A Study of the On-Chip Interconnection Network for the IBM Cyclops64 Multi-Core Architecture

Ying Ping Zhang   Taikyeong Jeong   Fei Chen   Haiping Wu
Ronny Nitzsche   Guang R. Gao

University of Delaware
Department of Electrical and Computer Engineering
Newark, Delaware 19716, U.S.A
{yzhang, ttjeong, fchen, hwu, ggao}@capsl.udel.edu, Ronny.Nitzsche@s2000.tu-chemnitz.de

Abstract

The designs of high-performance processor architectures are moving toward the integration of a large number of multiple processing cores on a single chip. The IBM Cyclops-64 (C64) is a petaflop supercomputer built on multi-core system-on-a-chip technology. Each C64 chip employs a multistage pipelined crossbar switch as its on-chip interconnection network to provide high bandwidth and low latency communication between the 160 thread processing cores, the on-chip SRAM memory banks, and other components.

In this paper, we present a study of the architecture and performance of the C64 on-chip interconnection network through simulation. Our experimental results provide observations on the network behavior: (1) Dedicated channels can be created between any output port to input port of the C64 crossbar with latency as low as 7 cycles. The C64 crossbar has the potential reach the full hardware bandwidth, and exhibit a non-blocking behavior; (2) The C64 crossbar is a stable network; (3) The network logic design appears to provide a reasonable opportunity for sharing the channel bandwidth between traffic in either direction; (4) A simple circular neighbor arbitration scheme can achieve competitive performance level comparing to the complex segmented LRU (Least Recently Used) matrix arbitration scheme without losing the fairness. (5) Application-driven benchmarks provide comparable results to synthetic workloads.

1   Introduction

The designs of high-performance processor architectures are moving toward the integration of a large number of mul-

tiple processing cores on a single chip [14]. The performance scalability of such chips requires sophisticated interconnection network architectures and their behavioral evaluation should begin in the logical design and verification stage. In this paper, a study of the architecture and performance of the interconnection network of the IBM Cyclops-64 (C64) chip are presented.
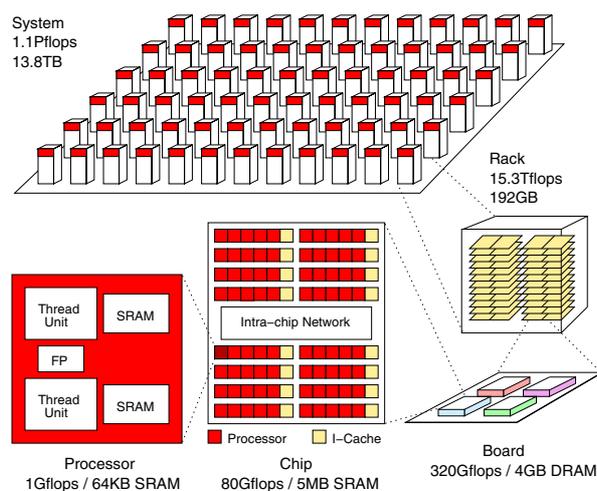
Figure 1. Cyclops-64 system Architecture.

The C64 system (Figure 1) is a petaflop supercomputer built on multi-core system-on-a-chip (SoC) technology, based on a cellular architecture and expected to achieve over one petaflop peak performance. A maximum configuration of a C64 system consists of *13,824* C64 processing nodes (1 million processors) connected by a 3D-mesh network [9].

Each node is composed of a C64 chip, external DRAMs and a small number of external modules. A C64 chip con-
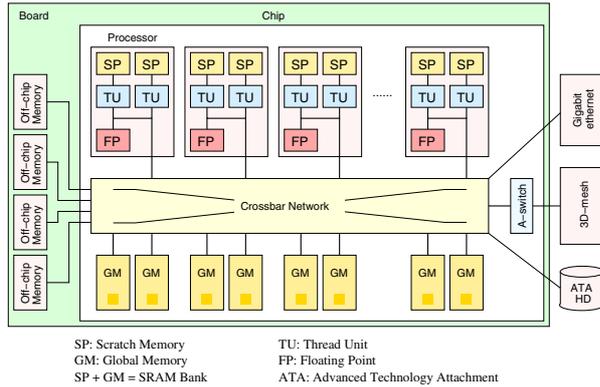
Figure 2. Cyclops-64 Node Architecture.

SP: Scratch Memory TU: Thread Unit
GM: Global Memory FP: Floating Point
SP + GM = SRAM Bank ATA: Advanced Technology Attachment

sists of up to *80* custom-designed 64-bit processors (each consists of two thread processing cores), *16* shared instruction caches (I-caches), *160* on-chip embedded SRAM memory banks and *80* floating point units (FP). It is interesting to note that there is no data cache on the chip. Instead, each SRAM bank on the chip can be configured into two levels: global interleaved memory banks (GM) which are uniformly addressable, and scratch pad memories (SP) that are local to individual processors [4].

The C64 chip configuration used in this study integrates *75* processors on a single chip. Each processor contains two thread units, one floating point unit and two 32KB SRAM memory banks. Groups of five processors share one I-Cache. Figure 2 shows the structure of the C64 chip.

The interconnection network embedded in the center of each chip is a seven stage pipelined crossbar switch with a large number of ports (*96* x *96*), which is much larger than any published design we are aware for SoC architecture. One primary motivation to employ a crossbar switch in the design of C64 chip architecture was to provide a uniform memory access model to significantly simplify the programming inside such a large scale multiprocessor chip. The ordering property of the crossbar was another valuable property of our design. We ensure there is a unique path from an input port to an output port and packets will be delivered in order.

