Server I/O Acceleration Using an Embedded Multi-core Architecture

Lurng-Kuo Liu[1]
Fei Chen[2]
Christos J. Georgiou[1]
Guang R. Gao[2]

IBM T.J. Watson Research Center[1]
Yorktown Heights, NY 10598

Computer Architecture and Parallel Systems Laboratory[2]
University of Delaware
Newark, DE 19716

CAPSL Technical Memo 68
May 12, 2006

Abstract

This paper presents a feasibility study on the use of an embedded multi-core system-on-a-chip (SoC) architecture to accelerate server I/O subsystem functions, as an alternative to implementation via finite state machines (FSMs) and hardwired logic. The multi-core solution is significantly more programmable than FSMs and avoids many of their shortcomings. For the purposes of this SoC we use the Cyclops scalable embedded multiprocessor architecture (CyclopsE) that comprises one or more processor clusters, one or more local memory banks for storing data and/or instructions, and a local interconnect implemented via a crossbar switch. The I/O functionality was implemented in C language and verified on an FPGA-based prototype that emulated a server I/O subsystem handling storage area network traffic. Results showed that for the higher-level protocols of interest, a modest number of processor cores are sufficient to handle traffic up to 8 Gb/s. Given the small silicon area and power requirements as well as good scalability of CyclopsE, much higher bandwidths can be accommodated with the use of larger numbers of cores.

1. Introduction

The evolution of the Information Technology (IT) industry has created a growing number of new business applications that require significant improvement in server technology performance. These new business applications often come with unprecedented demands for distributed storage as well. In the past few years, businesses installed significantly more storage capacity in their environments even when overall IT spending was basically flat. To effectively support these new business applications, the server processing and I/O capabilities need to be carefully balanced. Low-latency, high bandwidth, and security have become key requirements for an enhanced server I/O subsystem that handles distributed storage.

Computer systems are, in most cases, interconnected via a Fibre Channel SAN (Storage Area Network). Fibre Channel is an industry-standard protocol that is used primarily for transporting higher-level I/O protocols, for example SCSI, between servers and storage devices. The connection of the servers to the SAN is done via host bus adapters (HBAs), which are typically PCI or PCI-X bus cards. In some system configurations, PCI-based HBAs have direct memory access to the system memory. However, the PCI bus and adapters generally do not meet the high security and data integrity requirements of some high-end server systems, such as the IBM zSeries. In this case, additional functionality could be implemented within the I/O subsystem to provide error detection, isolation and handling in the transfer of data between the PCI-based HBAs and the server's system memory.

As the requirements for supporting high bandwidth I/O capability and security increase, the overhead introduced by this additional functionality may become a bottleneck in high-end server performance. This can be alleviated by increasing the processing capabilities of the server's I/O subsystem that sits between a PCI-based HBA and the server's I/O subsystem memory.

In this paper, we present a high performance system-on-a-chip (SoC) design that could be used to economically provide additional processing capacity in the zSeries I/O subsystem, and discuss a prototype that was built to accelerate specific I/O protocols, such as FICON (Fiber Connection) and Fibre Channel (FCP). The SoC is based on the CyclopsE embedded multi-core architecture that provides a scalable and easily programmable platform.

The paper is organized as follows: Section 2 discusses the system environment where function acceleration is being examined; Section 3, presents an overview of the SoC architecture and discusses the implementation of the FICON and FCP protocols; Section 4 discusses the prototype setup for simulating these protocols, and Section 5 presents simulation results.

2. System Architecture

FICON is an I/O protocol used in IBM mainframe computers and peripheral devices such as storage arrays and tape drives. Introduced in 1998, it takes the ESCON (Enterprise System CONnection) channel protocol and maps it into a Fibre Channel transport [1]. FICON now supports full-duplex data rates of 200 and 400 MB/sec and distances up to 100 kilometers.

The higher-level I/O protocol used by applications depends on the particular server architecture they are ported on. In the case of the IBM zSeries, the higher-level I/O protocol is based on channel programs that consist of multiple Channel Control Words (CCWs) [2]. A typical block diagram for FICON and FCP channel hardware is shown in Figure 1.

