University of Delaware
Department of Electrical and Computer Engineering
Computer Architecture and Parallel Systems Laboratory

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

Yingping Zhang *Taikyeong Jeong* *Fei Chen*
Ronny Nitzsche *Guang R. Gao*

CAPSL Technical Memo Revised.1 60
Nov. 21, 2005

†Dept. of Electrical and Computer Engineering
University of Delaware
yzhang,ttieong,fchen,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 Cyclops64 (C64) is a petaflop supercomputer built on multi-core System-On-Chip (SOC) technology, based on a cellular architecture. A maximum configuration of a C64 system consists of 13,824 C64 processing nodes (around one million processors) arranged around a 3D mesh network [8, 7]. Each node is composed of a C64 chip, external DRAMs and a small number of external modules. 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 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) The C64 crossbar can achieve as low as 7 cycle latency, reach the full hardware bandwidth, and exhibit *a non-blocking behavior*; (2) It is a *stable network*; (3) The network logic design appears to provide a reasonable opportunity for sharing the channel bandwidth between traffics in either direction; (4) The segmented LRU matrix arbitration scheme does not have any notable performance gain comparing with some simple schemes. (5) Application-driven benchmarks provide comparable results to synthetic workloads.

Contents

List of Figures

1 Introduction

The designs of high-performance processor architectures are moving toward the integration of a large number of multiple processing cores on a single chip [12]. The performance scalability of such chips requires a solid interconnection network architecture and its behavioral evaluation should begin in the design and verification stage. In this paper, we present a study of the architecture and performance of the IBM Cyclops64 (C64) chip's interconnection network.

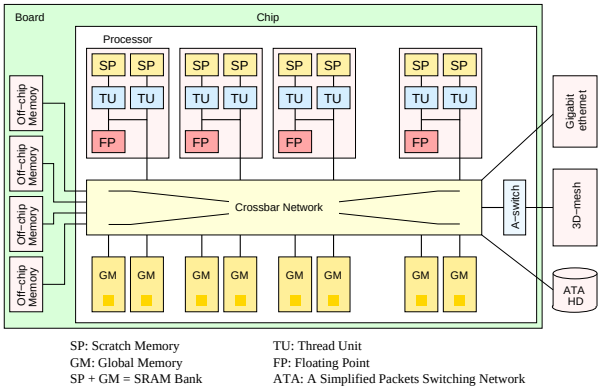

Figure 1: Cyclops64 Node Architecture.

The C64 is a petaflop supercomputer built on multi-core System-On-Chip (SOC) technology, based on a cellular architecture and expected to achieve over 1 petaflop peak performance. A maximum configuration of a C64 supercomputer consists of 13,824 C64 processing nodes (1 million processors) connected by a 3D-mesh network [8]. Each node is composed of a C64 chip, external DRAMs and a small number of external modules. A C64 chip consists of up to 80 custom-designed 64-bit processors, 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 1 shows the structure of the chip.

We conducted an empirical analysis of a fully pipelined C64 crossbar network. The performance analysis is done under certain constraints (fixed channel width and node size as well as topology). Different parameters, such as workload types, traffic patterns, injection rates and arbitration algorithms, are implemented during the performance simulation. Our experimental results provide the following observations on the network behavior: (1) The C64 crossbar can achieve as low as 7 cycle latency, reach the full hardware bandwidth, and exhibit *a non-blocking behavior*[1]; (2) It is a *stable network*[2]; (3) The network logic design appears to provide a reasonable opportunity for sharing the channel bandwidth between traffic in either direction; (4) The segmented LRU matrix arbitration scheme does not have any

[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]

notable performance gain comparing with some simple schemes. (5) Application-driven benchmarks provide comparable results to synthetic workloads.

The rest of this paper is organized as follows. Section 2 introduces the background of the study. Section 3 describes the details of the architecture of the C64 crossbar. Section 4 shows the experimental framework. Section 5 presents our major observations and performance analysis. Section 6 reviews the related work done on the study of architecture and performance of SOC multicore architectures. Finally, section 7 summarizes the conclusion.

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 supercomputer consisting of hundreds C64 processor can achieve over 1 petaflops peak performance [7].

The performance scalability of such chips requires a solid interconnection network architecture, whose behavior evaluation should begin in the design and verification stage. A significant portion of the design cycle for chip performance analysis is taken up by verification and testing. 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 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 traffics ?
- Can simple hardware arbitration mechanisms be used to achieve reasonable performance gains ?