The main challenges for the design of the on-chip interconnection network were the complication of the network and the limitation of the die area. A popular concern of crossbar switch is its quadratic cost in terms of chip area. In the C64 Chip design, the bandwidth of the crossbar increases linearly with the area, and quadratic with the number of ports. Besides, a moderate speed clock at 533MHz employed on the C64 chip leaves plenty room to grow, which is quite slow by today's multi-ghz standards. Consequently, with sophisticated architecture and engineering design, the completed crossbar is only *1.6* mm high and a

little under *17* mm wide (*27* mm^2). It turns out that the crossbar is just *6%* of the die area (the whole chip is *21* x *22 = 462* mm^2)!

To verify the performance of the design of the C64 crossbar switch in the logical and verification stage, two simulators were designed. An empirical analysis of the fully pipelined C64 crossbar network was conducted. The performance analysis was done under fixed channel width, node size, and topology constraints. Different parameters, such as workload types, traffic patterns, injection rates and arbitration algorithms, were tested during the performance simulation.

From the performance analysis, we have observed the following network behaviors: (1) The C64 on-chip crossbar has the potential to reach the full hardware bandwidth, and exhibit *a non-blocking behavior*[1]; (2) The C64 crossbar is a *stable network*[2]; (3) The network logic design provides a reasonable opportunity for sharing the channel bandwidth between traffic in either direction; (4) Simple arbitration schemes can achieve competitive performance level comparing to other more complicated schemes. (5) Application-driven benchmarks provide results comparable to synthetic workloads [18].

The experimental results and performance analysis have helped the architects for verifying the performance of the C64 chip interconnection architecture and conducted that a simple circular neighbor arbitration scheme instead of a complex segmented matrix arbitration scheme is chosen for the final design. This topic will be explored in sections 3 and 5 of this paper.

2 Background

A C64 chip consists of many simple, general purpose RISC style processor cores, shared I-caches, and multiple banks of embedded memory connected via a high-performance on-chip crossbar switch. A C64 system composed of thousands of C64 processors can achieve over one petaflop peak performance [8].

Verification and testing is a significant portion of the design cycle for chip performance analysis. For a pipelined crossbar network, verification should be conducted for two reasons: finding unforeseen phenomena that may happen in the interconnection (such as deadlocks and logic errors), and verifying the performance of the on-chip interface architecture [4].

In this paper, we are interested in the following questions regarding the C64 crossbar switch architecture:

- Will the C64 crossbar switch deliver the full pipelined

---

[1] Any two free ports can be connected, regardless of the settings in the switch

[2] The throughput does not degrade beyond the saturation point [5]

bandwidth if the communication traffic does not encounter network contention (i.e. non-blocking)?

- Can the C64 crossbar switch maintain its throughput when the network reaches its saturation (i.e. stable)?

- Can the C64 virtual channel mechanism be effectively exploited by the sharing between the forward and backward traffic?

- Can simple hardware arbitration mechanisms be used to achieve reasonable performance gains?

The rest of this paper will provide answers to these questions.

3 Architecture of the C64 Crossbar

3.1 An Overview of the C64 Crossbar
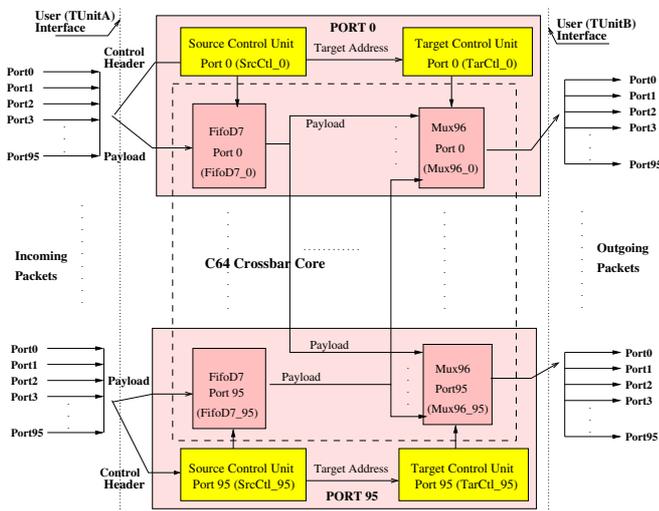
Figure 3. A block diagram of C64 crossbar

A crossbar switch has a physical element for every possible connection between ports, where every input has a cross-point with every output [3, 5]. The C64 crossbar is a *96* x *96* buffered crossbar switch in *7* pipelined stages with input/output queues. It is used to provide communication between the on-chip processors, SDRAM memory banks and I-Caches as well as off-chip DRAM memories, I/O devices, A-switches, and host interfaces [12, 16]. It possesses *96* bi-directional routing ports.

Figure 3 is a block diagram of the C64 crossbar switch where each routing port is presented as a small block. Each port of the crossbar can connect with the *95* ports and itself via its user interface. The user interface consists of an

incoming interface unit (TUnitA) and an outgoing interface unit (TUnitB). Each port owns a source control unit (SrcCtl), a target control unit (TarCtl), a 96-to-1 multiplexer (Mux96), a data FIFO (FifoD7), and some registers (not shown in this figure). In fact, the FifoD7 and their corresponding Mux96 in each port are combined into a C64 crossbar core providing a data path for the crossbar (See Figure 4). The FifoD7 employs two groups of data buffers store data for two virtual channels individually (not shown in the figure). A buffer group has *7* buffers with *92* bits each.