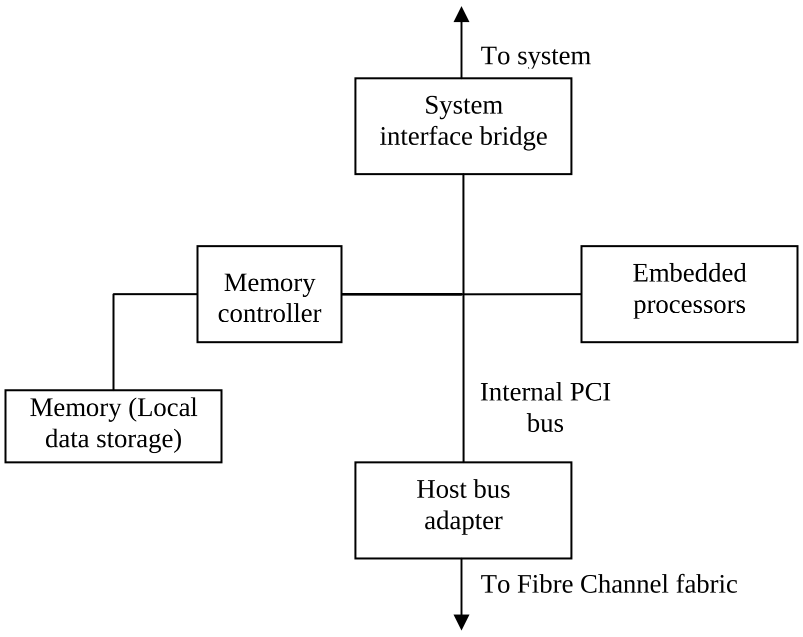

Figure 1: I/O subsystem configuration

[Gao: To help readers – figure 1 and the description below might be tightened a little more. For example, please say more about the "embedded professors" ? Where the proposed SoC will sit in this figure ?]

The system interface bridge contains the channel DMA engines, and allows the embedded processor to fetch or store small blocks of data directly from/to host memory. The HBA provides a channel interface to the Fibre Channel fabric. The HBA shares the local data storage memory that contains control structures and customer data with the embedded processor. To provide isolation of the HBA from the rest of the system as needed by the high security and data integrity requirements, all customer data transfers to/from the channel must pass through the local data storage.

The performance characteristics of our multi-core design were studied in a high-performance SoC design, as described in a later section. In our prototype, we implemented the required endpoint functionality for transferring packets between the channel logic and system interfaces for both FICON and FCP protocols. We also implemented packet header checking and validation, lost packet and out-of-order frame handling, storage interlock, local data storage addressing, and noncontiguous real memory addressing support.

The basic I/O transfer flow can be described as follows: An application constructs a channel program defining the I/O operation and initiates the I/O request by executing a start subchannel (SSCH) instruction [2]. The FICON and FCP interface protocols are command/response protocols based on queue structures. Required information for the I/O request as part of SSCH operand is placed in the queues to describe the I/O request. An HBA can access these queues to determine the work that is to be done; likewise, the I/O subsystem firmware can access these queues to determine when an I/O operation has completed.

In inbound operations, packets received from the Fibre Channel network are delivered to a speed-matching buffer through PCI write requests from the HBA. Each packet received from the link by the HBA is partitioned into multiple frames for delivering to the I/O subsystem. The I/O request is validated, at the I/O subsystem, to ensure that it complies with the I/O service it belongs to and to determine the protocol and packet context it comes from, in case context switching is needed. If needed, context is switched based on the control information obtained from memory. Various fields in the header are inspected to determine where to transfer the packet payload. Packets lost on the link as well as frames received out-of-order are also handled. Errors are isolated by not allowing subsequent data to be written in memory. I/O transfer completion status is placed into the corresponding response queue in the I/O subsystem firmware by the HBA. The firmware then checks to ensure that processing of frames is completed.

Outbound operations are simpler than inbound operations in the sense that less checking and error handling is required based on the assumption that the system bus offers higher data integrity than the SAN. Payload data are fetched either from local memory or from system memory based on the control information defined in the CCW. They are then combined with prepared header information and placed in the transmit buffer for transfer to the HBA.