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

3 Architecture of the C64 Crossbar

3.1 An Overview of the C64 Crossbar

A crossbar switch has a physical element for every possible connection between users, in which every input has a crosspoint with every output [5, 2]. The C64 crossbar is 96 x 96 buffered crossbar switch in

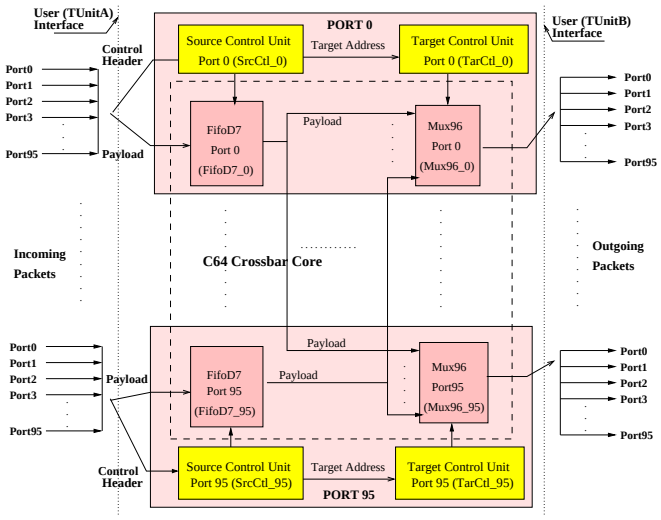

Figure 2: A block diagram of C64 crossbar

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 [13, 9]. It possesses 96 bi-directional routing ports.

Figure 2 shows that each port of the crossbar consists of an user interface, a source control unit, a target control unit, a 96-to-1 multiplexer, a data FIFO, and some registers. All of the 96 data FIFOs and their following 96-to-1 multiplexers are combined into a C64 crossbar core. Each of the data FIFOs employs two groups of data buffers (not shown in the Figure), storing data for two virtual channels individually. Each buffer group is composed of 7 buffers, of 92 bits each.

The packets delivered through the 7 pipelined stages of the crossbar to the destinations have a 95-bit fixed length. In principle, the least latency of the crossbar is 7 cycles, assuming one cycle for each stage. The full hardware bandwidth of the crossbar is 96 x 95 = 9120 bits/cycle[3]. The packets are routed through the C64 crossbar by the source control unit of the source port and the target control unit 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 others to the processors), respectively. It also supports block transfers.

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

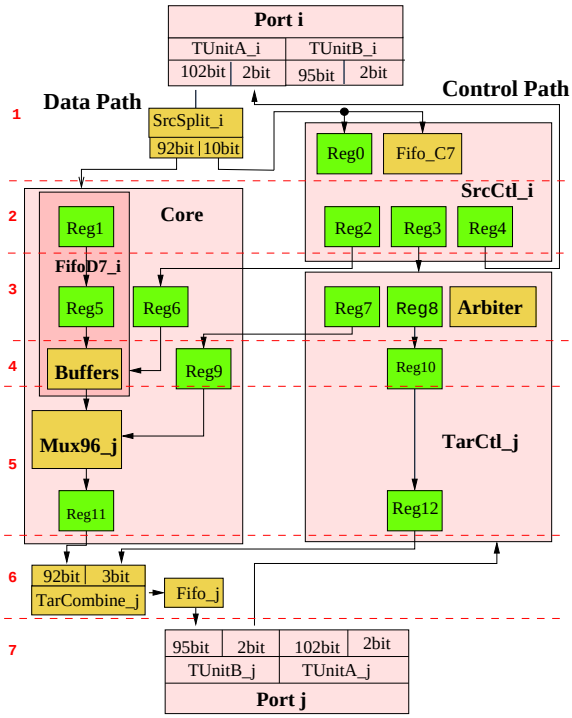

Figure 3: A Logic Channel of The C64

3.2 Routing Scheme & Flow Control

Figure 3 presents a logic channel between connected port i and port j, which is further divided into a control path and a data path. The former is constructed by the source control unit (SrcCtl_i) of port i (the source) and the target control unit (TarCtl_j) of port j (the destination). The latter is built by the data FIFO of port i (FifoD7_i) and the 96-to-1 multiplexer of port j (Mux96_j). The routing processes inside the C64 crossbar are controlled by source control units and target control units.

