Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2012, Article ID 564704, 15 pages
doi:10.1155/2012/564704

Research Article

An Evaluation of an Integrated On-Chip/Off-Chip Network for High-Performance Reconfigurable Computing

Andrew G. Schmidt,[1] William V. Kritikos,[2] Shanyuan Gao,[2] and Ron Sass[2]

[1] *Information Sciences Institute, University of Southern California, 3811 North Fairfax Drive, Suite 200, Arlington, VA 22203, USA*
[2] *Reconfigurable Computing Systems Lab, UNC Charlotte, Electrical and Computer Engineering Department, 9201 University City Boulevard, Charlotte, NC 28223, USA*

Correspondence should be addressed to Andrew G. Schmidt, andrewgschmidt@gmail.com

Received 28 September 2011; Accepted 18 February 2012

Academic Editor: Esam El-Araby

As the number of cores per discrete integrated circuit (IC) device grows, the importance of the network on chip (NoC) increases. However, the body of research in this area has focused on discrete IC devices alone which may or may not serve the high-performance computing community which needs to assemble many of these devices into very large scale, parallel computing machines. This paper describes an integrated on-chip/off-chip network that has been implemented on an all-FPGA computing cluster. The system supports MPI-style point-to-point messages, collectives, and other novel communication. Results include the resource utilization and performance (in latency and bandwidth).

1. Introduction

In 2007 the *Spirit* cluster was constructed. It consists of 64 FPGAs (no discrete microprocessors) connected in a 3D torus. Although the first integrated on-chip/off-chip network for this machine was presented in 2009 [1], the design has evolved significantly. Adjustments to the router and shifts to standard interfaces appeared as additional applications were developed. This paper describes the current implementation and the experience leading up to the present design. Since the network has been implemented in the FPGA's programmable logic, all of the data presented has been directly measured; that is, this is not a simulation nor emulation of an integrated on-chip/off-chip network.

A fundamental question when this project began was whether the network performance would continue to scale as the number of nodes increased. In particular, there were three concerns. First, would the relatively slow embedded processor cores limit the effective transmission speed of individual links? Second, were there enough resources? (Other research has focused on mesh-connected networks rather than crossbars due to limited resources [2–5].) Third, would the on-chip and off-chip network bandwidths be balanced so one does not limit the other? Although some of the data presented here has appeared in publications related to different aspects of the project, the aim of this paper is to provide a comprehensive evaluation of the on-chip/off-chip network. The results are overwhelmingly positive, supporting the hypothesis that the current design is scalable.

The rest of this paper is organized as follows. In the next section some related work is presented for on-chip networks. In Section 3 we describe the reconfigurable computing cluster project and the small-scale cluster, *Spirit*. Following that, in Section 4, the specifics of the present on-chip/off-chip network design are detailed. The next three sections describe the performance of the network under different scenarios. Specifically, Section 5 shows how MPI collective communication cores implemented in hardware can be interfaced directly to the network to improve the performance of message-passing applications in software. Section 6 demonstrates application-specific hardware cores interfacing to the network. In Section 7, a novel performance monitoring system is presented that uses the network with

minimal invasion to the processor. The paper concludes with Section 8 where a summary of the results thus far and future work are described.

2. Related Work

The idea of using a network on chip to replace global wiring was proposed by Dally and Towles [6] in 2001. That paper argued that the amount of resources required to implement the network (they estimated 6.6% of the chip) was small relative to the benefits which include opportunities for optimization and a universal interface (to promote reuse). Since then, most of the research in this area has taken a VLSI focus [7], looking at issues such as power, speed, and flow control [8, 9].

Nearly all of the academic research has focused on a 2D mesh or torus topology. This is a natural extension of the medium—routers are easily connected by wires allocated in the upper two metal layers. However, in industry, the network of choice is currently the complete graph (K_n where n is the number of cores); that is, all cores are directly connected to all other cores. Intel's Quick Path Interconnect exemplifies this approach. Industry acknowledges that this is an interim solution, but the public roadmap does not state what will replace it.

Although the body of academic research generally does not discuss interfacing the on-chip network to an external network, one can anticipate the current path. Increasingly, chip manufacturers are incorporating new cores, such as graphic processors, memory controllers, and network interfaces. VLSI researchers generally see the NoC as connecting modules of the chip. The natural eventuality would be to connect the network interface to the NoC as just another module. What sets the network presented here apart from this approach are two factors. (1) Unrelated traffic can cause contention that blocks messages destined for the network interface leading to jitter. (2) Multiple messages destined for off-chip locations have to be serialized. In summary, these related works rely on simulations with built-in assumptions about general-purpose traffic patterns, but HPC traffic has a bias towards the network interface.

3. Reconfigurable Computing Cluster

The reconfigurable computing cluster (RCC) project [10] is investigating the role (if any) field programmable gate arrays (FPGAs) have in the next generation of very-large-scale parallel computing systems. Faced with growing needs of computational science and the existing technology trends, the fundamental question is *how to build cost-effective high-performance computers?* To answer this question, the RCC project identified four research areas to investigate (which recently have gained attention [11]), namely, memory bandwidth, on-chip and off-chip networking, programmability, and power consumption. While the focus here is on the networking, the three remaining categories clearly play a significant role and, as a result, will also be present in the discussion.

There are numerous aspects to FPGAs that make them especially useful for high-performance computing. One advantage is that, because a single FPGA is now large enough (and has a rich enough set of on-chip IP), it is able to host an entire system on chip. This eliminates many of the peripheral components found on a standard commodity PC board which in turn offers size, weight, and power advantages. Also, by reducing the size of the node, there is a secondary advantage in that it reduces the distance between nodes enabling novel networking options. Research also indicates that FPGA floating-point performance will continue to rise (both in absolute terms and relative to single-core processors [12]).

Certainly, there are potential pitfalls that could limit the expected performance gains of such an approach. Until recently, FPGAs have had the reputation of being slow, power hungry, and not very good for floating-point applications [13]. However, these generalizations are based on a snap shot in time and depend heavily on the context in which the FPGA is employed. Moreover, while a number of HPC projects are using (have used) FPGAs [14–22], they have been used as "compute accelerators" for standard microprocessors. The RCC project's proposed approach is fundamentally different in that the FPGAs are promoted to first class computation units that operate as peers. The central hypothesis of the project is that the proposed approach will lead to a more cost-effective HPC system than commodity clusters in the high-end computing range.

The RCC's initial investigation began in early 2007 with the assembly of *Spirit*, a small-scale cluster of 64 all-FPGA compute nodes (with no discrete microprocessors in the design), arranged in a 4-ary 3-cube network topology. Each compute node consists of a Xilinx ML410 development board [23] with a Virtex-4 FX60 FPGA, 512 MB of DDR2 memory, gigabit ethernet this time only and a custom high speed network (among several other peripherals). Figure 1 depicts a simple representation of the organization of the cluster. The server node is a commodity x86 server used to manage the cluster and provide a network filesystem for the cluster. The server node connects to the cluster through a commodity Gigabit Ethernet switch to which each FPGA node also connect.

Each node receives an application-specific configuration bitstream over the Ethernet network through a custom application called FPGA session control (FSC) [24]. (This network is just used for administrative traffic, such as rsh/ssh, ntp, NFS, etc.) FSC enables collaborates to access the Spirit cluster remotely and upload ACE files to each node's CompactFlash. The user can then select a specific ACE file to boot from, which in turn configures the FPGA with the intended design. Further details regarding the network will be discussed in Section 4.

4. Spirit's Integrated On-Chip and Off-Chip Network

At the heart of the spirit cluster is the architecture independent reconfigurable network (AIREN) which is

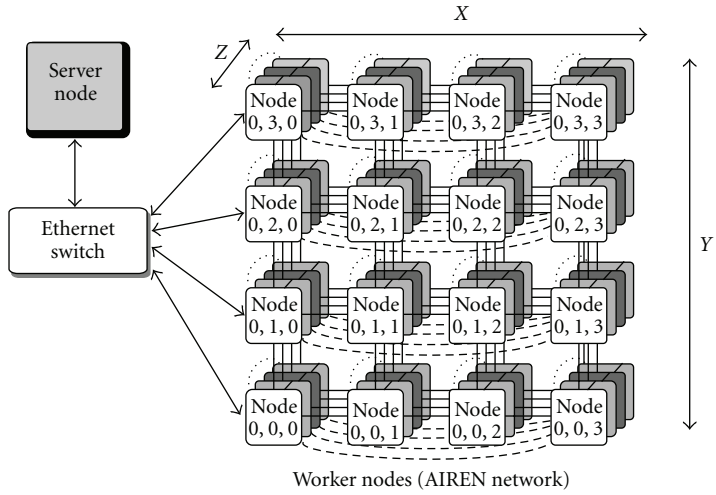

Figure 1: The RCC's Spirit cluster consists of 64 FPGAs connected in a 4-ary 3-cube through a high-speed network (AIREN).

