RAMP BLUE: A MESSAGE-PASSING MANYCORE SYSTEM IN FPGAS

Alex Krasnov, Andrew Schultz, John Wawrzynek, Greg Gibeling, Pierre-Yves Droz

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA, USA
email: {akrasnov,alschult,johnw,gdgib}@eecs.berkeley.edu, droz@ssl.berkeley.edu

ABSTRACT

We are developing a set of reusable design blocks and several prototype systems for emulation of multi-core architectures in FPGAs. RAMP Blue is the first of these prototypes and was designed to emulate a distributed-memory message-passing architecture. The system consists of 768–1008 MicroBlaze cores in 64–84 Virtex-II Pro 70 FPGAs on 16–21 BEE2 boards, surpassing the milestone of 1000 cores in a standard 42U rack. An architecture based on point-to-point channels and switches using a combination of custom and generic hardware provides the functionality. Virtual-cut-through dimensional routing on one of two hybrid topologies with virtual channels provides the connectivity. A control network with a tree topology provides management and debugging capabilities. A software infrastructure consisting of GCC, uClinux and UPC allows running off-the-shelf applications and scientific benchmarks. Initial performance is encouraging for emulation purposes. In this paper we report on the design and implementation of RAMP Blue and discuss our experiences and lessons learned.

1. INTRODUCTION

Research Accelerator for Multiple Processors (RAMP) is a multi-university project intended to define and create the next generation of tools for computer-architecture and computer-science research [1]. RAMP seeks to leverage the high degree of parallelism and density scaling afforded by modern FPGAs to emulate existing and new highly parallel computer systems.

The first large scale system built as a demonstrator and test system for RAMP is RAMP Blue. The goal of the RAMP Blue project was to build a cluster of 256–1024 simple processors capable of running off-the-shelf message passing applications and scientific benchmarks. The processor nodes are instantiated within the FPGAs of a BEE2 (Berkeley Emulation Engine 2) cluster and communicate via a custom network.

RAMP Blue is a demonstrator system, designed to move forward the development of critical RAMP infrastructure and provide insight into the problems with and limits of FPGA emulation of massively parallel multi-core architectures [2]. In this paper we summarize the design and implementation of RAMP Blue and document some critical BEE2 infrastructure shared by RAMP Blue and other applications.

Section 2 documents the gateware[1] and software infrastructure developed for the BEE2. Section 3 describes the design and architecture of RAMP Blue. Section 4 describes the implementation and status of RAMP Blue. Section 5 concludes and discusses future work.

2. BERKELEY EMULATION ENGINE 2

The BEE2 is a second-generation FPGA board developed at the Berkeley Wireless Research Center (BWRC) and was designed for a broad range of applications, including real-time DSP, scientific computing and high-performance reconfigurable computing [3]. This broad domain is reflected in the BEE2's architecture, shown in Figure 1.

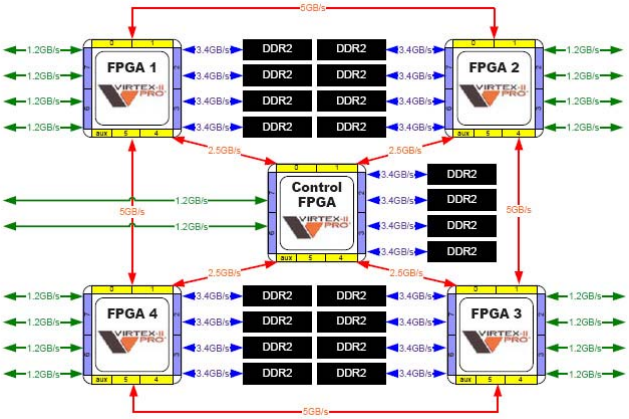


Fig. 1. General architecture of BEE2 module.

The BEE2 contains five Xilinx Virtex-II Pro 70 FPGAs,

[1]The term "gateware" refers to design files, typically written in a hardware description language, such as Verilog or VHDL, that instantiate hardware circuits within the FPGA.

1-4244-1060-6/07/$25.00 ©2007 IEEE.

the latest devices available at design time, each containing two PowerPC 405 hard cores and connected to four independent 72-bit DDR2 banks, capable of a peak throughput of 3.4 GBps. The four "user" FPGAs, are connected in a 5 GBps ring. The fifth, "control" FPGA, is connected in a 2.5 GBps star with the user FPGAs. The control and user FPGAs have two and four 10 Gbps high-speed serial I/O links off-board, respectively. These serial links run to 10GBASE-CX4 connectors, which allow Infiniband, 10 Gb Ethernet or XAUI (Ten-gigabit Attachment Unit Interface) connections over fiber or copper. Finally, a robust set of peripherals, including RS232 and Ethernet transceivers, are connected to the control FPGA, allowing the BEE2 to run Linux.