A source control unit consists of a data FIFO, a small amount of registers, and some control information generators (not shown in the Figure). The functions of the source control unit are: (1) generate control information to determine the operation mode (read/write) for the data FIFO; (2) manage data buffers for input payloads; (3) send a request to the target control unit of the destination port ; (4) deliver the chosen payload to the following multiplexer; and (5) forward the control header to the target control unit of the destination port.

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

The source control unit at a source port decides how to store the injected packets and select output

[3]Fixed 95-bit packets are transferred via each source-destination pair

packets from the buffers of its data FIFO. The target control unit at a destination port is used to choose a winner from all requested source ports competed for the same destination. It builds a path between the source port and the target port. The source control unit and the target control unit work together to route packets from sources to destinations through the 7 stage pipelined crossbar. The stages of the crossbar are divided due to their functions. At each stage, the crossbar performs a specific function to provide a parallel communication via the crossbar. The hardware design guarantees that the stages are able to be fully pipelined, although the pipeline may be partly stalled due to resource contention or shortage.

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, its input token counter decrements. Whenever a packet is delivered to its destination, an acknowledge token is fed back to the source port and its input token counter is incremented. When 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. Meanwhile, the crossbar is supposed to 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 5) as well as 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. After saturation, the network is going to maintain the highest throughput reached, because the packets are held in place (with delay) instead of being dropped.

3.3 Segmented Matrix Arbiter

The C64 crossbar is 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 such arbitration circuits at the on-chip system, the fairness, speed, cost and memory space required to store the states for the arbiter will play a very important role in the architecture design of the entire system. To address this issue, a segmented matrix arbiter is designed for the crossbar.

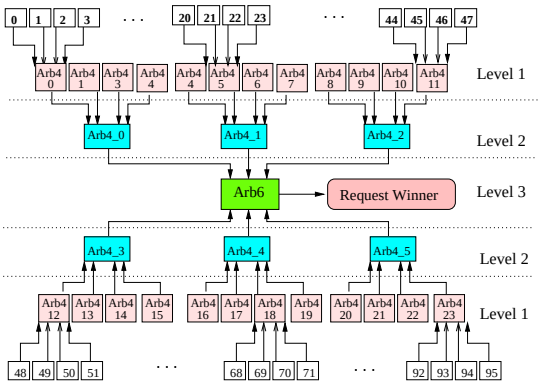

Figure 4: Segmented Matrix Arbiter

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 4). 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.

The significant achievement of this design compared with a pure matrix scheme 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. For 96 arbiters, the required space would be 96 x 96 x 95 / 2 = 437760 bits (430K bits). Obviously, this algorithm is unfeasible inside the crossbar in a real silicon. 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). It shows that 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 exist some questions about the segmented LRU matrix algorithm:

- Does the segment affect its performance and fairness?
- Does it perform competitively compared with other common arbitration schemes?

This paper will provides some answers to these questions. (See Section 5).

3.4 Virtual Channels

In a C64 chip, each of the 96 ports of the crossbar switch are shared between a processor and its respective memory bank (See Figure 5).

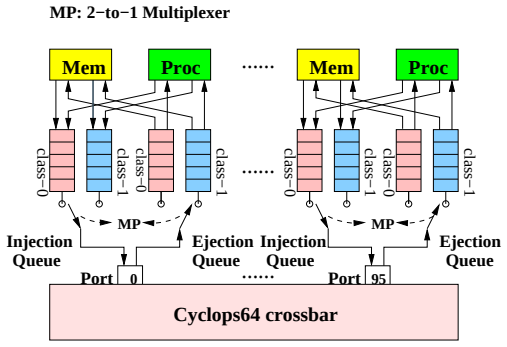

Figure 5: Virtual Channel Scheme

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 is used to avoid one direction's traffic being blocked by a long block transfer in another direction at the same port. If the blocking happens, an unpredictable delay would occur. In this design, two

different packet classes, `class 0` and `class 1` are used to represent the data type respectively. Here class 0 refers to the forward data. Class 1 refers to the return data. Both virtual channels share the same wires but each class has its own internal storage inside the pipeline stages. The two virtual channel of the same path can transfer bi-directional data in parallel, except competing for the same physical link via an arbiter.