3. SoC Architecture

Our SoC architecture is based on a programmable multiprocessor approach. The higher levels of VLSI integration made possible by 90nm, 65nm and smaller chip fabrication technologies have resulted in a shift towards multi-core chip implementations for the design of complex SoCs. As the cost of processor cores in terms of chip area is becoming smaller with each generation of fabrication technology, it is becoming increasingly appealing to explore the replacement of finite-state machines (FSMs) and random logic with programs running on the processor cores.

The implementation of logic functions with FSMs typically represents an optimal solution in terms of number of logic gates, power, and speed. This is a result of the custom-made nature of an FSM solution that is tailored to the specific requirements of a particular problem. However, FSMs are usually difficult to implement and verify for correctness, take a longer time-to-market and, more importantly, are not flexible enough to accommodate changes or enhancements to the original specification.

A programmable approach, on the other hand, does not have any of the above shortcomings of FSMs and, if it can meet the performance, cost and power requirements of the problem, it can be a very attractive solution. For the purposes of this SoC we use the Cyclops embedded multiprocessor architecture that comprises one or more processor clusters, one or more local memory banks for storing data and/or instructions, and a local interconnect implemented via a crossbar switch [2].

As shown in Figure 2, a single processor cluster contains 8 processor cores, a shared instruction cache, and local SRAM. The number of processors in a cluster and the size of the shared I-cache were selected to be on the design sweet-point considering silicon area, I-cache hit rate, and bandwidth between the I-cache and processors. The processors have a reduced general purpose instruction set derived from the PowerPC architecture, and a single-issue architecture with a four-stage deep pipeline. Each processor has its own register file, ALU, and instruction sequencer. The size of the instruction cache is 32 Kbytes, which is sufficient for network applications, as shown in [4]. The I-cache bandwidth to the processors is sufficient to prevent instruction starvation. As most working sets of the processor fit in the I-cache, sharing of the I-cache does not cause cache thrashing and increased instruction miss rate.

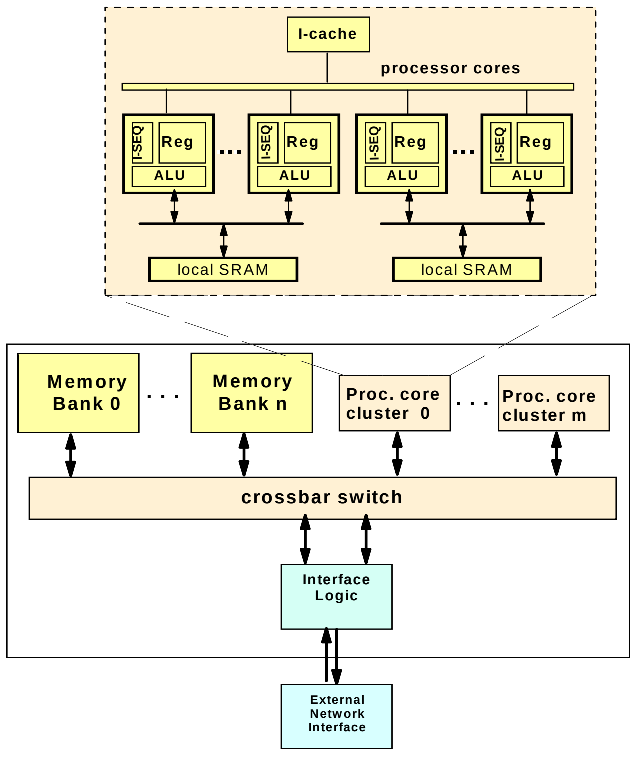

Figure 2: Embedded Cyclops multiprocessor architecture

A key consideration in our prototype implementation is the partitioning of I/O operations. Hardware resources on CyclopsE work collaboratively to handle the I/O operations initiated by applications running on the server. The basic data and process flow of an I/O operation is shown in Figure 3. The actual protocol handling and data routing are carried out by the code running in parallel on the processors contained in the multiprocessor subsystem. A processor core (i.e., dispatcher processor) handles the interface with HBA and dispatches frames to other processor cores (i.e., worker processor) based on the operational context of each frame. The dispatcher processor also validates the I/O request to ensure that it complies with the I/O service it belongs to.