BEE2 designs can leverage many existing IP cores for common peripherals; however, there are several unique characteristics of the BEE2 that necessitated the design of custom gateware and software as described below.

2.1. Gateware

The most complicated and vital piece of gateware is the DDR2 memory controller. At the lowest level each DIMM has a heavily pipelined independent controller supporting simple bank management. In front of the low-level controller is the user memory interface, a common interface acceptable to a wide variety of BEE2 users. The interface is implemented as a set of asynchronous command and data FIFOs that provide buffering and decouple the user clock domain from the 180 MHz DDR2 clock domain. A switch multiplexes multiple copies of this interface using a simple round-robin priority scheme to allow different user logic to share the same DIMM.

2.2. Bootstrapping and Configuration

The control and user FPGAs on the BEE2 play slightly different roles. The control FPGA is connected to peripherals, as well as to the Xilinx SelectMAP interface on the user FPGAs, giving it the ability to configure and read the status of the user FPGAs. The SelectMAP interface can then be reused as FIFO-based GPIO between the control and user FPGAs. One of the PowerPC cores in the control FPGA runs Debian GNU/Linux, allowing users to interact with the board and use existing software and drivers.

2.3. Off Board Communication

The BEE2 was designed to operate in clusters by using the high-speed serial links formed by bonding four of the Xilinx Multi-Gigabit Transceivers (MGT) to form a 10 Gbps full-duplex communication link. XAUI, a standard layer-1 point-to-point protocol, was chosen as the underlying protocol because it provides an upgrade path to 10 Gb Ethernet. Our XAUI block guarantees that no data will be lost on the link, except when the link state changes, *e.g.*, a cable is unplugged. Error correction is left to the user logic. In lab tests, the error rate over a stable link was found to be 10^{-16} to 10^{-10} bps.

3. RAMP BLUE DESIGN AND ARCHITECTURE

RAMP Blue is a message-passing multi-processor built from 768–1008 soft cores instantiated on a cluster of 16–21 BEE2 boards. RAMP Blue is intended to demonstrate the first large multi-board system using BEE2 boards and to experiment with the limits of soft-core designs.

3.1. High-Level Design

The primary design choices for this project included the soft CPU core, I/O interfaces, network topology, FPGA platform, memory allocation, operating system, and tool set. The remainder of this section will examine each of these choices.

3.1.1. Processor and I/O Interfaces Selection

The processor selection was significantly eased by the lack of feasible options, with the Xilinx MicroBlaze [4] being the only viable soft core for this project. MicroBlaze is a RISC processor optimized for implementation in Xilinx FPGAs and having a basic 32-bit ISA, a three-stage pipeline, a 32-word register file, and a Harvard-style direct-mapped L1 cache with configurable size. The core can be customized to include a single-precision FPU that shares the existing register file, a hardware multiplier, divider and barrel shifter, several CISC instructions, and several exceptions.

MicroBlaze also has considerable software support available, including a GCC backend and a port of uClinux [5]. Access to this infrastructure was essential to meeting the goal of running off-the-shelf code and benchmarks.

The MicroBlaze supports several interfaces including the IBM CoreConnect On-chip Peripheral Bus (OPB), Xilinx Local Memory Bus (LMB), Xilinx CacheLink (XCL), and Xilinx Fast Simplex Link (FSL). The FSL is a simple FIFO-like interface that provides unidirectional point-to-point connections. In our design, the FSL is used for the majority of the peripherals, though the LMB and the OPB are required for the dedicated block RAM, and the interrupt controller and timer, respectively. The use of the FSL was motivated by its point-to-point nature, which helped fit the system into the RAMP Description Language (RDL) [6] for research and debugging purposes.

3.1.2. FPGA Allocation

Although the BEE2 has five FPGAs per board, the control FPGA is already responsible for running Linux, configur-

ing the user FPGAs and providing debugging and monitoring support. The gateware required does not use all the resources of the control FPGA; nonetheless, we limited the soft cores to the user FPGAs in order to save on bitstream-generation time. The place-and-route cycle on a highly utilized Virtex-II Pro 70 design can take 3–30 hours on the fastest available machines. Maintaining separate bitstreams would greatly increase the implementation and testing times.