Our experiments evaluate different behaviors of the network with various network parameters. Experimental results and observations will be presented in section 5.

4 Experimental Framework

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

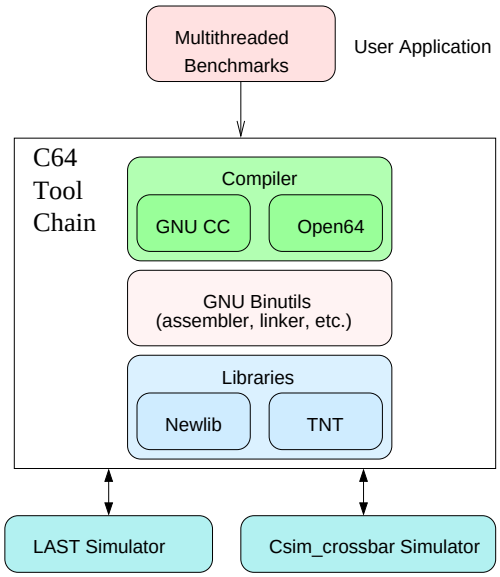

Figure 6: A Simulation Platform of the C64

As shown in Fig 6, it consists of user applications (multithreaded benchmarks), a C64 toolchain, and simulators. The C64 toolchain 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 testing of the C64. The gate level simulators are designed to build software models for all components of the C64 crossbar as well as other components on the C64 chip. They also model the timing constraints accurately for the system. We use it to detect bugs and verify the performance of the C64 system in the design phase.

4.1 Simulation Platform of the C64 (SPC)

Figure 6 shows that the combined experimental platform is composed of the following elements:

- Csim_crossbar simulator is a simulation tool written in C. It has following functions: (1) perform a cycle accurate simulation for a single C64 crossbar to verify the architecture; 2) provide synthetic workloads and traffic patterns to simulate the communication environment of the system; and (3) support early performance verification and testing for the C64 crossbar.

- LAST (Latency Accurate Simulation Testbed) simulator is also written in C . It has following functions: (1) perform a cycle accurate simulation for a single C64 chip to verify the architecture; (2) provide application-driven workloads and traffic patterns to simulate the communication environment of the system; and (3) support early performance verification and testing for the C64 chip.

- C64 software tool-chain is a test platform. It has following functions: (1) generate executable files for given multiple-thread benchmarks [6]; and (2) provide a platform for early system software development and testing.

4.2 Simulators

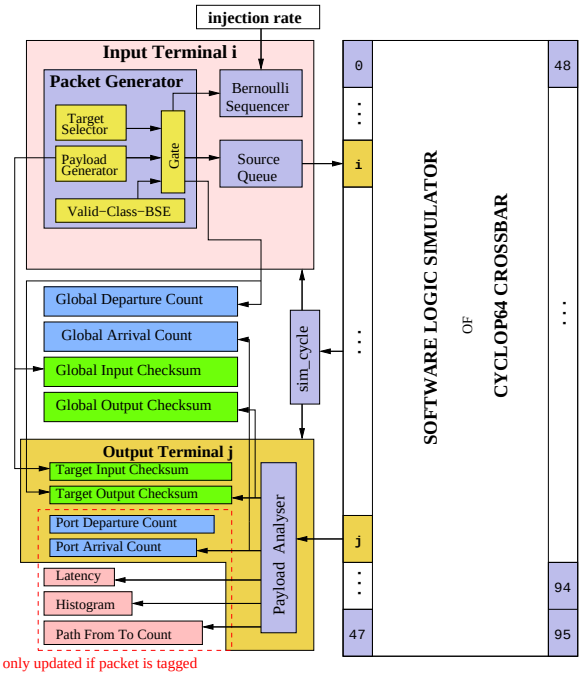

Figure 7: The C64 Crossbar Simulator

Csim_crossbar and LAST are the simulators used in our experiments. Figure 7 shows several common parts inside both simulators for the performance testing: Input Terminals, Crossbar Logic, local and global counters as well as Output Terminals. Input Terminals provide different traffic and synchronization information by generating packets, network parameters and control information. Once the simulator is running, incoming packets with particular distribution are generated in each Input Terminal and injected into the crossbar at a specific injection rate once a cycle. The target address inside

each packet is chosen according to the determined traffic pattern. Crossbar Logic is a gate level simulator, which simulates the logic of crossbar modules. It implements the communication between each source-destination pair via the crossbar. Output Terminals are used to collect and calculate data, detect communication errors, and store test results. Finally the simulators analyze and report the performance results from the information in various counters.