The packets delivered to the destinations through the *7* pipelined stages of the crossbar have a 95-bit fixed length. In principle, the minimum latency of the crossbar is *7* cycles assuming one cycle for each stage. A full hardware bandwidth of the crossbar is *96* x *95* = *9120* bits/cycle[3]. The packets are routed through the C64 crossbar by the SrcCtl of the source port and the TarCtl of the destination port. Flow control of the crossbar is implemented using a token protocol. Furthermore, the C64 crossbar provides two virtual channels for forwarded traffic (from the source processors to other destinations), and returned traffic (from other destinations to these processors, in case of a load operation), respectively. It also supports block transfers.

The C64 crossbar is designed to provide a non-blocking, stable interconnection network supporting efficient communication between components on the C64 chip. In the remaining part of this section, we will discuss the schemes used to support this approach, such as the flow control, routing scheme, virtual channels as well as the arbitration scheme of the crossbar.

3.2 Flow Control and Routing Scheme

Figure 4 presents a logic channel between connected source port *i* and destination port *j*, which is further divided into a control path and a data path. A data path between port *i* and port *j* is constructed by the SrcCtl of port *i* (SrcCtl_i) and the TarCtl of port *j* (TarCtl_j). A control path is built by the FifoD7 of port *i* (FifoD7_i) and the Mux96 of port *j* (Mux96_j). The routing processes inside the C64 crossbar are controlled by the SrcCtls and TarCtls.

Each single SrcCtl consists of a data FIFO (Fifo_C7), a small amount of registers, and some control information generators (not shown in figure). The functions of the SrcCtl are: (1) to generate control information which determines the operation mode (read/write) for the FifoD7s in the crossbar core; (2) to manage the data buffers inside each FifoD7 for input payloads; (3) to send requests to the TarCtl of the destination ports; (4) to deliver the chosen payloads to the Mux96s; and (5) to forward the control header to the TarCtl of the destination ports.

---

[3]Fixed 95-bit packets are transferred via each source-destination pair
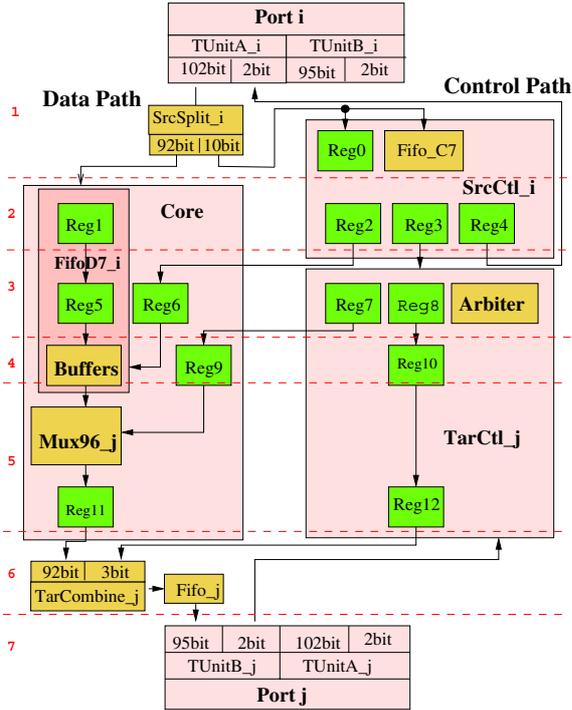
Figure 4. A Logic Channel of C64

Each TarCtl consists of an arbiter, several registers and control information generators (not shown in figure). The functions of a TarCtl are: (1) to generate control information to build a data path between the source port and the destination port; (2) to select a winner from all requested source ports; (3) to send back a token to the winning source port; and (4) to forward indication flags to the destination port.

The TarCtl at the destination port is used to choose a winner from all requested source ports competed for the same destination. It builds a data path between the source port and the target port. The SrcCtl and the TarCtl work together to route packets from sources to destinations through the 7 stage pipelined crossbar. The stages of the crossbar are divided by their functions. At each stage, the crossbar performs a specific function to provide a parallel communication via the crossbar.

Flow control of the C64 crossbar is realized by a token protocol, which is implemented by a 2-bit token and a token counter inside the interface of each port (not shown in the Figure). In this design, the input token counter is initialized to 7. Once a packet is injected into a port, the counter of its input token is decremented. Whenever a packet is delivered to its destination, an acknowledge token is fed back to the source port and the counter of its input token is incremented. Whenever the token counter of a port reaches 0, no further packets are allowed to be injected into this port.

Obviously, the C64 crossbar is a strictly non-blocking network [5] for both unicast and multicast traffic because any available output of the crossbar can be connected to any input by simply setting the output's multiplexer appropriately. At the same time, the crossbar should be a stable network. Many schemes, such as the buffer group scheme at each data FIFO, the injection and ejection queues at each port (See Figure 6), and the token flow control scheme are used to support this approach. They make it possible for the crossbar to continue delivering packets beyond the saturation point.