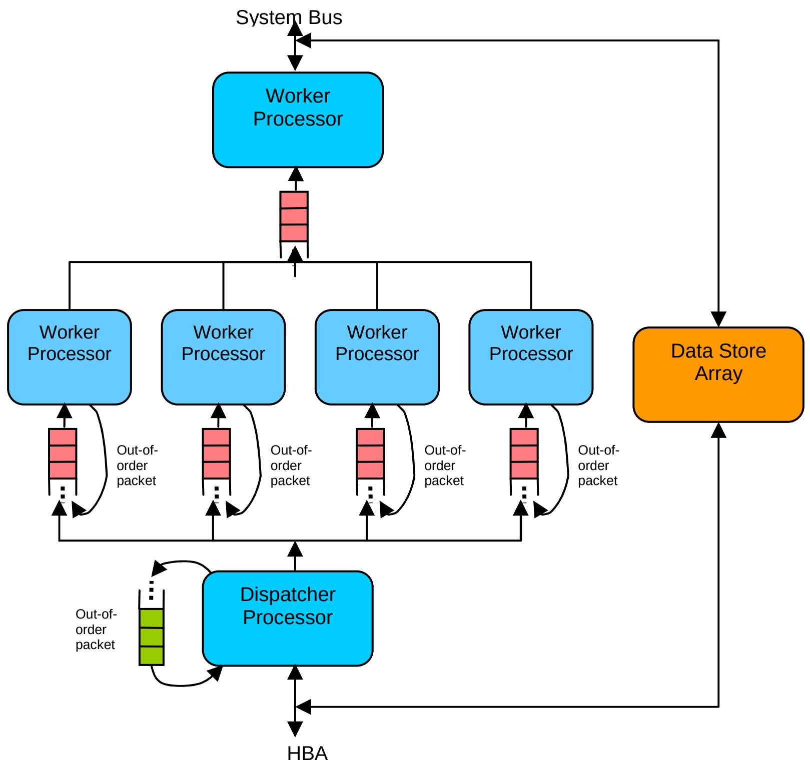

Figure 3: Basic I/O operation flow

The I/O requests generated by the dispatcher processor are placed in the request queues of their corresponding worker processors. Each worker processor then performs protocol handling, header checking, error handling and isolation, and data transfer as previously discussed. Errors are isolated by not allowing subsequent data to be written into system memory when an error occurred. When an I/O transfer is completed, its status is written in the response queue in the I/O subsystem firmware by the HBA. The firmware in the I/O subsystem will then check with CyclopsE to ensure that the processing of frames is completed.

The number of processor cores required for the transfer of data is based on the target network bandwidth and packet size used. Smaller packet sizes significantly increase the overhead in processing header information and thus would require more processor cores. The same applies for the network bandwidth. More processor cores would be required in more demanding network bandwidth environments.

4. Simulation environment

The experiment environment for the data router design is based on an iterative emulation of the CyclopsE embedded multiprocessor architecture – DIMES [5]. Directly mapping multi-core system logic into a FPGA chip proves to be impossible, because the target logic size well exceeds the capacity of even the largest FPGA chips existing in the market today. The iterative emulation technique exploits the repetition of identical logic elements that exists in the multi-core chip design by physically instantiating only one copy of the combinatorial logic element in the FPGA chip, while emulating multiple sequential logic instances with shift registers and/or RAM blocks in a time-sharing fashion. As shown in Figure 4, the DIMES system is implemented upon an off-the-shelf Alpha-Data ADM-XRC-II board with one Xilinx XC2V8000 FPGA chip. The host PC runs Linux and communicates with the emulated multi-core chip logic on the FPGA via the PCI bus. The current DIMES system can accommodate the target logic of 8 processors, on-chip memory banks, and the interconnection network of the proposed multiprocessor discussed earlier.