3.1.3. Cluster Topology Selection

Board-to-board connections use copper 10GBASE-CX4 cables, which provide full-duplex 10 Gbps links between FPGAs. Despite the high bandwidth, the latency of these links (tens to hundreds of cycles) is large compared to that of intra-board LVCMOS links (two to three cycles).

For 16 BEE2 modules, we use an all-to-all topology, with each module having one high-speed serial connection to each other module as shown in Figure 2 (a). This topology minimizes the number of inter-board links along any communication path, optimizing latency and reliability, with the path from one soft core to another containing at most four intra-board (two on each board) and one inter-board link. The FPGA and port assignments are rotated in order to use only vertical cables. Unfortunately, this topology does not scale beyond 17 modules without the control FPGA due to port limitations.

For 18–21 modules, we use a 3D-mesh topology, shown in Figure 2 (b). The FPGA and port assignments are again rotated in order to fit within the constraints of the rack and use only vertical cables.

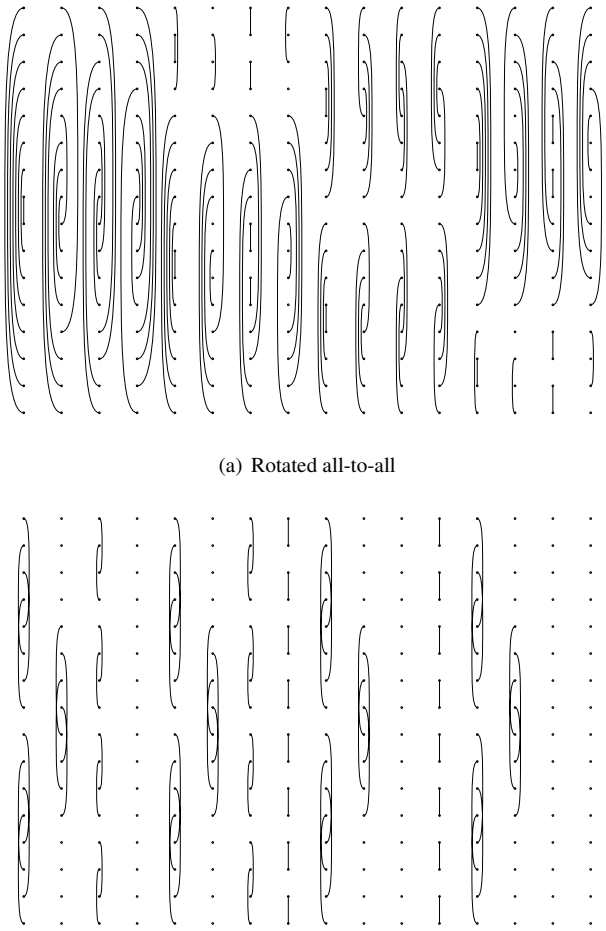
3.1.4. Memory Allocation

Since RAMP Blue is a message-passing architecture, a simple static partitioning of the available physical memory is sufficient. We currently instantiate 12 MicroBlaze cores per user FPGA using three of the four DDR2 DIMMs, resulting in four cores per 1 GB DIMM, each using a quarter of the memory—256 MB per core.

Four processor cores share a DRAM channel through a common bus and controller as shown in Figure 3. Although the high degree of memory-channel sharing in conjunction with small L1 caches can lead to inefficient DRAM usage, the banked design of DDR2 chips ensures that each core has its own memory bank, dramatically reducing memory-access conflicts. Furthermore, the low speed of the MicroBlaze cores relative to the 180 MHz DDR2 memory helps eliminate memory contention.

3.1.5. Operating System, Compiler and Applications

Each MicroBlaze core runs its own instance of uClinux. The choice of uClinux and GCC is natural given the goal of



(a) Rotated all-to-all

(b) Folded 3D mesh

Fig. 2. Possible cluster topologies. Rows and columns correspond to boards and ports, respectively.

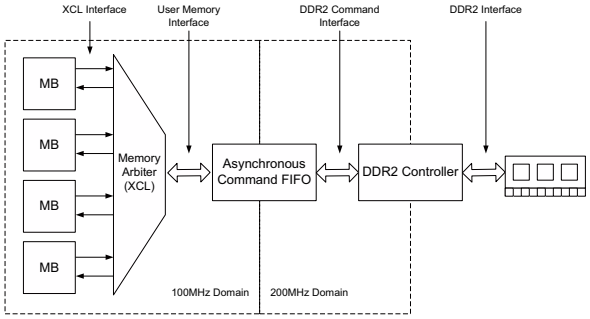


Fig. 3. Memory-system access from MicroBlaze cores.

running off-the-shelf applications. uClinux is a fork of the Linux kernel designed for microcontrollers without MMUs, a basic requirement of the Linux kernel. As a result, uClinux does not provide virtual memory or protection. However, almost all of the kernel support for device drivers, networking and file systems is available under uClinux. The POSIX API is fully supported with the exception of the *fork* system call. An extensive user application and library distribution is also available. Both the kernel and the user applications build using the GCC MicroBlaze backend provided by the open-source community.