3.3 Arbitration Scheme

The C64 crossbar is a 96-way crossbar with 96 bi-direction routing ports. Each port needs a 96-to-1 arbiter for the flow control. Due to the large number of the arbitration circuits at the on-chip system, the fairness, speed, cost and memory space will play a very important role in the architecture design of the entire system. With this concern, a complicate segmented matrix arbiter was designed for the C64 on-chip crossbar.
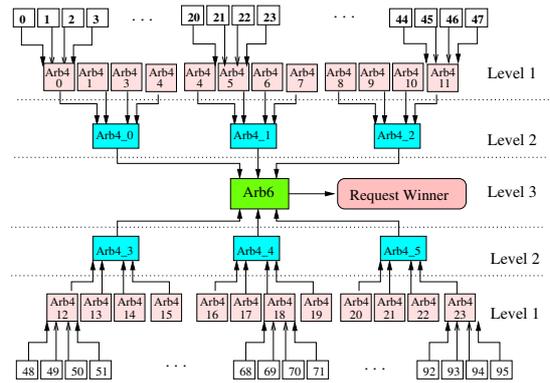
Figure 5. Segmented Matrix Arbiter

The arbiter is composed of *31* matrices segmented into three levels: level *1* with twenty-four *4* x *1* matrix arbiters, level *2* with six *4* x *1* matrix arbiters, level *3* with one *6* x *1* matrix arbiter (See Figure 5). All of the arbiters are organized hierarchically: outputs of level one are the inputs of level two, outputs of level two are the inputs of level three and the output of level three produces the request winner. During each cycle, only one winner can be selected at each port so that the maximum number of winners is *96* per cycle.

Comparing with a pure matrix scheme, the significant achievement of this design is the great savings in memory space, which is critical in an on-chip memory scheme. Generally, for each matrix arbiter with n inputs, n x (n - 1) / 2 bits are required to store the arbitration state. At the *96* arbiters, the required space would be *96* x *96* x *95* / *2* =

437760 bits (430K bits). Whereas, in a *96* segmented matrix arbiter, the required space would be *96* x (*30* x (*4* x *3 / 2*) + *6* x *5 / 2*) = *18720* bits (18K bits). Using a segmented arbiter, 412K bits can be saved for each chip and a great deal of space can be saved for an entire C64 system.

However, there still existed some questions about the segmented matrix algorithm:

- Does the segment affect its performance and fairness?

- Does it perform competitively compared with other common arbitration schemes?

Our simulation results and performance analysis successfully answered the questions and conducted the architects to use a simple circular arbiter to replace the complex segmented matrix arbiter (See section 5).

3.4   Virtual Channels

In a C64 chip, each ports of the crossbar switch are shared between a processor and its respective memory bank (See Figure 6).
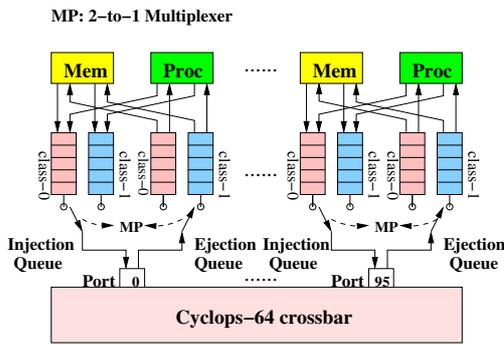
Figure 6. Virtual Channel Scheme

For full and fair utilization of crossbar bandwidth, a bi-directional mechanism called virtual channeling is provided to allocate bandwidth both for forward and backward traffic inside the crossbar. This scheme avoids traffic being blocked by a long block transfer in the other direction at the same port. If blocking occurs, an unpredictable delay will occur. At this design, two different packet classes, class *0* and class *1* are used to represent the forward and return data and types respectively. Both virtual channels share the same wires but each class has its own internal storage inside the pipeline stages, which ensure that the two virtual channels of the same path can transfer bi-directional data in parallel.

4   Experimental Framework

In this section, we introduce an experimental simulation platform of the C64 used in our performance study.

4.1   Simulation Platform of the C64
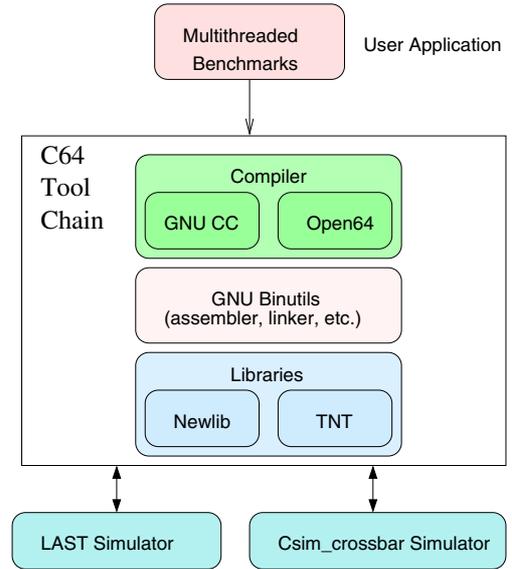
Figure 7. A Simulation Platform of the C64

As shown in Fig 7, the simulation platform consists of user applications (multithreaded benchmarks), a C64 tool-chain, and simulators. The C64 tool-chain integrates GNU CC, an assembler, a linker, and libraries which support C64 instruction set architecture. It provides a basic platform for early system software development and performance eveluation of the C64 system. The latency accurate simulators are designed to build software models for the C64 crossbar as well as other components on the C64 chip. With an accurate timing model, these simulators are sufficient for capturing, verifying and analyzing the performance characteristics of the multi-core on-chip architecture and were used at the design phase.