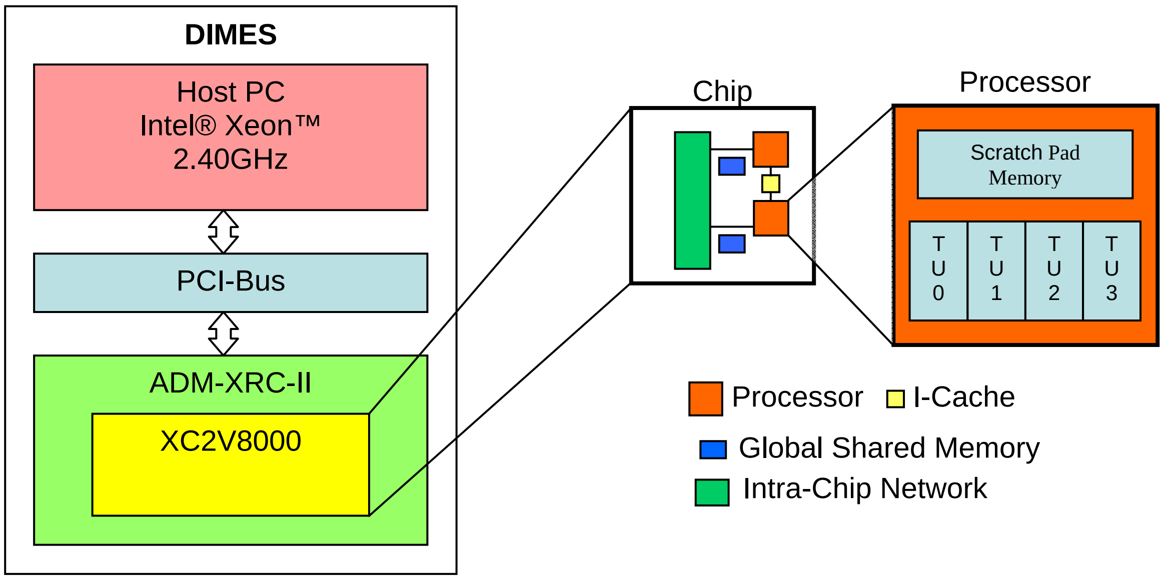

Figure 4: DIMES system setup

The simulation software is composed of three major modules shown in Figure 5. HBASim is a program that simulates a PCI-X master device and injects data frames into the I/O Accelerator. zSSim is another program that simulates a PCI-X slave device which acts as a data sink for data received from the I/O Accelerator. The I/O Accelerator simulation system receives data from HBASim as a PCI-X slave in one side and sends data to zSSim as a PCI-X master in the other side. The PCI-X interfaces are simulated by socket connections in our system, so that raw data frames can be moved from HBASim to the I/O Accelerator, and the processed data frames can be further delivered from CyclopsE to zSSim. While HBASim and zSSim are both stand-alone processes running within the host Linux system, the I/O Accelerator multi-core simulator is actually composed of two parts: the host-side module running as a Linux process and the target-side module running upon the emulated Cyclops embedded multi-core system in the Xilinx FPGA. The host-side module loads target application to the FPGA and handles socket communications with HBASim and zSSim, while the target-side module application actually performs the I/O Accelerator functionalities, such as packet header checking, protocol processing, and so on. Using the PCI device driver provided by the FPGA board vendor, information exchange between host-side and target-side modules is done by mapping part of the target-side module's physical memory into the host-side module's program virtual memory space.

Depending on the configuration, up to 8 physical processor cores can be deployed by the target-side module in the master-slaves mode. The host-side module application receives raw data frames from HBASim and puts them in the input data buffer of the target-side module. The dispatcher processor of the target-side module checks frames in the input data buffer and dispatches those frames to the worker processors for protocol processing, and tries to balance the loads of different worker processors at the same time. Once a frame has been processed by a worker processor, it is stored in its output data buffer, which is also visible by the host-side module. The host-side module is also responsible to check the output buffers of all worker processors and to send the processed frames out to zSSim, if any.