To meet our goal of running scientific benchmarks, we selected the NAS Parallel Benchmarks (NPB) suite [7] implemented in the Unified Parallel C (UPC) framework [8]. We chose the UPC version, rather than the the more common MPI version, because UPC is both higher-level and higher-performance than MPI in most cases. Additionally, our UPC implementation was developed at UC Berkeley, making the authors available for consultation.

3.2. Architecture and Memory

Figure 4 shows the top-level gateware architecture of a single MicroBlaze node, around which RAMP Blue was designed.

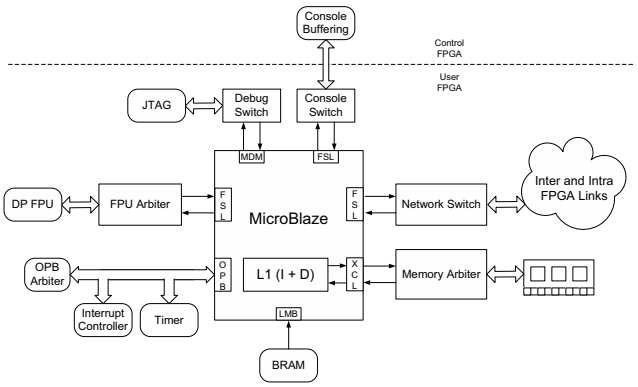


Fig. 4. Architecture of single node of RAMP Blue.

The RAMP Blue memory system was designed to reuse existing BEE2 infrastructure described in Section 2 while sharing the DIMMs between cores as shown in Figure 3.

Low-level access is provided by the existing DDR2 controller and asynchronous command FIFO, while high-level access is provided by a parametrized memory arbiter supporting the XCL cache transaction protocol. A set of tag FIFOs alongside the memory datapath allow read responses to be matched to the requesting processor cores.

The memory is divided by address, with the bank portion of the address statically parametrized by the core index, allowing each core to execute exactly the same code without interference. Since the XCL interface cannot be used when caches are disabled, each core is also connected using an LMB to a block RAM containing the boot and cache invalidation code that executes with caches disabled.

3.3. Console and Control Network

Each core must receive its code, data and user input over a channel accessible through Linux on the control FPGA. In our implementation, the physical layer for this channel can be either the SelectMAP data bus or the LVCMOS inter-chip link, with RAMP Blue using the latter due to much higher bandwidth.

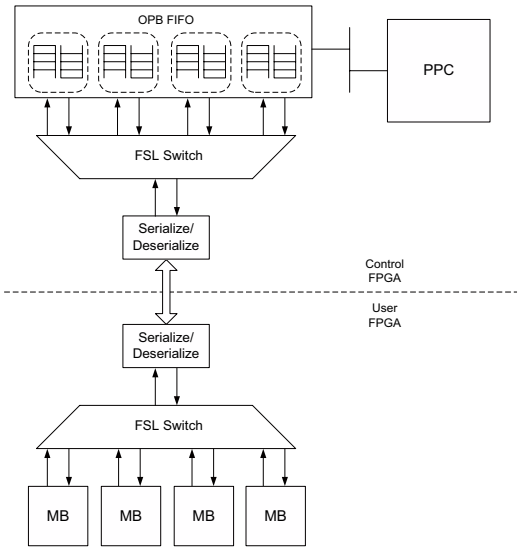


Fig. 5. Architecture of general-purpose control channel.

Figure 5 shows the gateware comprising the control channel. On the control FPGA, the OPB FIFO block provides a link between the PowerPC and a configurable number of FIFOs using a memory-mapped interface. On the other side of this block, the FIFOs export an FSL interface, allowing them to be connected to MicroBlaze cores through matching FSL switches and serializer/deserializer (SERDES) blocks on either side of the LVCMOS link. The SERDES blocks convert the data to and from a configurable width and transfer it across the inter-chip link using conservative signaling, while the switches ensure delivery to and from the appropriate port.

There are two abstractions on top of the control channel: a serial console allowing the control FPGA to monitor boot messages and a network interface providing standard network services, such as NFS and TELNET.