4.2   Simulators

The simulators, Csim_crossbar and LAST, are used to evaluate the efficiency of the hardware design and examine the new multi-core on-chip architecture. These simulators model the 96-port crossbar switch (the crucial part of the multi-core on-chip architecture), cycle accurately at the gate level, but model other parts (i.e the processors and memory banks, etc) at a functional level. This technique balances the simulation accuracy and speed. It provides a testbed for early performance analysis of the cellular architectures. Figure 8 shows several common parts inside both simulators for the performance testing. Input terminals provide different traffic and synchronization information by generating network parameters, packets and control information.
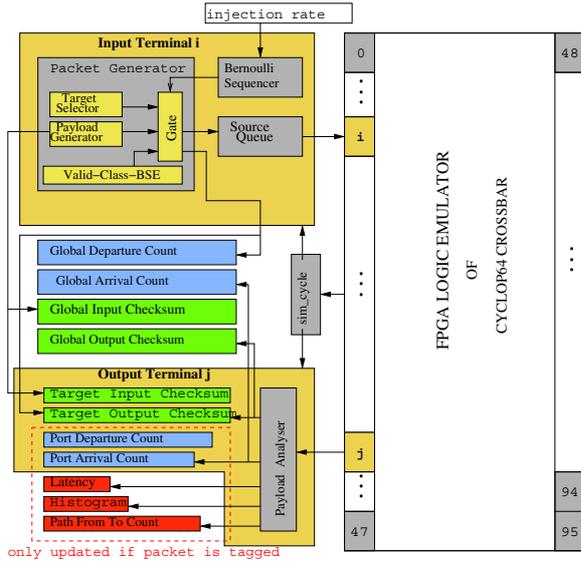
Figure 8. The C64 Crossbar Simulator

During simulation, incoming packets with particular distribution are generated in each input terminal and injected into the crossbar at a specific injection rate. The traffic pattern determines the target address inside each packet. Crossbar logic is a gate level cycle accurate simulator. It implements the communication between each source-destination pair on the crossbar. Output terminals are used to collect and calculate data, detect communication errors, and store test results. Finally, the data in various counters are analyzed and the performance results are reported.

These simulations played a very important role in the logic design process of the interconnection network of the C64 chip. For example, the simulator has evaluated and reported the efficiency of the hardware design under different arbitration schemes. Based on performance analysis of the test results, a simple circular arbiter scheme was chosen in the final C64 crossbar switch design instead of the segmented matrix arbiter. The latter is much more complicated than the former - see sections 3 and 5). The simple design has been shown not only to meet the cycle time requirements but also with similar fairness. Moreover, it is much easier to understand and debug.

5 Results

In this section, we present the simulation results of the C64 crossbar. Section 5.1 summarizes primary results and our observations. Section 5.2 analyzes latency of the C64 crossbar. Section 5.3 analyzes throughput of the C64 crossbar. Section 5.4 verifies the virtual channel scheme with micro-benchmarks and section 5.5 compares results from different arbitration schemes.

5.1 A Summary of Primary Results

In this research, we use both synthetic workloads and application-driven workloads to examine the interconnection network of the C64 on-chip architecture. The performance metrics, latency and throughput, are plotted as a function of network parameters, such as traffic patterns, virtual channel schemes on path, arbitration schemes, and injection rate. Our experimental results show that:

Observation 1 (See Section 5.2): Dedicated channels can be created between any output port to input port of the C64 crossbar with latency as low as *7* cycles. The C64 crossbar can achieve the full hardware bandwidth - i.e. exhibiting *a non-blocking behavior* [5].

Observation 2 (See Section 5.3): The C64 crossbar can maintain its throughput without any degradation even when the traffic load is beyond the *saturation* point exhibiting *stable network behavior* [5].

Observation 3 (See Section 5.4): Although the forward and backward traffic shares the same channel, the network logic design provides a reasonable opportunity for sharing the channel bandwidth between traffic in either direction. This is important for not giving a high priority only to the forward traffic because the load operations will generate return values (resulting in traffics in the reverse direction).

Observation 4 (See Section 5.5): Simple arbitration schemes can achieve the same or better performance gain than a complex segmented matrix arbitration scheme without losing the fairness.

Observation 5 (See Section 5.2 and 5.4): Application-driven benchmarks provide results comparable to synthetic workloads and constitute great metrics for verifying the design of the system architecture and analyzing performance behavior of the entire system.

5.2 Latency of the C64 Crossbar

Experimental results show that for the permutation spatial distributed traffic, the C64 crossbar has zero contention and both of its minimum and maximum latencies are always *7* cycles (See Figure 9), regardless of the injection rate. The results confirm that the C64 crossbar is a strictly *non-blocking* circuit-switched network because the permutation of the inputs and outputs can be forwarded without any conflict [5] and it requests no rearrangement for setting up dedicated paths between unused inputs and unused outputs.