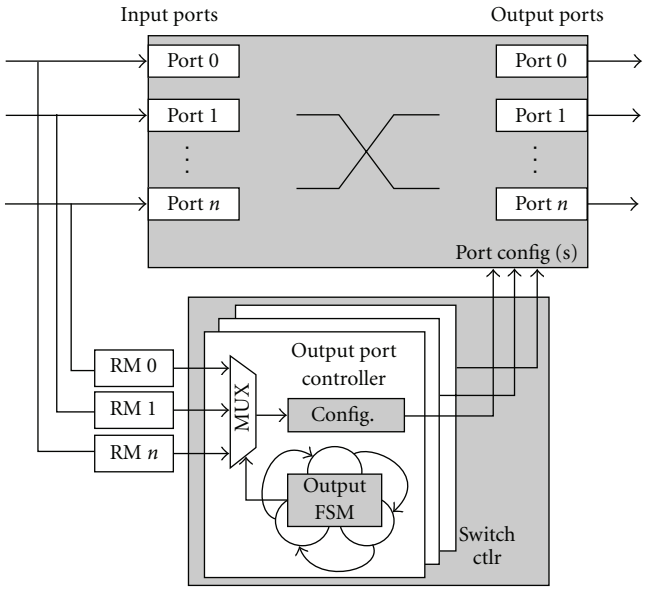

Figure 2: Block diagram of the AIREN router.

an integrated on-chip/off-chip network that not only supports efficient core-to-core communication but node-to-node as well. Figure 14 shows a high-level overview of the FPGA's system on chip. On the FPGA, AIREN spans several IP cores, of which the AIREN router is the most central component. The original implementation of AIREN has been published in a short paper in 2009 [1]; however, significant architecture improvements have been made which warrant further discussion.

AIREN offers several configurability options in order to provide the desired network behavior. These include the ability to change the number of input and output ports (radix) of the switch, the data width, operating frequency, whether the switch has software controlled routing or hardware accelerated routing, the dimensionality of the off-chip network (ring, mesh, cube, etc.), and the routing methodology. These details and more regarding the AIREN network will be provided within this section.

4.1. AIREN Router. Figure 2 details the internal components and functionality of the AIREN router. The router consists of a single crossbar switch implemented in the FPGA's programmable logic along with routing decision modules that inspects the header of each packet and passes the routing decision to the switch controller. The switch controller manages the connections and deals with contention for output ports. The crossbar switch is implemented as a generic VHDL model to allow easy customization of the number of ports needed for a given implementation. The signals that are switched include 32 bits of data along with 4 bits of flow control signals. On-chip cores connect to the switch for a high-bandwidth, low-latency communication. Off-chip cores communicate through a set of special ports which interface to the FPGAs multi-gigabit transceivers for communication with other FPGAs.

4.2. AIREN Interface. One of the first architectural enhancements to the AIREN router is changing from the use of an in-house interface (AIREN network interface) to connect compute cores to the network. A consequence of that interface was less than optimal latency of 0.08 μs across the switch. More recently, work has been done to migrate to a more widely used and accepted interface standard supported by Xilinx known as LocalLink [25]. With the adoption of this standard, compute cores can be quickly connected to the network. In addition, LocalLink provides flow control which reduces the need to incorporate buffers throughout the network, freeing up scarce on-chip memory resources (BRAM). Furthermore, LocalLink supports frames, which means headers and footers are more easily identifiable and results in a reduction of the latency across the switch to 0.02 μs. Figure 3 provides a simple diagram of how the

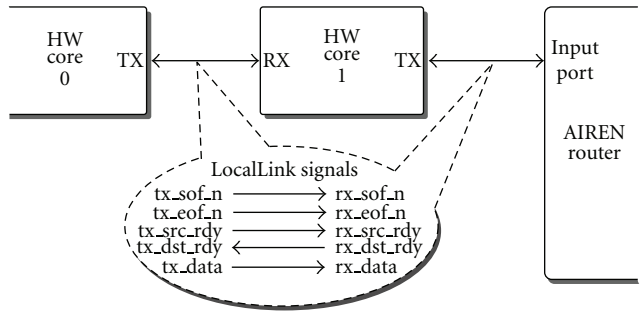

Figure 3: Simple illustration to show the Xilinx LocalLink standard implemented within the AIREN network to support both core-to-core and core-to-router connectivity.

LocalLink standard is incorporated into the network and specifically can allow compute cores to be chained together and/or connected to the AIREN router. LocalLink is used in both the software- and hardware-controlled routing configurations.

4.3. AIREN Routing Module. Each input port connects to both a port on the crossbar switch and to a routing module hardware core. The routing module examines the header of each incoming packet to determine its destination. Each node receives a unique node ID, and each core has a core id that is only required to be unique on each node, a decision that was made to simplify routing to first route to a node and then route to a core. The routing module looks at the packet header and identifies the packet's destination in terms of `node ID` and `core ID`. If the packet's node ID matches the local node ID, then the core ID is used to route the packet to the corresponding core. If the IDs differ, the routing module directs the packet to the corresponding node's off-chip port, depending on the routing algorithm. In the work presented here, a simple dimension-ordered routing algorithm is used to determine which outgoing port the message is routed to [6], although the routing module is configurable to support different off-chip routing algorithms. Once a routing decision has been made, the routing module passes the routing request to the switch controller and must wait until it is granted access to the crossbar switch. When an input port is granted access to the output port, it owns that port until the packet is finished.

A packet includes within the first 8-byte header both the node and core ID, as can be seen in Figure 4. The LocalLink start-of-frame (SOF) signal marks the beginning of a packet. The routing module analyzes the header to make the routing decision; otherwise the routing module is a passive component in the system. At present no footer is used; however, for future applications to use such a feature would require no modifications in the AIREN network. Finally, when the LocalLink end-of-frame (EOF) signal is asserted, the routing module can deassert its request for the output port and the switch controller is free to reconfigure the connection to another input port.

4.4. AIREN Switch Controller. Once the routing decision has been made and the output port identified, the switch controller configures the crossbar switch to connect the input and output ports. By itself, the crossbar switch merely connects inputs to outputs with a single clock cycle of latency to register inputs to outputs. However, to control which inputs are connected to which outputs requires a switch controller. The AIREN network is configurable to support either a software controller or a hardware controller. The software controller enables the processor to set the connections within approximately 0.07μs. For systems with minimal changes to the communication path, the software-controlled switch is appealing as it offers low resource utilization. With each arriving packet, an interrupt is triggered to the processor to make the routing decision, which sets the connectivity of the input and output ports.

Of course, as the radix increases so too does the demand on the processor to make routing decisions. The switch can be controlled by hardware by connecting each input port to a routing module. Each routing module can make its own routing decision in parallel and notify the switch controller. The switch controller monitors each routing module and configures the switch based on input requests and output ports availability. Requests for separate output ports can be handled in parallel. To deal with contention, a simple priority encoder is used to give predetermined ports a higher priority. This mechanism can also be customized to support other arbitration schemes with minimal effort.

4.5. On-Chip/Off-Chip Network Integration. The architecture independent reconfigurable network (AIREN) card was designed in house to connect the nodes of the cluster. Each link in the direct connect network is a full-duplex high-speed serial line capable of transmitting/receiving at 8 Gbps (4 Gbps in/4 Gbps out). Each FPGA node in *Spirit* has eight links for a theoretical peak bandwidth of 64 Gbps to/from each node. The network was physically implemented by fabricating a custom network interface card (see Figure 5) that routes eight of the FPGA's integrated high-speed transceivers to SATA receptacles (we use SATA cables but not the SATA protocol). The AIREN network card has additional hardware used to manage the cluster, configure the FPGAs, and debug active designs.