3.4. Debugging Interface

Each Xilinx FPGA has an internal JTAG chain, which can be accessed via the FPGA fabric to perform in-circuit debugging. Two forms of hardware debugging are implemented in

RAMP Blue, both of which use the JTAG port, making them mutually exclusive.

The first form uses the Xilinx Microprocessor Debug Module (MDM) and the Xilinx Microprocessor Debugger (XMD), which allow the user to access the MicroBlaze registers and memory and set breakpoints, as well as to connect with GDB to perform full remote debugging.

The second form uses the Xilinx ChipScope in-circuit logic analyzer to capture traces of arbitrary signals. Additional trace buffers interface between each MicroBlaze and the corresponding ChipScope unit, capturing traces of events occurring prior to connecting with ChipScope.

3.5. MicroBlaze Network

In RAMP Blue, messages are passed along a custom network composed of intra-FPGA, intra-BEE2 and inter-BEE2 links. The high-level design choices listed below were made to simplify the network design and implementation.

- **Routing:** Packets are statically (non-adaptively) source routed at each hop in the network, with broadcast not supported.
- **Topology:** Chip-level crossbar embedded in a board-level mesh embedded in a system-level all-to-all graph or chip-level crossbar embedded in a system-level 3D mesh.
- **Packet Format:** The gateware is oblivious to packet format except for the route header, added by the source and stripped by the packet buffers. For compatibility, the packet format is Ethernet II encapsulated in a small amount of control information.
- **Delivery Guarantees:** Delivery is guaranteed end-to-end in software. The gateware does not perform checksumming or retransmission.
- **Flow Control:** Virtual cut-through, blocking only if the next buffer is not available.

3.5.1. Network Gateware

The switch is designed as the composition of two simple elements: the buffer unit and the crossbar switch. Buffer units store a single packet at each hop in the network and make requests to the switch for the next hop. Each buffer unit uses a single block RAM to provide buffering for an MTU of up to 2048 bytes and exposes FSL read and write interfaces with additional signals used for arbitration. Packets are marked with start and end control bits, allowing for error tolerance.

The crossbar switch unit is fully parametrized in the number of ports, and data width and latency. Changing the data width trades performance for resource usage, allowing for overall performance optimization or acceleration of the place-and-route cycle. For each output buffer unit, the switch arbitrates among the inputs requesting that output using a starvation-free round-robin policy. Once an input has won the arbitration, it transmits the entire packet.

3.5.2. MicroBlaze Interface

For each MicroBlaze core, the network exposes an FSL interface. Packets are transmitted by first testing whether the buffer unit is available with a non-blocking FSL write and then copying the remainder of the packet without blocking. When a packet is available for receiving, the buffer generates an interrupt, instructing the MicroBlaze to copy the packet using a series of FSL reads. This interface is both simple and low-performance. A future design using a small DMA engine should greatly improve the performance of the entire RAMP Blue system.

The network driver is almost exactly the same as the driver used for the control-network link. The driver provides an Ethernet-device abstraction to the operating system, prepending the source route before transmission. The source routes are computed in a distributed manner and sent to the driver using custom *ioctl* system calls by a parametrized network-setup program that runs on each node at boot time. The routes are thus static and not adaptive to total link failure, such as a cable being unplugged.

3.5.3. FPGA to FPGA Communication

As Figure 6 shows, intra-BEE2 and inter-BEE2 communication reuses the buffer and switch units. For each off-FPGA link, there is a pair of buffer units that act as a proxy for that link, interacting directly across the link without an intermediate switch. These buffer units interface to the switch on their FPGA, allowing incoming packets to be routed directly to a MicroBlaze on that FPGA or across another link.

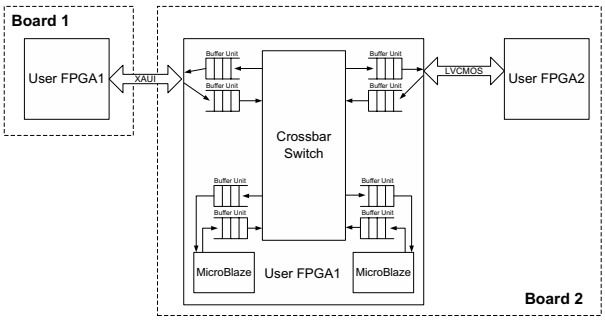


Fig. 6. Example communication using multiple hops through multiple FPGAs.