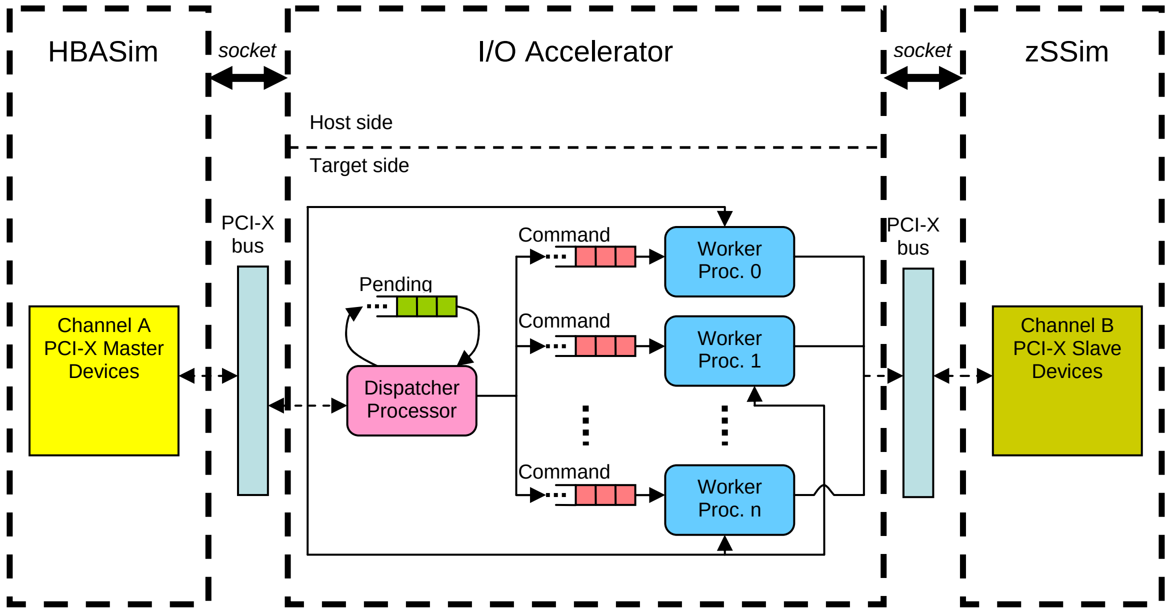

Figure 5: Test software structure

5. Results

Our approach for server I/O acceleration is based on CyclopsE which, as previously described, is a self-contained architecture with its own memory and local interconnect. The number of processors and memory size are design parameters that depend on the target protocols to be implemented. It is our intention to provide a guideline for the SoC designer to determine these performance parameters through our experiment results shown in this section.

We conducted a set of experiments to determine the number of processors that would be needed to perform the transfer of FICON and FCP packets at various data rates and packet sizes. Our implementation of the I/O Accelerator for FICON and FCP was developed, parallelized, and implemented in C language. The I/O Accelerator program was then compiled with the CyclopE compiler and ran on the simulation environment, as described above. We have constructed test cases based on actual FICON and FCP trace data and collected statistics from the simulation runs of these test cases.