The off-chip network is centered around the AIREN network card. Since each node can connect up to eight other nodes, a variety of network topologies are possible. The

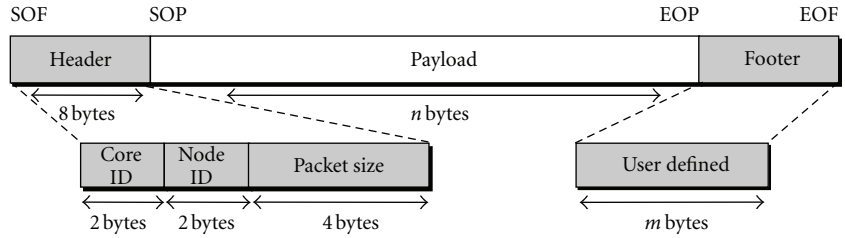

Figure 4: AIREN packet structure.

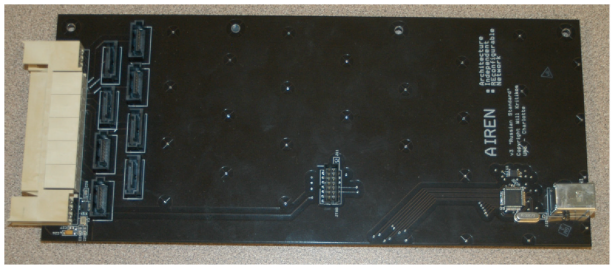

Figure 5: AIREN network card.

experiments reported here use a 4-ary 3-cube (four nodes in each of the three dimensions). Within the FPGA, each MGT is interfaced through the Xilinx Aurora protocol [26]. Aurora is a component released by Xilinx which wraps the multi-gigabit transceiver present on most of the new Xilinx FPGAs in an simple LocalLink interface. To this core the AIREN network adds custom buffers and flow control to more efficiently support back pressure between nodes. The Aurora core is configured to support bidirectional transfers of 32-bit at 100 MHz, resulting in a peak bandwidth of 4.00 Gbps. To this, Aurora uses 8B/10B encoding which further reduces the peak bandwidth to 3.2 Gbps.

4.6. AIREN Resource Utilization. There are three key parameters that can change the performance and resource requirements of our network core. First is the number of ports. This is the dominant parameter because, in general, the resources required to implement a full crossbar switch grow exponentially with the number of ports. The second parameter is the port bit width; it defines the number of bits transmitted in parallel during each clock cycle. The resources of the crossbar grow linearly with the number of bits per port. The third parameter is clock speed. Technically, all of the devices we consider have a maximum clock speed of 500 MHz; however, in practice, unless the design is hand optimized, frequencies between 100 and 200 MHz are typical. In Table 1 the resources consumed by a software-controlled and a hardware-controlled AIREN network. The purpose is to report on the scalability of the approach given the Virtex4 FX60 FPGA device resources. The key difference is the 4-input look-up table (4-LUT) utilization that is needed for the switch controller when performing hardware routing. In contrast, the software routing requires slightly more slice flip-flops (FFs) so the processor can interface with each output port configuration register.

When considering trends, it may be more instructive to increase the ports on the switch proportional to the growth in the number of logic resources. When considering different FPGA devices (different CMOS technologies), the resources needed to support a high-radix full crossbar switch on an FPGA dramatically decreases so much so that current Virtex 6 and emerging Virtex 7 devices make a full crossbar switch approach, such as AIREN, highly feasible. This is seen by fixing the switch to consume no more than 15% of the resources, as seen in Table 2. Since the behavior of the switch is very predictable and increasing the number of ports per switch will only reduce contention, this data suggest that the proposed approach is not just acceptable for the current generation of FPGA devices but also for the foreseeable technologies.

4.7. AIREN Performance. First, we test the network performance in isolation. Network layer messages between hardware and software are identical in the AIREN network. However, the ability of a core to generate or accept a message depends on the core. If a software task is the intended recipient, there is a significant amount of overhead (a processor interrupt, context switch, data, and more copies) compared to a typical hardware core (i.e., usually designed to produce or consume the message at network line speeds). There is also a speed difference if the message stays on chip versus off chip. This is because the network layer message is transmitted using a data link layer, Aurora, which handles transmission details (such as 8B/10B encoding, clock correction, etc.). Hence, we test five different combinations. The latencies are measured by performing a ping-pong test, measuring the round-trip time for a message to be transmitted from source to destination and then sent back. The round-trip time is then divided by two. The various latencies for a single hop are shown in Table 3. Using the hardware core to hardware core testing infrastructure, we looked at how the latency scaled with multiple hops as well. The results are summarized in Table 4. The results show that, as the number of hops increases, the latency per hop is decreasing. This suggests that the dimensionality of the network can be trusted to predict the overall latency of the network. This is an extremely important result because it speaks to the overall scalability of the design. (Note: all of these tests were conducted without any other jobs running on the machine; thus there was no outside contention.)

Table 1: AIREN resource utilization on V4FX60 FPGA.

No. of ports	Software routing		Hardware routing	
	No. of Slice FFs (%)	No. of 4-LUTs (%)	No. of Slice FFs (%)	No. of 4-LUTs (%)
4	433 (0.85%)	669 (1.32%)	305 (0.60%)	655 (1.30%)
8	703 (1.39%)	1,559 (3.08%)	511 (1.01%)	2,009 (3.97%)
16	1,263 (2.49%)	5,589 (11.05%)	964 (1.91%)	8,228 (16.27%)
32	2,471 (4.89%)	20,757 (41.05%)	1,933 (3.82%)	33,603 (66.46%)

Table 2: Largest 32-bit full crossbar switch using 15% of device.

Xilinx part	CMOS	Year	No. of ports
Virtex 2 Pro 30	90 nm	2002	16
Virtex 4 FX60	90 nm	2004	20
Virtex 5 FX130T	65 nm	2009	35
Virtex 6 LX240T	40 nm	2010	70
Virtex 7 855T	28 nm	2011	84

Table 3: Hardware send/receive latency through one hop.

Sender/receiver pair	Latency (μs)
hw-to-hw (on chip)	0.02
hw-to-hw (off chip)	0.80
sw-to-hw (on chip)	0.15
sw-to-hw (off chip)	0.98
sw-to-sw (off chip)	2.00

Table 4: Node-to-node latency with AIREN router.

Hops	Latency	Latency/hop
1	0.81 μs	0.81 μs
2	1.56 μs	0.78 μs
3	2.31 μs	0.77 μs
4	3.08 μs	0.77 μs

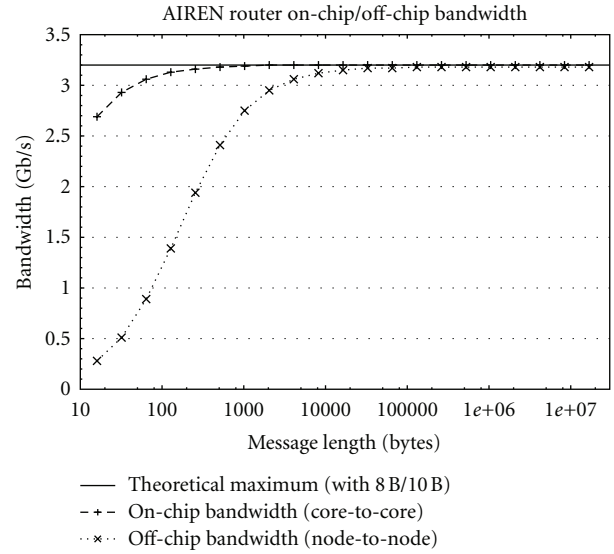

Figure 6: on-chip and off-chip bandwidth of AIREN network with hardware based routing.

To measure the bandwidth of the network, we varied the message length and repeated the ping-pong test. The measured bandwidth is plotted against the message length in Figure 6. Although the channel transmits at 4 Gbps, the data link layer performs 8B/10B encoding, a technique used to keep the sender and receiver synchronized. This, plus other communication protocol overhead, will decrease the effective data rate. Thus, after encoding, the peak theoretical bandwidth is 3.2 Gbps. Based on the data, messages on order of 10,000 to 100,000 bytes long approach optimal throughput. For on-chip bandwidth tests, messages are sent from one core to another core on the same node through the crossbar switch.