The experimental results also present that without conflict in the destination port, the crossbar can be fully pipelined and reach the full hardware bandwidth. Its overall latency is caused only by the *7* pipeline stages of the C64 crossbar under our test conditions.
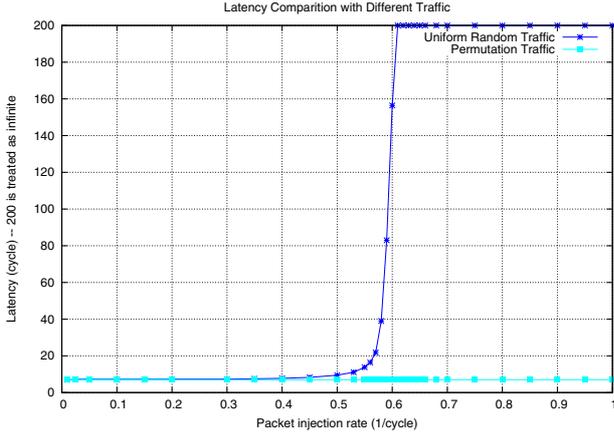
Figure 9. Latency for Different Traffic

For uniform random traffic, because of contention, the latency of the C64 crossbar increases toward infinite at the saturation point with about *0.6* of the injection rate.

In general, the most accurate way to measure the performance of a network is to use application-driven workloads, which generate sequences of messages applied to the network directly from the intended application [5].

In our experiment, LAST simulator makes it possible to generate application-driven workloads and to verify the observations we got from synthetic workloads. A set of multithreaded benchmarks were used to test the latency, such as hello-world, heat-condition, laplace, matrix-multiply, etc. Their average latency is listed in table 1.

Table 1. Average Latency of Benchmarks

Application Name	Average Latency (Cycles)
hello-world	7.036
matrix-multiply	21.590
heat-conduction	46.391
laplace	59.192

Table 1 show that most of the average latencies of tested application-driven benchmarks are much bigger than *7* cycles, the minimum latency of the C64 crossbar. It depends on the number of packets and traffic distribution generated by the LAST simulator. At the hello-world benchmark, only *2360* forward packets are generated and almost evenly dispatched to *77* of the 96 ports. So the possibility of contention is very low and the average latency of hello-world is very close to the minimum average latency of the C64 crossbar. However, in matrix-multiply benchmark, totally *2986233* forward packets are generated. However 2638187 of them, about 88

5.3   Throughput of the C64 Crossbar

The throughput behavior of a network beyond saturation point characterizes its stability [5]. In our experiments, throughput is measured at each source-destination pair and a modeling source queue is used to accurately simulate the injection rate without the effect of saturation. Figure 10 shows throughput results tested by the same workloads and traffic patterns as above.
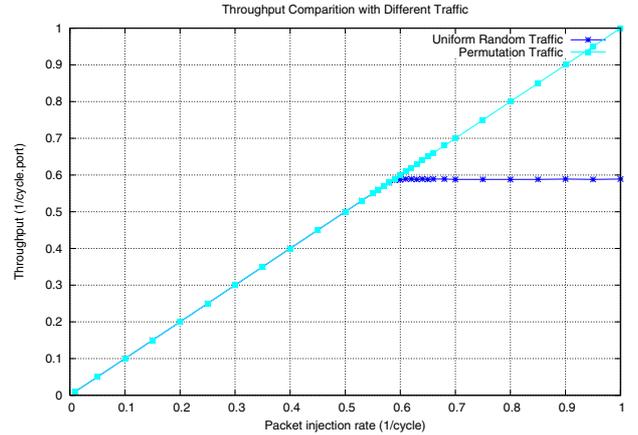
Figure 10. Throughput for Different Traffic

Figure 10 shows under permutation traffic, the throughput increases linearly as the injection rate because there is no contention. When contention exists under random traffic, the throughput of the C64 crossbar increases as the injection rate increases until the saturation point is reached, which is about *0.6*. Beyond saturation point, the throughput of the C64 crossbar does not degrade as the injection rate increases further. It confirms that the C64 crossbar is a stable network.

In order to verify the synthetic workload results with application-driven workloads, a set of micro benchmarks was designed. In this benchmark, a number of slave threads are created by the master thread. All threads write data to the same array located in the memory bank of port *0*. The program is compiled by the C64 tool chain and executed in LAST simulator, which generates application-driven workloads for the crossbar. Test results (Table 2) show a throughput comparable to the synthetic workloads.

5.4   Forward and Backward Traffic

In Section 3.5, we described how each port of the crossbar has virtual channels of class *0* and class *1* to handle both forward traffic and backward traffic. They are also able to execute in parallel.

To verify the advantage of the virtual channels, a multithread micro benchmark was designed and the LAST sim-

Table 2. Results of A Micro-Benchmark

Thread Number	Execution Time (cycles)	Received Packets	Throughput
2	21979	3986	0.182
10	35970	9803	0.273
20	44945	15909	0.355
40	63324	28345	0.448
60	82036	40782	0.498
80	98762	53143	0.539
100	118721	65632	0.553
120	136291	78075	0.573
150	162688	95944	0.590

Figure 11. Latency of Arbiter Schemes.

ulator was used to simulate all the chip logic and generate application-driven workloads for the crossbar switch. In the benchmark, two threads from different processors are assigned to execute two loops. In the first test, both loops keep writing data to the same global array located at memory bank *0* of port *0*. In this situation both loops are transferring forward traffic. In the second test, we use one loop to write and the other to read at the same time to the same array. In this case, both forward and backward traffic are created at the same port time. The results show that both cases have similar performance behaviors.

5.5 Arbitration Schemes

To study the performance behavior of different arbitration schemes, five of them are simulated: pure LRU matrix (PLRU)[4] [5], segmented LRU matrix arbitration schemes (SLRU), uniform random scheme (RAND)[5], circular neighbors scheme (CIRC)[6], and fixed priority scheme (WORS)[7]. Figure 11 and 12 show the compared results. The fixed priority scheme is the worst case.