3.5.4. Reliability and Deadlock

To simplify the implementation, the amount of error checking in the network is sufficient only to guarantee the proper operation of the network itself. Bit errors in the route header

can result in misrouted packets. We rely on deserialization and next-hop checking to discard such packets in the network and destination checking to discard such packets at the endpoint. Corrupted start or end bits can also result in discarded packets. The low bit error rates observed in the lab suggest that these should be relatively rare occurrences.

Multi-processor networks admit the possibility of deadlock, which occurs when all buffers in a cycle become full. With virtual cut-through, a common solution is to partition the buffers and place a partial ordering on them [9]. Another popular method is to use virtual channels [10]. In RAMP blue, the all-to-all topology requires a combination of these methods to avoid deadlock in all cases. A partial ordering on the buffers is enforced using dimensional routing, with sufficient dimensions provided by virtual channels. The cost of this implementation is two additional receive buffers on the intra-board links out of a total of 38 buffers and a corresponding increase in the crossbar-switch ports.

3.6. Floating Point Unit

Most of the off-the-shelf applications and scientific benchmarks that we were interested in running on RAMP Blue require double-precision floating-point arithmetic. MicroBlaze does not provide a double-precision FPU, and even the integrated single-precision FPU is too large (1000 slices) to be instantiated once per core, leading us to a *shared* double-precision FPU implementation.

3.6.1. Floating Point Unit Architecture

To avoid the complexity of implementing and verifying an IEEE-compliant double-precision FPU, the actual arithmetic core uses the Xilinx LogiCORE floating-point blocks for addition, multiplication, division, and comparison [11].

The FPU arbitrates among a parametrizable number of FSL-based MicroBlaze interfaces. Operations are requested as four 32-bit FSL writes, with the control bits encoding the operation. A deserializer assembles the two 64-bit operands and requests the correct functional unit. When access to the input bus is granted, the operands are transferred into the functional unit. When the operation is complete, and access to the output bus is granted, the result is transferred to the serializer. Two 32-bit FSL reads by the MicroBlaze then return the 64-bit result. Exceptions are returned as silent NAN values. Figure 7 shows an overview of this architecture.

The shared FPU also makes sense for reasons other than resource usage. Given a heavily pipelined FPU and a lightly pipelined blocking MicroBlaze, providing each core with its own FPU would vastly underutilize the FPU pipelines. On the other hand, multiple MicroBlaze cores can take advantage of the FPU pipelining using multi-threading, achieving high overall utilization of a shared FPU.

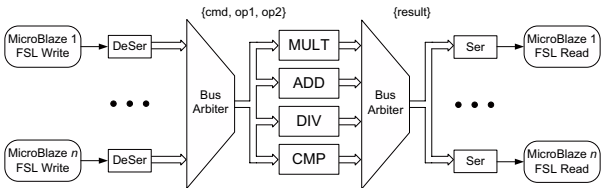


Fig. 7. Shared double-precision FPU architecture.

3.6.2. Compiler Integration

Support for the shared FPU was added to GCC using the software floating-point library interface. Whenever an FPU operation is required, the compiler emits code to send the operands to the FPU and then block until the result comes back. Blocking is not optimal and does not take advantage of the available pipelining. However, since the FPU is highly shared, the added software complexity required to manage non-blocking execution of operations was deemed unlikely to greatly improve overall system performance.

4. RAMP BLUE IMPLEMENTATION

RAMP Blue currently comprises 16–21 BEE2 boards, each with 12 MicroBlaze cores per user FPGA. Thus, 768–1008 cores can be connected in a cluster and can run the NAS Parallel Benchmarks using UPC. Several key subsystems were benchmarked independently. As the system is still in a state of development, these numbers should be considered a lower bound on performance. Furthermore, since the primary goal of RAMP Blue is emulation, raw performance matters only insofar as it affects emulation performance. High-performance architectures can be accurately emulated by accounting for the difference between the raw and the emulated clock cycles as described in [6].

4.1. Subsystem Performance

The synthetic Whetstone benchmark showed an FPU performance of 2.5 to 3 MFLOPS, corresponding to an overall system performance of 2–3 GFLOPS for 768–1008 MicroBlaze cores. Although Whetstone is not a realistic benchmark, it is useful in noting the difference in performance between the software-emulated and the hardware-accelerated FPUs. In this test, it showed a speedup of 15 for scores of 200 KFLOPS and 3 MFLOPS, respectively.