5. Accelerating Collective Communications

One of the advantages of implementing the networking cores in the FPGA fabric is the ability to rapidly introduce new hardware cores in the network data path. This allows offloading of general protocol processing [27] as well operations such as MPI collective communication operations [28]. Support for the standard MPI type collective communication tasks such as barrier, broadcast, and reduce can be added to any AIREN network implementation with the addition of a special-purpose core [29, 30]. These cores communicate among themselves using a programmable communication pattern and avoid interrupting the processor. Figure 14 shows how these cores can be connected to the AIREN router.

Specifically, the theory of operation is based on three key ideas. First, computation accelerator cores are not treated as "co-processors." Rather, they are first-class computation units that can receive and generate MPI-like messages in the system. This provides (i) a natural mechanism for processors and accelerator cores to operate in parallel and (ii) a simple performance model for computational scientists to follow. Second, the flexibility of the FPGA allows computer engineers to explore different ways of moving software functionality into hardware. For example, the active, decision-making components of the MPI collective communication operations are currently handled in the user-space MPI library. With the FPGA approach, this functionality can be moved into programmable logic as illustrated in Figure 7. Finally, by using FPGA-based compute nodes, we have the

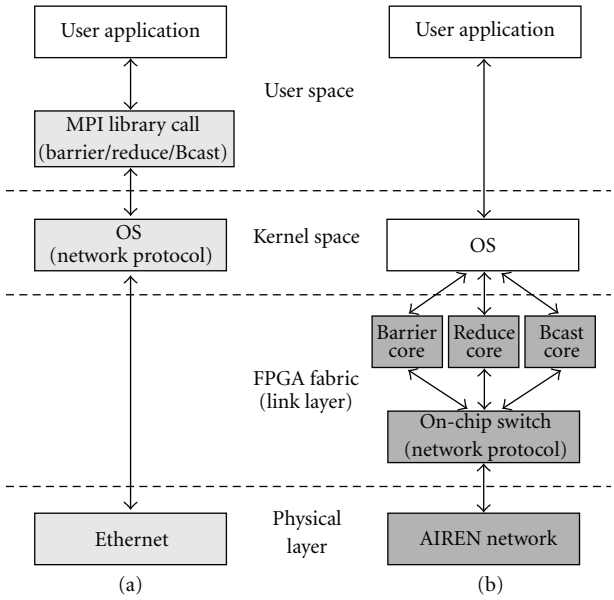

Figure 7: Example of system software refactored into hardware.

ability to tightly integrate on-chip and off-chip communication, which is the focus of this paper.

With the construction of an integrated on-chip and off-chip network operating at low latencies and high bandwidths, it became necessary to add support communication and compute accelerators. A drawback of a pure FPGA cluster is the slower clock rates of the processors (300 MHz PowerPC 405 on the Virtex4 FX60 FPGA). However, eliminating the processor entirely would drastically slow down development and reduce the chances for adoption due to the added complexities commonly associated with hardware IP development.

As a result, the RCC project chose to use the message passing interface (MPI) by replacing many of the MPI primitives with more efficient hardware-accelerated implementations. These include point-to-point operations, such as `MPI_Send` and `MPI_Recv`, as well as collective communication operations like `MPI_Barrier`, `MPI_Reduce`, and `MPI_Bcast`.

Each MPI accelerator core is connected to the AIREN network as well as the system bus. The processor is also connected to the system bus and can trigger the MPI operation through a few bus transactions needed to set up the operation. Once the MPI core is started, it performs any necessary DMA requests and communication associated with the operation.

5.1. `MPI_Send` and `MPI_Recv` in Hardware. For example, `MPI_Send` requires the processor to designate the receiving node, the base address of the data to be transmitted, and the size of the transfer. The core then issues the DMA request to off-chip memory to retrieve the data. As the data is received from off-chip memory, it is sent across the AIREN network as one or more packets to the destination. The destination receives the data and stores the data based on the corresponding `MPI_Recv` command. The tight integration between the memory controller and the network enables these transfers to occur with minimal involvement from the processor. Furthermore, the AIREN network does not require the data to be buffered and copied between the network and main memory.

Figure 8 illustrates how the hardware-based `MPI_Send` and `MPI_Recv` interact with the AIREN network. In the first step (1) the processor stores data in main memory to be sent to another node and (2) the processor initiates the DMA transfer indicating the destination node, database address, and size of the transfer (along with MPI communicator details). During the third step (3) the DMA core initiates the DMA transfer from (4) off-chip memory and (5) then generates and transfers the data to the AIREN router. The AIREN router then (6) transfers the data to the Aurora link layer core which will transfer (7) the packet to the receiving node's Aurora link layer core. Next, (8) the packet is then transferred to the destination node's router where it (9) routed to the DMA core. The DMA core interrupts (10) the destination processor, asking for a physical address (11) to store the data so that it can (12) store the data in off-chip memory. Finally, (13) the processor can use the data. The total time is dependent on the size of the transfer and the number of hops, but initial latency with respect to a single hop is shown in Table 3.

5.2. `MPI_Barrier` in Hardware. The easiest collective to implement is `MPI_Barrier`. Although many message-passing applications primarily synchronize by reasoning about the sequence of messages, there are few extremely important applications that heavily rely on barrier. For these applications, as the scale increases, the most important characteristic is reducing the time between the last task entering the barrier to when all tasks exit the barrier. Hardware is valuable here because it can typically propagate a message in a few cycles versus a software process which has a lot more overhead. There are several message-passing patterns that can be used to implement barrier (which have been explored in [29]).

Another example would be to synchronize processes or tasks executing in parallel. `MPI_Barrier` would typically be used to assure that all the tasks are operating in lockstep. A hardware-accelerated version of this core can perform all of the internal node-to-node communication without requiring any intervention from the processor, dramatically reducing the time for the barrier to complete. Four tree-based algorithms are used in this evaluation due to their low algorithm complexity and overall scalability: binomial, binary, start, and linear. Figure 9 reports the average time to complete a barrier for these topologies on the AIREN network.

Binomial Tree. This topology utilizes all the possible channels on each node. For example in a 4-ary 2-cube each node has 4 neighbor nodes directly connected. Theoretically, message transmissions can happen in all the channels in parallel, which could achieve the highest topology parallelism.

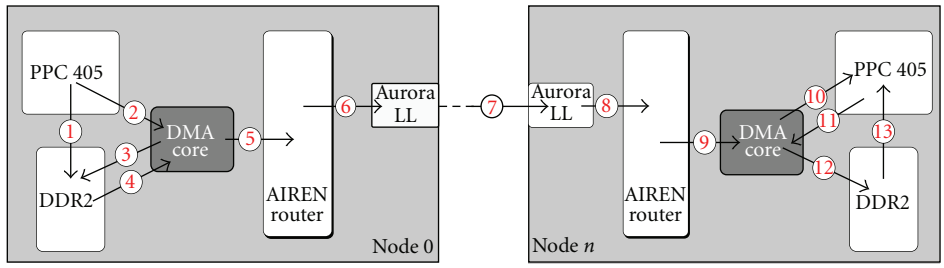

Figure 8: Example data flow of an AIREN hardware base `MPI_Send`/`MPI_Recv`.

Binary Tree. This topology is a subset of the binomial tree where each parent can only have at most two children. As the dimensionality of the network increases, this results in a reduction of the peak parallel performance as compared to the binomial tree.

Star Tree. This topology "virtually" connects the root node to all the other nodes. Messages hop through multiple nodes to the destination via the on-chip router. The benefit of this approach is that all nodes except the root node only need to act as children to the parent, simplifying the work done on each child node. However, when the number of nodes is large, the number of physical channels on the root node becomes the limiting factor causing contention and degrading the overall performance of the system.

Linear Tree. This topology has no parallelism. Every node has only one parent, and one child directly connected. Messages are relayed one node to another from the leaf to the root. This presents the worst case for barrier because the number of hops is linearly proportional to the size of the network.