4.3 Assumptions of Simulation

In order to measure the latency and throughput of C64 crossbar, csim_crossbar and LAST simulators are used and the following assumptions are made:

- The processes generating packets at the sources are independent and memoryless. Each generated packet is inserted into a infinte incoming buffer related to the source port.
- It has a zero-load latency network [5] with infinite incoming and outgoing buffers out of the crossbar interfaces. Therefore, the contention delay over a shared resource outside the crossbar is ignored, and packet dropping will never happen.
- Each incoming packet has the same probability to win the arbitration. Each stage of the pipelined crossbar completes in one cycle.
- The network is treated as buffered: if a packet is blocked at some stage, it is considered to be buffered rather than to be lost. It does affect the future states of the system.
- Each port has infinite computing power, which means that the computation of routine functions takes zero time.
- The network is synchronous: packets move from stage i to stage i + 1 at each cycle.
- In the same port, the buffers of two classes can be counted independently.
- Injection and consumption of packets take zero time.

In addition to the above assumptions , we added one more constraint for non-blocking verification. It is assumed that the network only transfers messages from processors to memory banks, also called forward traffic.

5 Results

In this section, we are going to present the test result from the crossbar simulation. The chapter is organized as follows. 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 by micro-benchmarks. Section 5.5 explores the results from different arbitration schemes and section

5.1 A Summary of Primary Results

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

Observation 1 (See Section 5.2): We have observed that a dedicated channel can be created between a output port to any input port of the C64 crossbar with no contention and 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): We have observed that the C64 crossbar can maintain its throughput without any degradation even when the traffic load is beyond the *saturation* point i.e. exhibiting *stable network behavior* [5].

Observation 3 (See Section 5.4): We have demonstrated that although the forward and backward traffic shares the same channel, the network logic design appears to provide a reasonable opportunity for sharing the channel bandwidth between traffic in either direction. This is important because the load operations will generate return values (resulting in traffics in the reverse direction) and we should not give a high priority only to the forward traffic.

Observation 4 (See Section 5.5): We have verified that the complex segmented LRU matrix arbitration scheme, does not have any notable performance gain comparing with some simple arbitration argorithms, such as the uniform random scheme and the circular neighbors scheme, under our experimental condition.

Observation 5 (See Section 5.2 5.4): We have shown that 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 Analysis of the C64 Crossbar Latency

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 8), 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 its overall latency is caused only by the 7 pipeline stages of the C64 crossbar under our test conditions.

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.

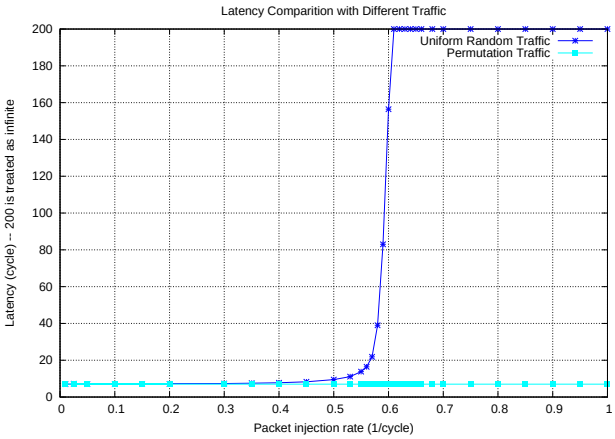

Figure 8: Latency for Different Traffic

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 multi-threaded benchmarks were used to test the latency, such as hello-world, heat-condition, laplace, matrix-multiple, etc. Their average latency is listed in table 5.2.

Table 1: Average Latency of Benchmarks

Application Name	Average Latency (Cycles)
hello-world	7.136
matrix-multiple	21.590
heat-conduction	46.391
laplace	59.192

5.3 Analysis of the C64 Crossbar Throughput

According to the definition [5], the throughput behavior of a network beyond saturation point characterizes its stability. In our experiments, throughput is measured upon each source-destination pair and a modeling source queue is used to accurately simulate the injection rate without the effect of saturation. The same workloads and traffic patterns are used in the study as Figure 9 indicates.