The figures show that the fixed priority arbitration scheme has the worst performance; circular neighbor scheme can achieve a little bit better performance gain over others without losing the fairness; all other three schemes present very similar performance behaviors under uniform random traffic which uniformly and randomly generates packets and chooses destinations.

There are two reasons that could explain why all of four of the arbitration algorithms have similar performance. Either they all are fair arbitration schemes or there is a low contention probability for a random traffic in the network.

---

[4]Implement a least recently severed priority at a matrix scheme

[5]Choose winner uniform randomly from all requests

[6]Virtually align the ports in a circle and Choose the winner from the next request port clockwise or counterclockwise to the port of previous winner

[7]Set port 0 to the highest priority and port 95 to the lowest. All others are in between, respectively
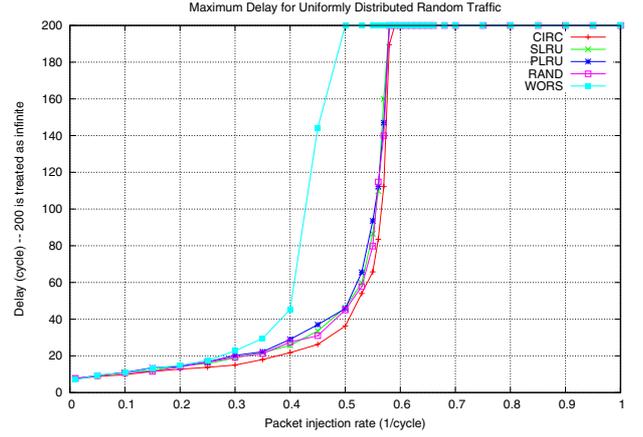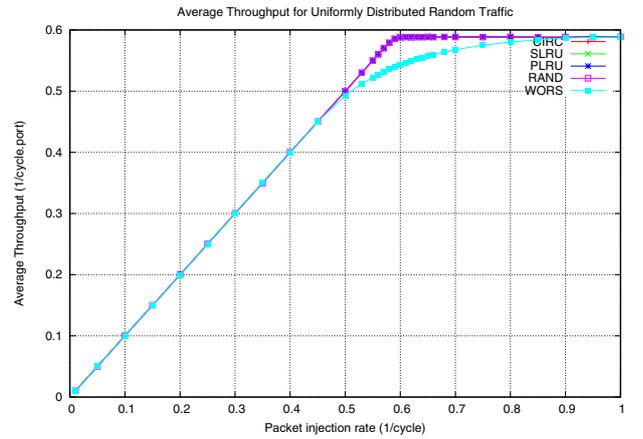
Figure 12. Throughput of Arbiter Schemes.

Let us define, P_0 as the probability of getting no packet, P_1 as the probability of getting only one packet at the same time, *n* as the input port number and *r* is the injection rate. Then we have:

$$P_0 = (1 - \frac{r}{n})^n \tag{1}$$

$$P_1 = r \, (1 - \frac{r}{n})^{n-1} \tag{2}$$

$$P_{contention} = 1 - P_1 - P_0 \tag{3}$$
$$= 1 - r \, (1 - \frac{r}{n})^{n-1} - (1 - \frac{r}{n})^n$$

We know *n = 96* and *r = 0 ∼ 1* so that the contention probability is about 12% at *r = 0.6* and it is about 27% at *r = 1.0*. It shows that there is at most about one contention every four packets at each port and there is no remarkable difference among the four arbitration schemes.

Our experimental results and performance analysis demonstrated that a simple arbitration scheme can achieve the same or better performance gain over the complex and expensive segmented LRU matrix scheme. It inspired questions about the original logical design of the segmented matrix arbiter of the C64 crossbar. Obviously a simple arbitration scheme, like circular neighbor arbiter, is much more cost efficient, easier to understand and design, and more space efficient. In a circular scheme, the storage size (the number of latch bits) is linear with the number of ports.

Consequently, a simple circular neighbor arbiter is applied to the hardware design of the C64 crossbar switch. It has been proved that the new design can not only make the cycle time but also is fairer.

6 Related work

Although there are a strong trend moving toward integration of multi-core processors on a single chip, there has been few publications of on-chip interconnection design and performance studies of large-scale multiprocessor-on-a-chip technology such as the C64 chip architecture reported in this paper.

A number of microprocessor chip vendors, leading by Intel, AMD and others, have chip design (some already begin appear in the market) that employ a small number of cores: i.e dual-cores, four cores, etc. However, they usually a very different architecture model by integrating the cache-based SMP architecture on a single chip. There are also a number of work on chip multiprocessors on a bigger scale such as Freescale multi-core programmable DSP (from Freescale Semiconductor, Inc [1]), picoArray (from picoChip Designs Ltd. [11]), etc., but we do not aware of any publications on the performance analysis of their interconnection network design in the scope of this paper.

There have been some research work on interconnection network under multiprocessor SoC architectures, such as the intra-chip switch [2], bus-based architecture [17], and crossbar-based architecture [13]. Some performance studies of interconnection networks for multi-core SoC architectures can be found in [13, 14, 17]. For example, Rekesh and his colleagues presented their study about the area, power, performance and designs issues for the on-chip interconnections based on a hypothetical chip multiprocessor [14].

In the past, crossbar switch network has been studied by a number of researchers including Franklin [10] and Mudge [15].