5.3. `MPI_Reduce` in Hardware. The `MPI_Reduce` collective is a little more complicated than performing a barrier synchronization. The reduce core performs the specified reduce function in hardware, such as ADD or MAX, allowing the designer to take advantage of the parallelism within the FPGA when implementing each function [30]. In this approach, the reduce core receives its local data and fills the compute core's input pipeline. Each compute core then calculates its own local results. As results are produced, they are immediately transferred to the parent node. In fact, the tight integration between the computation and network creates one large pipeline between all of the nodes. Also, the only involvement from the processor is to begin the `MPI_Reduce` and provide a destination to store the final results. This significantly reduces the software overhead of traversing the network protocol stack.

When a reduce operation is initiated by the PowerPC (PPC), it tells the DMA engine what local data should be sent to the reduce core. The local data are fetched by the DMA engine and passed to the reduce core through the AIREN router. Depending on the role of the node (root, intermediates or leaf), the reduce core processes the data and transmits the result to the DMA on root or the parent node (intermediate or leaf). When the final result is sent to the DDR2 memory through the DMA engine, the reduce core interrupts the PPC indicating the reduce is complete.

Figure 10 overviews the reduce core's implementation on the FPGA, primarily comprised of three buffers, one or more compute cores, and the reduce control logic. Two input buffers are used for the compute core's input operands while the third buffer is for the computed results. The current implementation is designed to support double-precision floating-point data with a buffer size of 512 elements. Although the reduce core can be implemented with more FPGA resources to achieve better performance, the buffer size was selected to support the reduce operation of the processors (PowerPC or MicroBlaze) or custom hardware cores with minimal resource overhead, freeing up resources for other hardware accelerators or other collective communication cores. Furthermore, the reduce hardware core is operating at 100 MHz and is seriously handicapped compared to a commodity x86 class processor. If the reduction is to result in a single scalar value, the hardware core is very efficient. However, if the resulting reduction is a large vector, then it becomes a question of memory bandwidth and sequential computational speed.

The reduce core connects to the AIREN router through a single bidirectional connection which also reduces the resource utilization. As a result, the reduce core is designed to receive one message from the router at a time. To handle multiple incoming messages, FIFO_A is dedicated to store the data from local DMA transfers, FIFO_B stores the data from all of the children nodes, and FIFO_C stores the temporary result. FIFO_C can also feed the result back to the compute core if the node has more than two children. Likewise, for leaf nodes since no local computation is necessary (leaf nodes simply send their local data to their parents), FIFO_A is used to send data directly to the router and onto its parent.

While the compute core can be any associative operation, in these experiments, the compute core is a pipelined double-precision ADD unit. The ADD unit was chosen for its prevalence in a significant number of the `MPI_Reduce` function calls. The results presented in Figure 11 are more closely tied to the dataset size and the number of nodes rather than to the specific reduce operation being performed. By replacing the ADD unit with another compute core, only the difference in the latency across the compute core's pipeline will differ. The experiments are run over the four previously mentioned tree structures, and, as can be seen from the

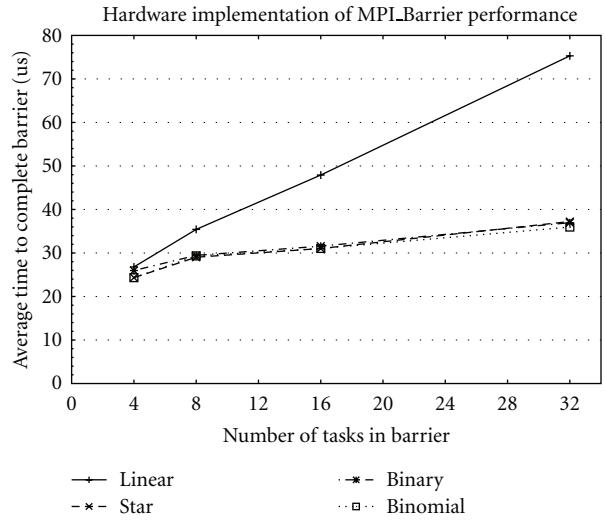

Figure 9: Measured `MPI_Barrier` performance of different network topologies on *Spirit*.

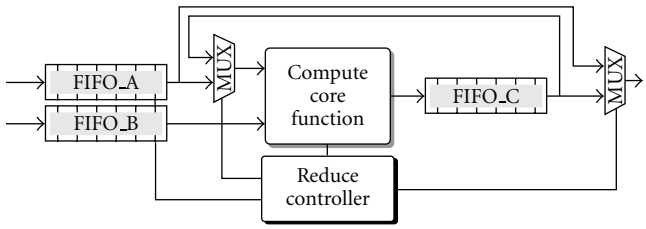

Figure 10: Block diagram of the hardware implementation of `MPI_Reduce`.

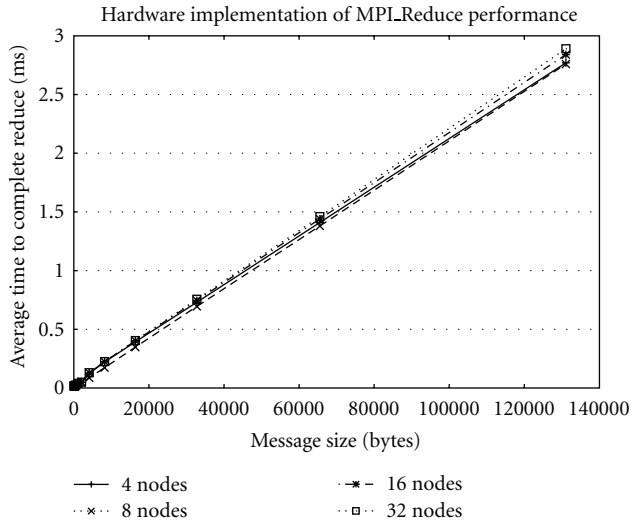

Figure 11: Measured `MPI_Reduce` performance of different network topologies on *Spirit*.

results, the average time to completion is less dependent on the tree structure than was found for barrier. Finally, the reason the hardware `MPI_Reduce` is linear is an artifact of our synthetic benchmark. A tight loop just performing reductions allows the hardware to organize its self into a deep pipeline with one task starting the next reduction before the previous one completely finished.

6. Integrating Compute Cores with AIREN

To demonstrate the capabilities of the AIREN network, a custom compute core was designed to perform matrix-matrix multiplication. Matrix-matrix multiplication can be decomposed into many multiply and accumulate (MAcc) steps. Figure 12 shows how the single-precision multiplier and adder are combined to form one multiply accumulate unit. The two inputs for the row and column values are connected to the inputs of the multiply unit. The result of the multiplication is one of the inputs to the adder. The other input to the adder is one of eight accumulation registers. The accumulation registers are necessary to hide the latency of the adder. The number of registers is proportional to the latency of the adder. If only one accumulation register was used, then the system would have to stall while waiting for the previous step's accumulation to finish. In this system the intermediate results of independent multiply-accumulate operations are being stored in the registers.

Floating Point Unit. While many FPGA-based floating point units have been presented in the literature [31–36], we chose to use the parameterizable floating point unit generated by the Xilinx CoreGen utility. The parameters of the floating point core include precision (single, double, or custom), utilization of DSP48 primitives in the FPGA, and latency (which affects both clock frequency and resource utilization). We wanted to maximize the number of multipliers and adders, while maintaining at least a 100 MHz clock frequency, use 100 percent of the DSP48 units, and approximately 50 percent of the LUT and FF units in the V4FX60 FPGA. We configured the multiply unit with a six clock-cycle latency, no DSP48 slices, 500 LUTs and 500 FF. The single-precision

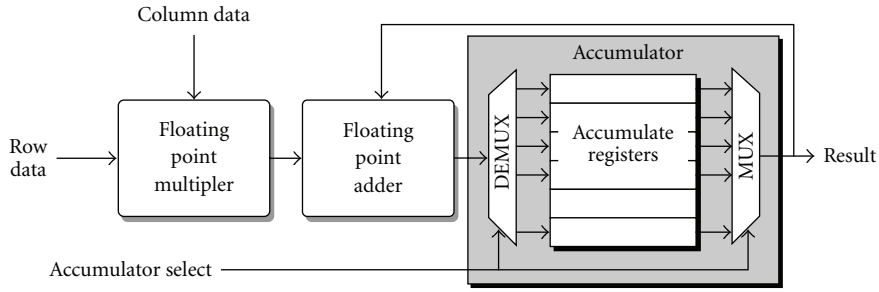

Figure 12: Single precision multiply accumulate unit.

adder was configured to have a seven clock-cycle latency and use four DSP48 slices, 500 LUTs and 500 FFs. These resources allowed us to instantiate up to 32 multipliers and 32 adders while using only half of the LUT and FF resources while maintaining the 100 MHz minimum system clock.

