at USC in the early 1990s is a configurable emulation platform for a multiprocessor system with no more than 8 processors and thus has scalability issues [12]. The main goal is to evaluate different MIMD systems having various cache protocols and consistency models. Due to cost constraints, only the memory controller in RPM is implemented with configurable hardware while processor cores, main memories and interconnect are off-the-shelf resulting in target inflexibility. The relative speed of RPM components is varied to emulate the impact of component technologies.

6 Results from Prior NSF Support

NSF Prime Award #CCR-0205471, UC Berkeley subaward no. #81-1052776, "Decentralized Streaming Architecture (DSA) for High Capacity Systems," PI: Andre DeHon, Caltech. Co-PI: John Wawrzynek, Prime award, $609,805, subaward (UCB) amount was $279,805, 10/1/02-9/30/05. The main theme of this research was to investigate highly parallel machine abstractions, models of computation, and their implementation on reconfigurable computing fabrics. Whereas a sequence of primitive instructions was the core principle in traditional ISA models, we envisioned stream-connected graphs as the model for Decentralized Streaming Architectures. DSA is a single, unifying system model for future computing systems which scales from modest, single-chip, single-processor systems, to multiprocessor ICs, to a wide range of heterogeneous System-on-a-Chip designs, to large-scale, heterogeneous, multi-component systems that evolve over decade long life-cycles. At Berkeley, we developed the SCORE Model (Stream Computations Organized for Reconfigurable Execution). SCORE naturally supports hardware virtualization and was developed as an abstraction for dynamically reconfigurable architecures. We also extended SCORE to be an organizing methodology for large scale digital system design, where independent computation units communicate asynchronously. The concepts in SCORE and other similar models has lead to the RAMP design framework described in this proposal. This DSA subaward also lead to the development of the BEE2 FPGA-based reconfigurable computing platform and resulted in five refereed journal and conference papers, two PhD Dissertations, and 2 masters theses.

NSF ITR AWARD #CCR-0085899, "Taming the Data Flood: Systems that Evolve, are Available and Maintainable," $2,399,802, 10/1/2000 - 9/30/2003. PI: David Patterson. This ITR award is the primary funding at Berkeley that led to two high-profile projects: OceanStore and Recovery Oriented Computing (ROC). The faculty involved included Katherine Yelick and John Kubiatowicz at Berkeley and Matthew Merzbacher at Mills College. OceanStore is a global persistent data store designed to scale to billions of users. It provides a consistent, highly available, and durable storage utility atop an infrastructure comprised of untrusted servers. OceanStore has led to the publication of 9 refereed journal and conference papers, including a best student paper award. The Recovery-Oriented Computing (ROC) project is a joint Berkeley/Stanford research project that is investigating novel techniques for building highly dependable Internet services. ROC emphasizes recovery from failures rather than failure-avoidance. This philosophy is motivated by the observation that even the most robust systems still occasionally encounter failures due to human operator error, transient or permanent hardware failure, and software anomalies resulting from "Heisenbugs" or software aging. The ROC research resulted in 15 refereed journal and conference papers plus the cover of Scientific American.

References

[1] James Ball. The Nios II Family of Configurable Soft-Core Processors. In *Hot Chips 17*, August 2005.

[2] Kenneth Barr, Heidi Pan, Michael Zhang, and Krste Asanović. Accelerating multiprocessor simulation with a memory timestamp record. In *IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, TX, March 2005.

[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] C. Chang, K. Kuusilinna, B. Richards, and R.W. Brodersen. Implementation of BEE: a Real-time Large-scale Hardware Emulation Engine. In *Proceedings of FPGA 2003*, pages 91–99, Feb 2003.

[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] Derek Chiou. FAST: FPGA-based Acceleration of Simulator Timing Models. In *Proceedings of the Workshop on Architecture Research using FPGA Platforms, held at HPCA-11*, February 2005.

[8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An overlay testbed for broad-coverage services. *In ACM Computer Communication Review*, July 2003.

[9] K. Lawton. Bochs. http://bochs.sourceforge.net.

[10] 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.

[11] S. Shubhendu Mukherjee, S. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. Hill, J. Larus, and D. Wood. Wisconsin Wind Tunnel II: A Fast and Portable Architecture Simulator. In *Workshop on Performance Analysis and its Impact on Design (PAID-97)*, June 1997.

[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] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct 2002.

[14] Simflex: Fast, accurate & flexible computer architecture simulation. http://www.ece.cmu.edu/ simflex/flexus.html.

[15] Thomas F. Wenisch, Roland E. Wunderlich, Babak Falsafi, and James C. Hoe. TurboSMARTS: Accurate microarchitecture simulation sampling in minutes. Technical Report CALCM-2004-3, Carnegie Mellon University, November 2004.

[16] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003.