We used the Netperf benchmark [12] to determine the performance of the MicroBlaze-to-MicroBlaze network for user code running under uClinux. Netperf tests bulk-transfer throughput and round-trip response time using both TCP and UDP. There is no significant variation in either throughput or response time for the different link combinations for single communicating pairs. On the other hand, as the payload size grows, there is a clear increase in round-trip re-

sponse time. This result suggest that the software overhead of transmitting and receiving is dominating the effects of latency in different links, even with the inter-board links having a much greater latency than the intra-board links do. Thus, there is an urgent need for a DMA-based network interface and driver.

4.2. Cluster Connectivity

Each BEE2 has a 100 Mb Ethernet connection to a control-network switch. This switch is also connected to a management server, which stores the NFS file systems for both the PowerPC and the MicroBlaze cores. Each PowerPC and MicroBlaze core has an IP address on the control network and can be accessed via SSH and TELNET, respectively. Packets are routed to and from the MicroBlaze cores by the PowerPC cores. Additionally, all point-to-point links in the system are instrumented using iptables rules that sample traffic and forward it through IP-in-IP tunnels over the control network. This traffic aggregates on the management server and can be viewed in real-time using a modified version of the EtherApe graphical network-traffic analyzer. The whole cluster is located behind a NAT firewall and can be accessed from the Internet.

Debugging is performed by connecting a Xilinx Parallel Cable IV between the management server and the JTAG header on a BEE2 module. Up to four modules can be debugged simultaneously using either XMD or ChipScope, given the parallel-port hardware and software limitations. Serial console and monitor output from any one module are also available. More scalable management capabilities are currently being developed.

4.3. FPGA Implementation

A full 12-core RAMP Blue design consumes 32,991 slices (99%), 61,891 LUTs (93%), 37,198 flip-flops (56%), and 181 block RAMs (55%). These results assume the following processor options: 90 MHz core clock, no optional functional units, all optional exceptions, 2 KB I-Cache, 8 KB D-Cache, LMB block RAM, and OPB peripherals. The infrastructure consists of three DDR2 controllers, four XAUI blocks, double-precision FPU, and 8-bit network buffers and crossbar switch. Based on these results, we believe that a configuration with 16 MicroBlaze cores and four DDR2 controllers is feasible with extensive optimizations.

4.4. Operating System, Software and Benchmarks

In RAMP Blue, uClinux, traditionally used in embedded systems, must support software designed for large high-performance computers. Fortunately, since uClinux maintains almost all of the functionality of a traditional Linux kernel, most software that runs on a normal Linux system also runs on a uClinux system, given proper compiler and libraries.

4.4.1. UPC and GASNet

The most complex part of porting the UPC framework to a new platform is ensuring that its communication layer, Global-Address-Space Networking (GASNet) [13], works properly with both the kernel and the underlying communication hardware. GASNet provides message-passing conduits implemented over specific networking hardware or generic UDP sockets, with RAMP Blue using the latter mechanism for simplicity. Future RAMP systems should gain a significant amount of performance by implementing custom Remote DMA (RDMA) network hardware and a corresponding GASNet conduit. Such an implementation will decouple computation and communication and eliminate the overhead associated with the UDP/IP stack and uClinux networking.

4.4.2. NAS Parallel Benchmarks

The NAS Parallel Benchmarks are a standard benchmark suite consisting of scientific code useful for measuring the performance of message-passing systems. Due to memory limitations, only class-S (sample) datasets can currently be run on RAMP Blue. The next larger class uses more memory than is available, given the uClinux, UPC and NPB memory-allocation details. The implementation used is based on NPB 2.4, which is not optimized for a particular architecture [7].

The UPC implementation of NPB 2.4 consists of six benchmarks: Embarrassingly Parallel (EP), Multi-grid (MG), Conjugate Gradient (CG), Fast Fourier Transform (FT), Integer Sort (IS), and Block-Tridiagonal with I/O (BTIO). We are currently able to run all of these except BTIO, with various restrictions on the number of threads due to benchmark implementation details.

The execution time is currently dominated by communication costs, a large fraction of which can be eliminated by implementing a DMA-based network interface. Therefore, the current results should be considered a lower bound on performance. Additionally, the primary goal of RAMP—to provide a flexible platform for research in parallel architectures and software—can be met using even medium-performance hardware.

4.5. Bugs and Implementation Hurdles

In the course of implementing the RAMP Blue system, several bugs and implementation hurdles came up, most significantly due to the adoption of MicroBlaze. Several bugs were found and fixed in both the MicroBlaze gateware and the GCC and uClinux ports. The process of isolating these

bugs was hindered by the lack of good debugging tools. Future RAMP projects seeking to run off-the-shelf software should leverage more robust compilers and operating systems or implement improved debugging capabilities.