MAcc Array. Figure 13 shows MAcc units assembled into an array to support matrix-matrix multiplication. The FIFOs around the edge of the array help to keep new data available every clock cycle. The MAcc units are connected as an variable-sized array. On an FPGA the available resources limit the size of the array. For the Virtex-4 FX60 FPGA on the ML410 development board, there are 128 discrete processing slices (DPS). Single-precision floating point multiplication and addition can consume zero to several DSPs based on the desired operating frequency and latency. Therefore, a critical design decision is the size of the MAcc array.

Application-Specific Memory Controller. It has been shown that very low memory bandwidth is achievable for random access to off-chip memory from CCMs implemented in FPGA logic [37]. The MAcc array does not need random access to memory. We have developed a controller which uses burst transactions to off-chip DDR2 memory to request sequential blocks of data to fill the FIFOs in the MAcc array. This is a commonly overlooked advantage to building CCMs in FPGAs—you can often build a memory controller running in parallel to the computation unit which will request data in exactly the right order for the computation. This lessens or eliminates the need for a memory hierarchy and can fully utilize the relatively small off-chip memory bandwidth present in most FPGA-based systems. The FPGA system uses a multiport memory controller (MPMC) wrapper around the low-level DDR2 memory controller. The MPMC connects to the PowerPC processor and the MAcc array via a native port interface (NPI) connection. A custom controller was written to implement the NPI standard interface and fill the requested data into the row and column FIFOs.

6.1. MAcc Array Integrated with AIREN. Integrating the MAcc array with AIREN includes a processor for control and coordination, off-chip memory controller to provide access to the large matrices to be accessed by the application specific memory controller, and the network IP cores for AIREN. Figure 14 illustrates this configuration, along with the MPI hardware accelerator cores that can be used to support `MPI_Send/Recv`, `MPI_Barrier`, `MPI_Broadcast`, and MPI_Reduce. The base architecture includes all but the hardware accelerator core, in this example the MAcc array. Development begins with this initial infrastructure already in place. Therefore, the designer needs only to integrate the new core within this construct, saving significant development time since the base system is already functionality in place.

While early efforts focused on the use of Ethernet prior to the development of AIREN, the performance bottleneck of the Ethernet network was far too limiting. Instead, AIREN, provides a significantly greater bandwidth and much tighter integration between the hardware-accelerator cores and off-chip memory. The use of the AIREN network does require additional resources that are not necessary with a traditional Ethernet approach. In this study the initial Ethernet design used a 16×2 (almost fully utilized FPGA resources) MAcc array, but used Fast Ethernet instead of the AIREN network. The single node performance yielded ≈ 3.8 GFLOPS; however, as the system was implemented on the *Spirit* cluster, the Ethernet network quickly limited the performance as the system scaled. A significant amount of time was spent by each node waiting for the transfers of data between nodes to complete.

To include the AIREN network required reducing the size of the MAcc array, from 16×2 to 8×2. This meant the peak performance would drop from 6.4 GFLOPS to 3.2 GFLOPS (100 MHz clock frequency). Ultimately, the sacrifice in performance was justified based on the insufficient performance provided by fast Ethernet. Comparable tests were run for the 8×2 MAcc array and the performances are presented in Figure 15.

In this implementation, all network transfers are performed across the AIREN's high-speed network. Transfers are also performed in parallel (for both matrix A and matrix B). This is due to the fact that each node has two DMA cores and can be performing an `MPI_Send` and `MPI_Recv` on each matrix in parallel. Furthermore, the hardware implementation of `MPI_Barrier` is used to speedup synchronization between nodes. Overall, the results show significant performance gains even over a much larger 49 node implementation with twice the computed resources (16×2 versus 8×2). Certainly it makes sense that,

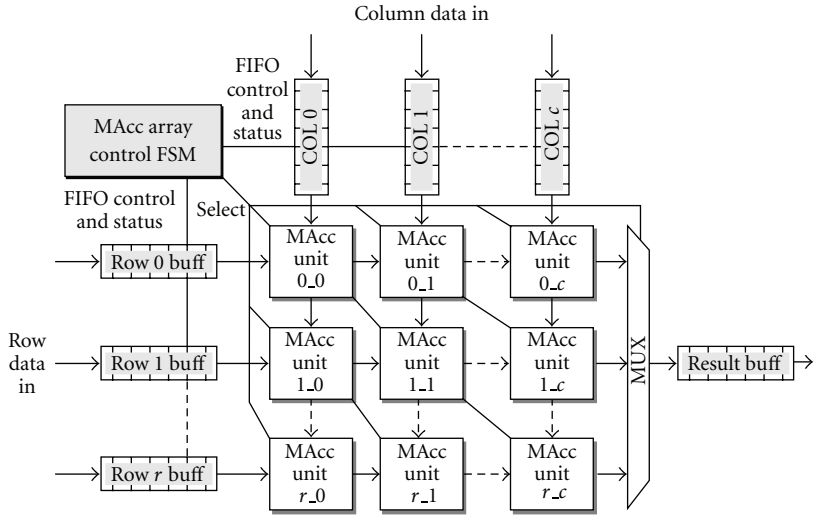

Figure 13: A variable sized array of MAcc units.

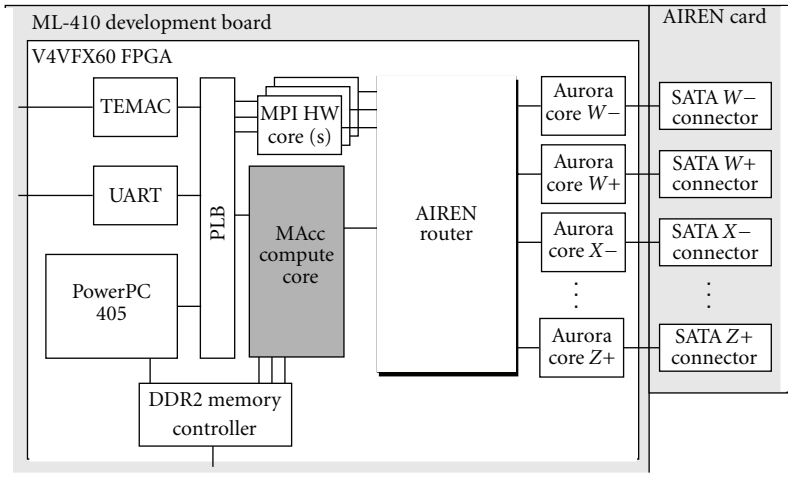

Figure 14: Single ML410 base system configuration for MAcc Array with AIREN.

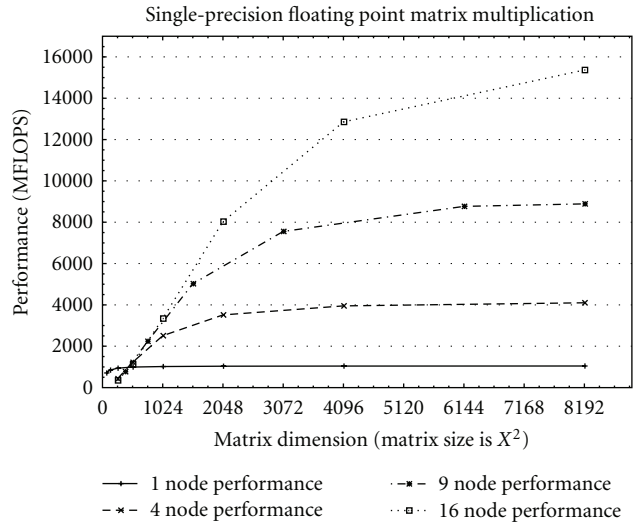

Figure 15: Performance of the 8 × 2 MAcc array implemented on a single, 4, 9, and 16 nodes of the Spirit cluster with the AIREN network.