Figure 9 shows when contention exists, 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, which confirmed

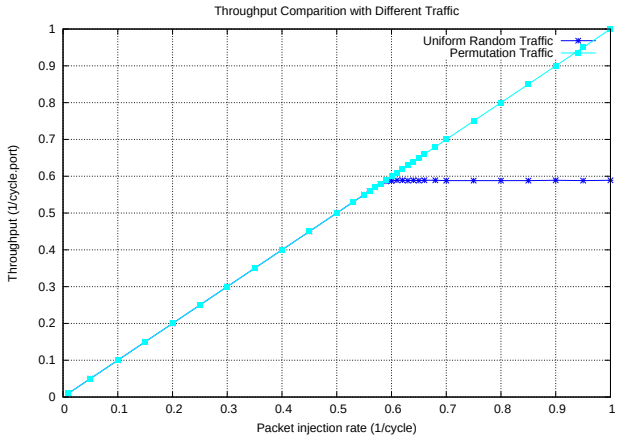

Figure 9: Throughput for Different Traffic

that the C64 crossbar is a stable network. Under permutation traffic, the throughput increases linearly as the injection rate because of no contention.

In order to verify the synthetic workloads' results with application-driven workloads, a set of micro benchmark is 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 show a throughput comparable to the synthetic workloads.

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

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 multi-thread micro benchmark was designed and

the LAST simulator 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 0 of port 0. It means both loops are transferring forward traffic in this situation. 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 is created at the same port time. The results show that both cases have similar performance behaviors.

5.5 Arbitration Schemes

Other than the pure LRU matrix (PLRU)[4] [5] and segmented LRU matrix arbitration schemes (SLRU), we also simulated three others to learn how they affect the crossbar performance. They are the uniform random scheme (RAND)[5], the circular neighbors scheme (CIRC)[6], the fixed priority scheme (WORS)[7]. Figure 10 and 11 show the compared results. The fixed priority scheme is the worst case.

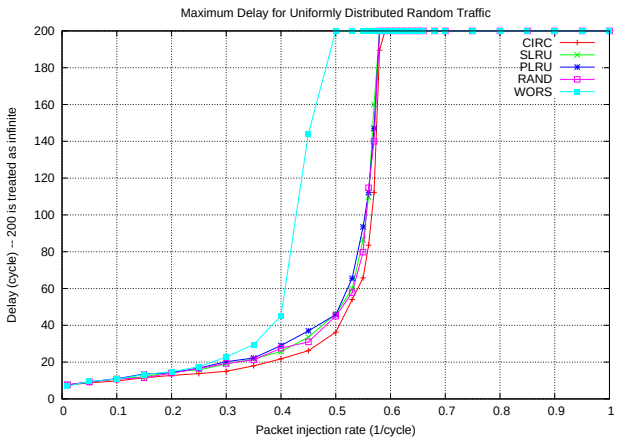

Figure 10: Latency of Arbiter Schemes.

From the figures, we can see that except for the fixed priority arbitration scheme, all other four 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. One possibility is that they all are fair arbitration schemes. Another possible explanation is the low contention probability for a random traffic in the network. 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:

[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

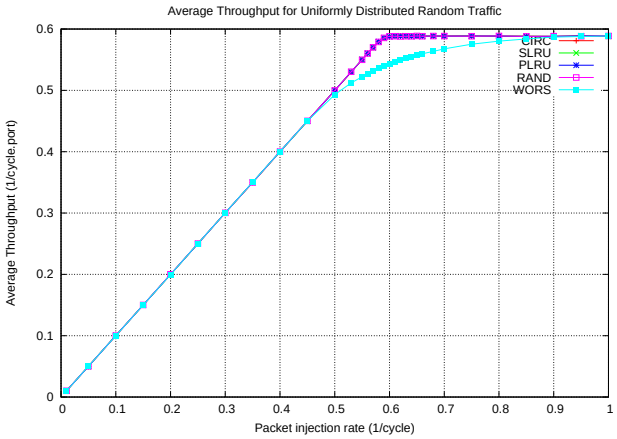

Figure 11: Throughput of Arbiter Schemes.

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

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

$$\begin{aligned} P_{contention} &= 1 - P_1 - P_0 \\ &= 1 - r\ (1 - \frac{r}{n})^{n-1} - (1 - \frac{r}{n})^n \end{aligned} \quad (3)$$

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 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 bring up questions: if the complex segmented LRU matrix scheme is really a good choice for the crossbar switch? Could a more simple and less expensive scheme be used to replace it? A decision should be made by the architecture designers.