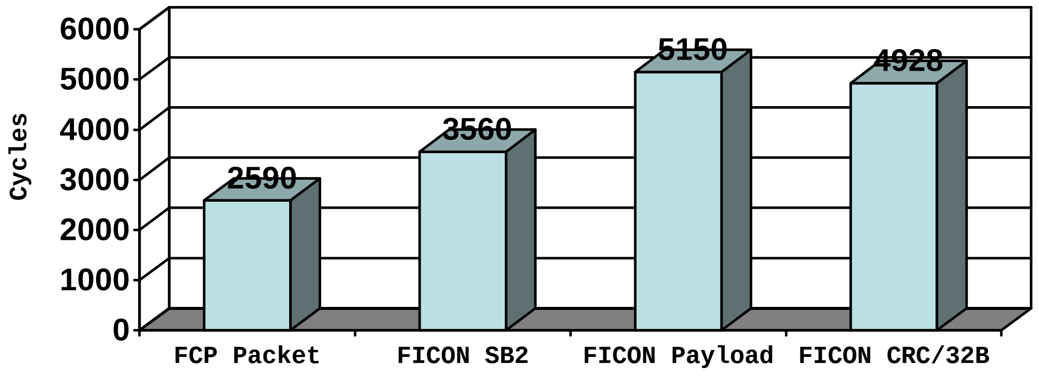

Figure 6: FICON/FCP functionality performance profiling on CyclopsE

[Gao: please check figure 6 and the following descrition to make them matching each other. For example: the name FICON SB2, FICON Payload and FICON CRC/32B should be clearly 1-1 explained. Then, the numbers on top each bar should also match the numbers listed in the text below – e.g. 5,180ns and 17,420ns – where we can find these numbers in the fiture ?
].
Figure 6 shows the cycles spent on some major protocol functions. FICON is more demanding in computation requirements than FCP due to its complexity in protocol and error handling. CRC computation in FICON took 4,928 cycles for every 32 bytes of payload data. This is expensive and should be offloaded to dedicated hardware. In our analysis, assuming CRC computation is offloaded to hardware, on average of 2,590 cycles are needed for the processing of each FCP packet and 8,710 cycles are needed for each FICON packet. The latter involves memory accesses to resolve indirect addressing (3,560 cycles, shown as FICON SB2, in Figure 6). With CyclopE operating at 500 MHz, it means that it will take 5,180ns and 17,420ns to process a FCP packet and FICON packet, respectively.

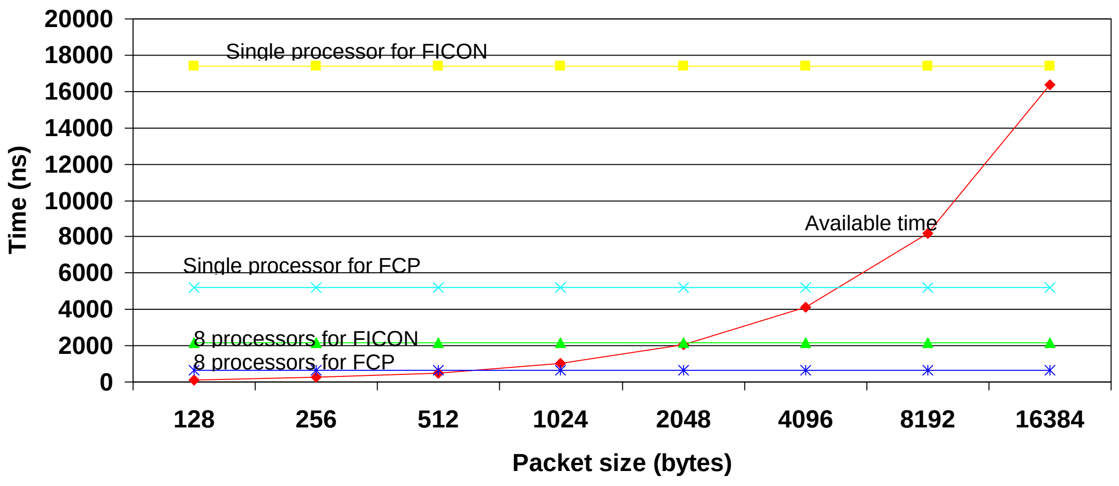

Figure 7: Processing requirements as a function of packet size

[Gao: some of the colors in Figure 7 is hard to read. Also, I cannot identify/comprehend easily the two data points mentioned the end of the following paragraph. Is there a way to improve this illustration ?]

The time available for processing a packet is a function of packet size and network speed. The larger the packet size the more time is ~~available~~ required for the data router to process the data routing information. Figure 7 shows the minimum packet size requirements for FCP and FICON data routing, assuming the network bandwidth is 8 Gb/s. The results show that 8 CyclopsE processors operating at 500 MHz can handle FICON traffic at 8 Gb/s, for the 2Kbyte Fibre Channel frame sizes currently being used. For FCP, 8 processors can handle frame sizes as small as 1,024 bytes. Configurations with more processors may be used for higher line speeds or smaller packet sizes.