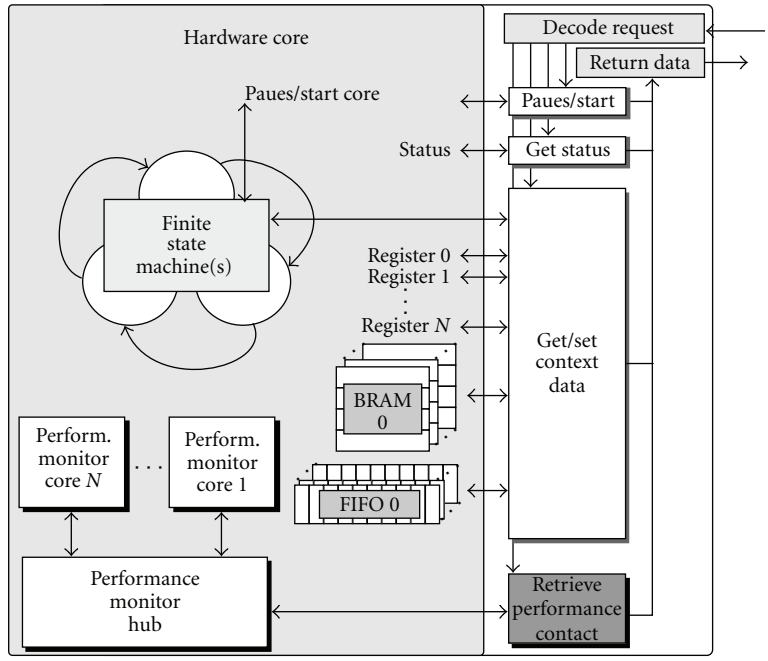

Figure 16: Block diagram of hardware core and its performance monitoring infrastructure.

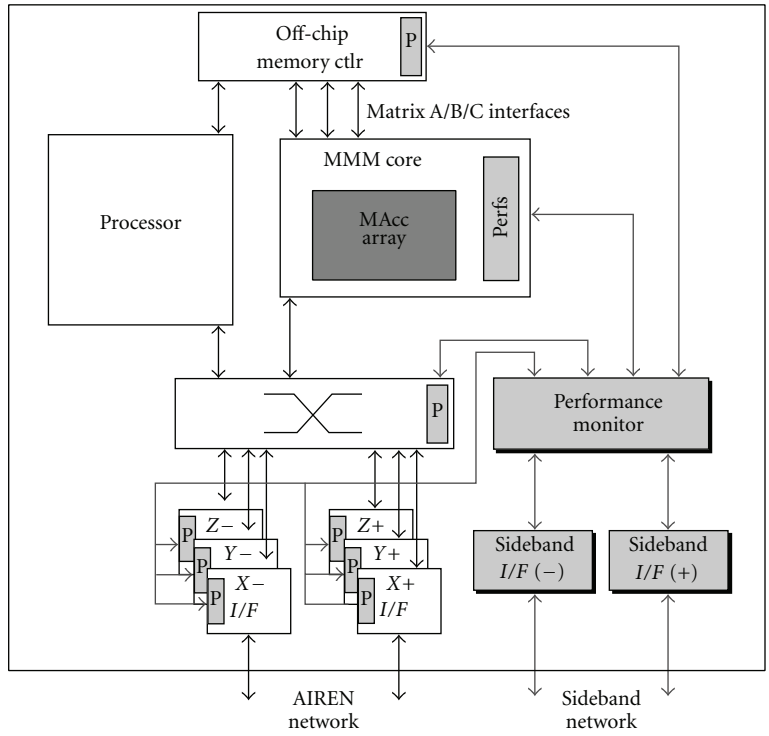

Figure 17: Block diagram of the performance monitor infrastructure used in the matrix-matrix multiplication evaluation.

when transferring large datasets between nodes, the higher bandwidth AIREN implementation will outperform Fast Ethernet. However, for a designer to realize the need to sacrifice single node performance to include the necessary on-chip infrastructure for the network might not be so easily identifiable.

7. Performance Monitoring with AIREN

Another unique aspect of the AIREN on-chip/off-chip network is that it can be used to support a sideband network independent of the primary data network. Recent work [38, 39] has begun to explore using this sideband network to

monitor multichip system performance. Because the monitoring is handled in hardware, it has the ability to probe more extensively and less invasively than a processor core that uses the primary network. Capabilities provided by this network include the ability to retrieve status (health), get/set context (state), and performance (detect stalls, e.g.). The motivation for this research and its integration into the existing AIREN network is that in high-performance computing, as the number of tasks increases it becomes increasingly difficult to quickly and efficiently perform basic capabilities like checkpoint/restart and is even more complicated to identify whether the system is making progress towards a solution.

The ability to retrieve status information and get/set context may improve system resiliency; however, we envision a system that is more proactive. By monitoring the performance of hardware cores and other key components (channel utilization, rate of interrupts, how often FIFOs are empty, etc.), we hypothesize that it is possible to characterize and more accurately predict when failures will occur as well as determine if useful work is being performed. Of course, the increasing amount of performance data per node requires an efficient network. Current research uses the two remaining network channels on the AIREN network to form a sideband network as a ring through the cluster. Performance monitoring data can be passed between nodes without interfering with the primary AIREN network.

To accomplish this, we have integrated a performance monitoring infrastructure (consisting of a monitor hub and monitor cores) into the existing monitoring system to allow a separate monitoring node access to the performance information without any additional overhead on the cluster's compute nodes. The *performance monitor hub* is instantiated within the hardware core, an example of which can be seen in Figure 16. The purpose of the hub is to collect the performance data and send it to the monitoring node for future evaluation. The sampling period can be adjusted down to microsecond intervals, depending on the amount of data to be sent per node.

A *performance monitor core* connects to the performance monitor hub and performs a specific monitoring function. Examples include timers and counters, to more complicated pattern detectors, utilization and efficiency monitors, and histogram generators. These, and many more, are regularly added to hardware cores (a convenient feature FPGAs offer to the HPC community) by a designer. We have created a repository of these monitor cores that, with this performance monitoring infrastructure, can be easily added to a design. Since many designs do not fully utilize all of the FPGA resources, the cost is often negligible. For a designer to see runtime information at such a low level (without introducing interference to the system and processor) will also result in more efficient designs.

While analyzing the performance of the 8×2 MAcc Array, it became clear that even on a single node the effective performance was approximately a third of the theoretical peak performance. Some of the performance loss can be attributed to software overhead; however, in order to fully understand the performance sink, the performance monitoring system was integrated into the system. For the purposes of this experiment, the MAcc Array, memory controller interfaces, and network interfaces were monitored with the intention identify to which core(s) was under performing. Figure 17 illustrates how the performance monitoring infrastructure is incorporated into the existing matrix-matrix multiplication design and with the AIREN network.

From this runtime performance data, it was identified that the custom memory controller to read the matrices from off-chip memory was severely underutilized. In effect, the memory controller was obtaining ≈32% of the available bandwidth (0.53 GB/s out of 1.6 GB/s). The custom memory interface performs requests per row; however, the row size was modified from 16 to 8 without redesigning the memory controller interface. Furthermore, the utilization of the on-chip buffers within the MAcc array were underutilized, often only storing as much as a single row of the submatrix. With a more efficient memory interface, this utilization will increase; however, a resource tradeoff could be made by reducing the size of the buffers to the largest submatrix to be evaluated.

8. Conclusion

8.1. Summary. The integrated on-chip/off-chip network developed under the RCC project was originally proposed in 2005. It has changed considerably since then, largely because application development informed our decisions. As this paper reports, the network has proven to be general enough to cover a wide range of applications (from MPI point-to-point messages to system-wide monitoring) while maintaining scalability.

Perhaps the most striking difference between the network architecture presented here and NoC literature is the use of a single full crossbar switch. Our results are different from other efforts due to several reasons. First, our target is the programmable logic of an FPGA not CMOS transistors, and this affects how one writes the hardware description language. As a result, we achieve much better resource utilization. Also, because our target frequency is much lower (100s of MHz) instead of Gigahertz, the physical distance that a signal can travel in one clock period is different. Specifically, our "reach" is in most cases the entire chip whereas future gigahertz CMOS chips might only be able to reach a fraction of the way across a chip. Second, most NoC assumes that a full crossbar is too expensive in terms of resources. However, within the context of a HPC system, the question is "what is too expensive?" In the case of an FPGA, the programmable logic resources are fungible. In most of the HPC applications we have worked with, the compute accelerator cores provide an enormous amount of computation. Overall performance is limited by the bandwidth off-chip so adding additional compute cores does not improve performance. In that circumstance, it makes sense to use as much of the available resources as possible to increase the network performance.