5. CONCLUSION

The goal of RAMP Blue was to create a message-passing multi-core system, capable of running off-the-shelf applications and scientific benchmarks, on the BEE2 platform. The current implementation meets these goals. RAMP Blue is currently able to run uClinux on 768–1008 independent MicroBlaze cores on 16–21 BEE2 boards. The cores are able to run the majority of the NAS Parallel Benchmarks and compute accurate double-precision floating-point values.

The RAMP Blue project has also been useful in pushing RAMP development and providing the following interesting lessons.

- Analysis of the performance and resource usage of the FPU showed that sharing a single FPU is essential and unlikely to impact performance.
- Debugging infrastructure should be a first-class primitive. Debugging RAMP Blue—a large and complicated system—using existing debugging primitives proved to be cumbersome.
- Built-in error checking and tests are essential, since variations in FPGA designs can expose bugs that occur only with certain bitstreams and on certain boards.
- Gateware should be reusable, general-purpose and optimized for place-and-route, since the design-compile-debug cycle for FPGAs is long.

Work is currently under way to implement various performance enhancements suggested throughout this paper. A parallel project has ported the entire RAMP Blue design to RDL in order to separate the emulation of a multi-core architecture from the underlying FPGA implementation, as well as improve parametrization and debugging support. A beta release of this design is forthcoming.

6. ACKNOWLEDGMENTS

We would like to thank Dave Patterson, Krste Asanovic, the other RAMP PIs, as well as Jue Sun, Dan Bonachea, and the students of UC Berkeley CS 252 Spring 2006, who have all contributed to the design and implementation of RAMP Blue. We would also like to thank Chen Chang, Henry Chen, and Dan Burke for all their work in creating the BEE2.

This work was funded in part by the National Science Foundation Grant No. CNS-0551739. The authors acknowledge the support of the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program. The authors also acknowledge the contributions of the students, faculty and sponsors of the Berkeley Wireless Research Center and the National Science Foundation Infrastructure Grant No. 0403427. Special thanks go to Xilinx for their continuing financial support and donation of FPGAs and development tools.

7. REFERENCES

[1] J. Wawrzynek, D. Patterson, M. Oskin, S. Lu, C. Kozyrakis, J. Hoe, D. Chiou, and K. Asanovic, "RAMP: Research Accelerator for Multiple Processors," *IEEE Micro*, vol. 27, no. 2, 2007.

[2] A. Schultz, "RAMP Blue: Design and Implementation of a Message Passing Multi-processor System on the BEE2," Master's thesis, University of California, Berkeley, 2006.

[3] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: A High-End Reconfigurable Computing System," *IEEE Design and Test of Computers*, vol. 22, no. 2, 2005.

[4] Xilinx, *MicroBlaze Processor Reference Guide*, http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf.

[5] J. Williams, "MicroBlaze uClinux Project Home Page," http://www.itee.uq.edu.au/~jwilliams/mblaze-uclinux/.

[6] G. Gibeling, A. Schultz, J. Wawrzynek, and K. Asanovic, "The RAMP Architecture, Language and Compiler," University of California, Berkeley, Tech. Rep., 2007. [Online]. Available: http://ramp.eecs.berkeley.edu/

[7] R. van der Wijngaart, "NAS Parallel Benchmarks Version 2.4," National Aeronautics and Space Administration, Tech. Rep. NAS-02-007, 2002. [Online]. Available: http://www.nas.nasa.gove/News/Techreports/2002/PDF/NAS-02-007.pdf

[8] W. Carlson *et al.*, *Introduction to UPC and Language Specification*. Center for Computing Sciences, Institute for Defense Analyses, 1999.

[9] K. Gunther, "Prevention of Deadlocks in Packet-Switched Data Transport Systems," *Communications, IEEE Transactions on [legacy, pre-1988]*, vol. 29, no. 4, pp. 512–524, 1981.

[10] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547–553, 1987.

[11] Xilinx, *LogiCORE Floating-Point Operator v1.0*, http://www.xilinx.com/bvdocs/ipcenter/data_sheet/floating_point.pdf.

[12] R. A. Jones, "Netperf: A Network Performance Benchmark (Revision 2.0)," Hewlett-Packard Company, Tech. Rep., 1995.

[13] D. Bonachea, "GASNet Specification, v1.1," Lawrence Berkeley National Laboratory, Tech. Rep. CSD-02-1207, 2002. [Online]. Available: http://upc.lbl.gov/publications/CSD-02-1207.pdf