6 Related work

High performance computer design are moving toward integration of multi-core processors on a single chip. A number of companies, including AMD, Broadcom, Freescale, PMC-Sierra, Sun, and Via, utilized technology of multicore SoC architectures [3]. Various of interconnection for the multiprocessor on-chip architectures have been proposed and implemented, such as the intra-chip switch [1], bus-based architecture [14], and crossbar-based architecture [10]. Some architecture and performance studies of interconnection networks for multi-core SoC architectures have been studied [11, 14, 10]. For example, Rekesh, et al., presented their study about the area, power, performance and designs issues for the on-chip interconnections based on a hypothetical chip multiprocessor [11].

In the context of the Bluegene/C architecture - most relevant to the C64 architecture under this discussion, there are a number of performance studies reported [15, 7, 4]. However, these studies are not specific to the performance aspects of the C64 crossbar switch network as interested in this paper.

In the emerging field of multi-core SoC architecture, the authors are not aware of any other similar studies of on-chip interconnection networks for a large-scale multiprocessor-on-a-chip architecture such as C64.

7 Conclusion

This paper presents the results on the performance evaluation of the interconnection network of the IBM Cyclops64 multicore architecture. These results show that the architecture of crossbar can deliver full pipelined bandwith and exhibit non-blocking and stable behavior under certain traffic patterns. The virtual channel mechanism can efficiently balance the traffics in either direction by sharing between the forward and backward traffics. The study shows, except the fixed prority arbitration scheme, all four others tested demonstrated very similar performance, but the segmented LRU matrix scheme achieves a great saving in memory space. The results from application-driven workloads contribute a great metric for verifying the design of the system architecture and analyzing the performance.

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

Acknowledgments

We acknowledge support from IBM, in particular, Monty Denneau for his introduction and guidance in the Cyclops-64 architecture research. We acknowledge support from several federal agencies without which this work will not be possible. We thank ETI for support of this work. We appreciate the great help from Michael Bodnar and Mark A Pellegrini, who proofread the paper and corrected its English mistakes. We thank all of CAPSL members for helpful discussions and cooperation, in particular, Juan Cuvillo, Ziang Hu, Weirong Zhu, Yuan Zhang, Joseph Manzano, Dimitrij Krepis, Yuhei Hashi, Hirofumi Sakane, Haiping Wu, etc.

References

[1] 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.

[2] Michael K. Chen, Xiao-Feng Li, Ruiqi Lian, Jason H. Lin, Lixia Liu, Tao Liu, and Roy Ju. Shangri-la: Achieving high performance from compiled network applications while enabling ease of programming. In *Proceedings of ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI05)*, Chicago, Illinoi, June 2005.

[3] Robert Cravotta. Multicore trend continues at fall processor forum. –EDN, http://www.edn.com/article/CA470045.html, Oct. 10 2004.

[4] Juan Del Cuvillo, Weirong Zhu, Ziang Hu, and Guang 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] William James Dally and Brian Towels. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[6] Juan B. del Cuvillo, Ziang Hu, Weirong Zhu, Fei Chen, and Guang R. Gao. *Toward a Software Infrastructure for the Cyclops64 Cellular Architecture*. Department of Electrical and Computer Engineering, University of Delaware, Newark, Delaware 19716, capsl techincal memo 55 edition, April 2004.

[7] Juan B. del Cuvillo, Weirong Zhu, Ziang Hu, and Guang 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.

[8] 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.

[9] Guang R. Gao, Juan del Cuvillo, Ziang Hu, Robert Klosiwicz, Clement Leung, Jason McGuiness, Hirofumi Sakane, and Ying Ping 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.

[10] Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun. A single-chip multiprocessor. *Computer*, 30(9):79–85, 1997.

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

[12] Rakesh Kumar, Victor Zyuban, and Dean M. Tusssen. Interconnections in multi-core architecture: Understanding mechanisms, over threads and scaling. In *Procedings of the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*, Madison, Wisconsin, June 4 2005.

[13] Hirofumi Sakane, Levent Yakay, Vishal Karna, Clement Leung, and Guang 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.

[14] 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.

[15] Weirong Zhu, Yanwei Niu, and Guang 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.