8.2. Future Work. We are presently designing a replacement for *Spirit*. It will likely use Virtex 7 devices that have 36

10 Gbps transceivers organized into a 5-ary 4-cube where each link is four-bonded transceivers yielding a bisection bandwidth of 20 Tbps. Even though the number of FPGAs is scaled from 64 to 625, the impact on our network design is minimal. This is because we are going from six network ports to eight network ports on the crossbar. In contrast, because of Moore's law, the size of a single FPGA will grow dramatically. Our Virtex 4 devices have around 57,000 logic cells; the Virtex 7 will likely have between 500,000 and 750,000 logic cells.

References

[1] A. G. Schmidt, W. V. Kritikos, R. R. Sharma, and R. Sass, "AIREN: a novel integration of on-chip and off-chipFPGA networks," in *Proceedings of the 17th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '09)*, April 2009.

[2] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "LiPaR: A light-weight parallel router for FPGA-based networks-on-chip," in *Proceedings of the ACM Great Lakessymposium on VLSI (GLSVLSI '05)*, pp. 452–457, April 2005.

[3] S. Kumar, A. Jantsch, J. Soininen et al., "A network on chiparchitecture and design methodology," in *Proceedings of the IEEE Computer Society Annual Symposium on (VLSI '02)*, vol. 102, pp. 117–124, 2002.

[4] R. Gindin, I. Cidon, and I. Keidar, "NoC-based FPGA: architecture and routing," in *Proceedings of the 1st International Symposium on Networks-on-Chip (NOCS '07)*, pp. 253–262, May 2007.

[5] G. Schelle and D. Grunwald, "Exploring FPGA network on chip implementations across various application and network loads," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 41–46, September 2008.

[6] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference*, pp. 684–689, June 2001.

[7] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, pp. 71–121, 2006.

[8] I. Walter, I. Cidon, A. Kolodny, and D. Sigalov, "The era of many-modules soc: revisiting the noc mapping problem," in *Proceedings of the 2nd International Workshop on Networkon Chip Architectures*, ACM, 2009.

[9] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis, "Evaluating bufferless flow control for on-chip networks," in *Proceedings of the 4th ACM/IEEE International Symposium on Networks on Chip (NOCS '10)*, pp. 9–16, May 2010.

[10] R. Sass, W. V. Kritikos, A. G. Schmidt et al., "Reconfigurable Computing Cluster (RCC) Project: investigating the feasibility of FPGA-based petascale computing," in *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'07)*, IEEE Computer Society, April 2007.

[11] P. Kogge et al., "Exascale computing study: technologychallenges in achieving exascale systems,," Tech. Rep. TR-2008-13, DARPA Information Processing Techniques Office (IPTO), 2008, http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf.

[12] K. D. Underwood, W. B. Ligon III, and R. R. Sass, "An analysis of the cost effectiveness of an adaptable computing cluster," *Cluster Computing*, vol. 7, pp. 357–371, 2004.

[13] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[14] D. A. Buell, J. M. Arnold, and W. J. Kleinfelder, *Splash2: FPGAs in a Custom Computing Machine*, Wiley-IEEE Computer Society Press, 1996.

[15] D. Burke, J. Wawrzynek, K. Asanovic et al., "RAMP blue: Implementation of a manycore 1008 processor system," in *Proceedings of the Reconfigurable Systems Summer Institute*, 2008.

[16] C. Pedraza, E. Castillo, J. Castillo et al., "Cluster architecture based onlow cost reconfigurable hardware," in *Proceedings of the International Conferenceon Field-Programmable Logic and Applications*, pp. 595–598, IEEE Computer Society, 2008.

[17] M. Saldana and P. Chow, "TMD-MPI: an MPI implementationfor multiple processors across multiple FPGAs," in *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pp. 1–6, IEEE Computer Society, 2006.

[18] A. P. Michael Showerman and J. Enos, "QP: a heterogeneousmulti-accelerator cluster," in *Proceedings of the International Conferenceon High-Performance Cluster Computing*, 2010.

[19] P. P. Kuen Hung Tsoi, A. Tse, and W. Luk, "Programming framework for clusters with heterogeneous accelerators," in *Proceedings of the International Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2010.

[20] NSF Center for High Performance Reconfigurable Computing (CHREC), "NOVO-G: adaptivelycustom research supercomputer," April 2005, http://www.xilinx.com/support/documentation/sw manuals/edk92i ppc405 isaext guide.pdf.

[21] RAMP, "Research accelerator for multiple processors," August 2008, http://ramp.eecs.berkeley.edu.

[22] R. Baxter, S. Booth, M. Bull et al., "Maxwell—a 64 FPGA supercomputer," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS '07)*, pp. 287–294, August 2007.

[23] Xilinx, "ML410 Development Platform—User's Guide UG085," 2007.

[24] Y. Rajasekhar, W. V. Kritikos, A. G. Schmidt, and R. Sass, "Teaching FPGA system design via a remote laboratory facility," in *Proceedings of the 18th Annual Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 687–690, IEEEComputer Society, September 2008.

[25] Xilinx, "Local Link Interface Specification SP006 (v2.0)," July 2005.

[26] Xilinx, "LogiCORE IP Aurora v3.0 Users Guide 61," 2008.

[27] R. G. Jaganathan, K. D. Underwood, and R. Sass, "A configurable network protocol for cluster based communications using modular hardware primitives on an intelligent NIC," in *Proceedings of the 11th Annual IEEE Symposiumon Field-Programmable Custom Computing Machines (FCCM '03)*, IEEE Computer Society, 2003.

[28] K. D. Underwood, *An evaluation of the integration ofreconfigurable hardware with the network interface in clustercomputer systems*, Ph.D. thesis, Clemson University, August 2002.

[29] S. Gao, A. G. Schmidt, and R. Sass, "Hardware implementation of MPI barrier on an FPGA cluster," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 12–17, August 2009.

[30] S. Gao, A. G. Schmidt, and R. Sass, "Impact of reconfigurable hardware on accelerating MPI_reduce," in *Proceedings of the*

International Conference on Field Programmable Technology (FPT '10), IEEE Computer Society, December 2010.

[31] J. Liang, R. Tessier, and O. Mencer, "Floating point unitgeneration and evaluation for fpgas," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 185–194, April 2003.

[32] S. Chen, R. Venkatesan, and P. Gillard, "Implementation of Vector Floating-point processing Unit on FPGAs for high performance computing," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE '08)*, pp. 881–885, May 2008.

[33] W. Ligon III, S. McMillan, G. Monn et al., "A re-evaluation of the practicality of floating-point operations on fpgas," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '98)*, p. 206, IEEE Computer Society, Washington, DC, USA, 1998.

[34] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, "Embedded floating-point units in FPGAs," in *Proceedings of the 14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '06)*, pp. 12–20, February 2006.

[35] K. S. Hemmert and K. D. Underwood, "An analysis of the double-precision floating-point FFT on FPGAs," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 171–180, April 2005.

[36] X. Wang, S. Braganza, and M. Leeser, "Advanced componentsin the variable precision floating-point library," in *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '06)*, pp. 249–258, IEEE Computer Society, Los Alamitos, Calif, USA, 2006.

[37] A. G. Schmidt and R. Sass, "Characterizing effective memory bandwidth of designs with concurrent High-Performance Computing cores," in *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 601–604, Amsterdam, The Netherlands, August 2007.

[38] B. Huang, A. G. Schmidt, A. A. Mendon, and R. Sass, "Investigating resilient high performance reconfigurable computing with minimally-invasive system monitoring," in *Proceedings of the 4th International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA '10)*, IEEE Computer Society, November 2010.

[39] A. G. Schmidt, B. Huang, and R. Sass, "Checkpoint/restart and beyond: resilient high performance computing with FPGAs," in *Proceedings of the 19th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'11)*, IEEE Computer Society, May 2011.