In the context of the C64 architecture, there are a number of performance studies reported [4, 6, 7, 8, 19]. However, these studies are not specific to the performance aspects of the C64 crossbar switch network as interested in this paper.

7 Conclusion

This paper presents the performance evaluation of the interconnection network of the IBM C64 multi-core architecture. These results show that the architecture of crossbar has the potential to deliver full pipelined bandwidth and exhibit non-blocking and stable behavior under certain traffic patterns. The virtual channel mechanism can efficiently balance traffic in either direction by multiplexing between the forward and backward traffic. This study shows, except the fixed priority arbitration scheme, all four others demonstrated very similar performance. Our performance analysis for arbitration schemes has inspired the architects of the C64 on-chip interconnection network to use a simple circular arbitration scheme to replace the complex segmented matrix scheme in the hardware design. In fact, it has been proved that this replacement not only achieves the required time cycle and performance but also reduces the complexity, cost and storage space. The results from application-driven workloads contributed a useful metric for verifying the design of the system architecture and analyzing the performance.

In addition to verification, the network of SoC for high performance computing should be analyzed by characterizing the architecture model, network interconnections, and overall system performance. The C64 crossbar switch architecture was simulated by the processing of comparing and analyzing the simulation results, and precisely presenting the observations based on both synthetic workload and application-driven workload distribution.

Acknowledgments

References

[1] Freescale announces industrys first 90nm multi-core programmable DSPs in volume production.

http://www.physorg.com/news4045.htm.

[2] L. Barroso, K. Gharacholoo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A scalable architecture based on single-chip multiprocessing. In *ISCA-27*, 2000.

[3] M. K. Chen, X.-F. Li, R. Lian, J. H. Lin, L. Liu, T. Liu, and R. Ju. Shangri-la: Achieving high performance from compiled network applications while enabling ease of programming. In *Proceedings of ACM SIG-PLAN 2005 Conference on Programming Language Design and Implementation (PLDI05)*, Chicago, Illinoi, June 2005.

[4] J. D. Cuvillo, W. Zhu, Z. Hu, and G. R. Gao. Fast: A functionally accurate simulation toolset for the cyclops64 cellular architecture. In *Workshop on Modeling, Benchmarking and simulation (MoBS), Held in conjunction with the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*, Madison, Wisconsin, June 4 2005.

[5] W. J. Dally and B. Towels. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[6] J. del Cuvillo, W. Zhu, and G. R. Gao. Landing openmp on cyclops-64: An efficient mapping of openmp to a many-core system-on-a-chip. In *ACM International Conference on Computing Frontiers*, Ischia, Italy, May 2006.

[7] J. del Cuvillo, W. Zhu, Z. Hu, and G. R. Gao. Toward a software infrastructure for the cyclops-64 cellular architecture. In *20th International Symposium on High Performance Computing Systems and Applications*, St. John's, Newfoundland and Labrador, Canada.

[8] J. B. del Cuvillo, W. Zhu, Z. Hu, and G. R. Gao. Tiny threads: a thread virtual machine for the cyclops64 cellular architecture. In *Proceedings of 5th Workshop on Massively Parallel Processing (WMPP05), in conjuction with the 19th International Parallel and Distributed Processing Symposium (IPDPS2005)*, Denver, Colorado, April 2005.

[9] M. Denneau. Computing at the speed of life: The blue gene/cyclops supercomputer. In *CITI Distinguished Lecture Series*, Rice University, Huston, Texas, September 25 2002.

[10] M. Franklin. Vlsi performance comparison of banyan crossbar communications networks. In *IEEE Trans. on Comp., vol. 30, no. 4*, pages 283–291.

[11] G. Panesar, D. Towner and A. Duller. Deterministic parallel processing. In *Proceesings of Micro-grid Workshop on Scalable on-chip Parallelism*, Amsterdam, July 2005.

[12] G. R. Gao, J. del Cuvillo, Z. Hu, R. Klosiwicz, C. Leung, J. McGuiness, H. Sakane, and Y. P. Zhang. *Programming Method and Software Infrastructure for Cellular Architecture*. Department of Electrical and Computer Engineering, University of Delaware, Newark, Delaware 19716, capsl techincal memo 48 edition, July 2003.

[13] L. Hammond, B. A. Nayfeh, and K. Olukotun. A single-chip multiprocessor. *Computer*, 30(9):79–85, 1997.

[14] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. *SIGARCH Comput. Archit. News*, 33(2):408–419, 2005.

[15] T. Mudge and B. Makrucki. Probablistic analysis of a crossbar switch. In *Proc. of the 9th Ann. Int. Symp. Computer Architecture*, pages 311–319, April 1982.

[16] H. Sakane, L. Yakay, V. Karna, C. Leung, and G. R. Gao. Dimes: An iterative emulation platform for multiprocessor-system-on-chip designs. In *IEEE International Conference on Field-Programmable Technology (FPT'03)*, Tokyo, Japan, December 2003.

[17] L. Zhang and V. Chaudhary. On the performance of bus interconnection for socs. In *Proceedings of 4th Workshop on Media and Stream Processors (in conjunction with IEEE/ACM MICRO-35)*, Istanbul, Turkey, November 2002.

[18] Y. P. Zhang. A study of architecture and performance of ibm cyclops64 interconnection network. Master's Thesis, University of Delaware, July 2005.

[19] W. Zhu, Y. Niu, and G. R. Gao. Performance portability on earth: A case study across several parallel architecture. In *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 15*, Apr 2005.