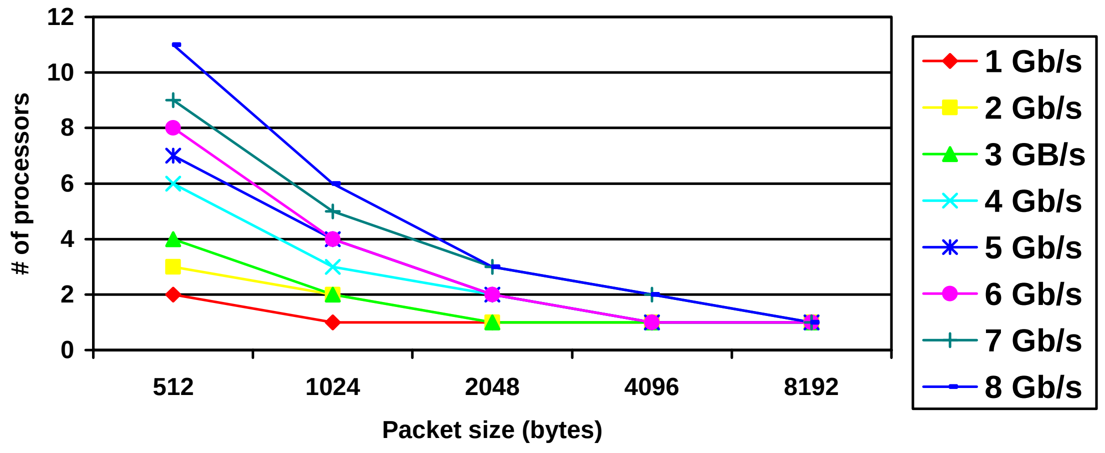

Figure 8: SoC hardware requirements for handling FCP

Figure 8 shows the SoC hardware requirements for handling FCP and Figure 9 shows the SoC hardware requirements for handling FICON. The number of processors needed to handle traffic at a given speed decreases with an increase in packet size. This is because the time required for packet header processing/checking is basically independent of packet size. Larger packet sizes will result in a smaller packet incoming rate and thus would require fewer CyclopsE processors.

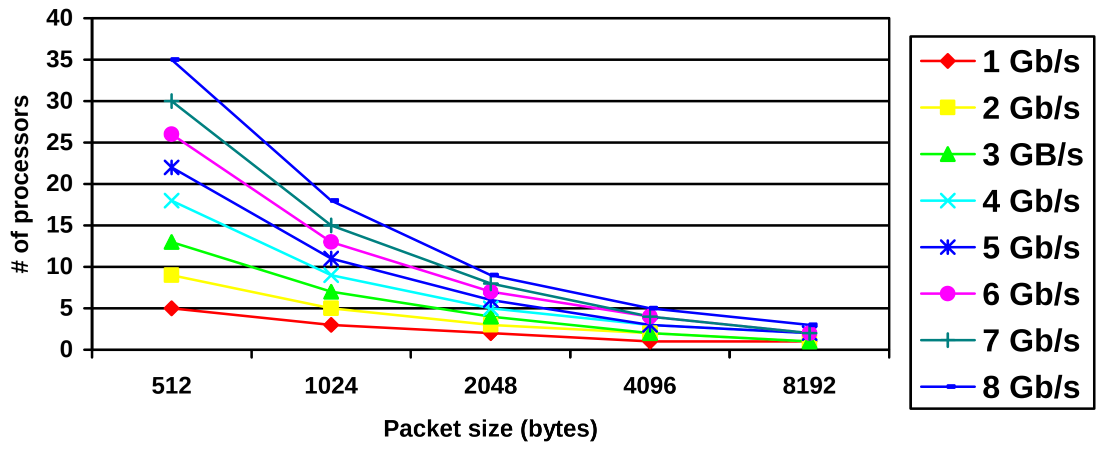

Figure 9: SoC hardware requirements for handling FICON

The CyclopsE architecture is both scalable and silicon-area efficient, providing a lot of flexibility in the design of SoCs, as it allows a wide range in the number of integrated processor cores. We have estimated that a basic 8-core cluster that contains 32 Kbytes of local SRAM and 32Kbytes of I-cache occupies 9 mm^2 in 90nm CMOS technology. The area of larger configurations increases almost linearly with the number of integrated cores.

6. Conclusions

In this paper, we have investigated the acceleration of server I/O functionality using the CyclopsE embedded multiprocessor architecture. The processing functions involved in the FICON and Fibre Channel protocols were parallelized, written in C, and simulated on the DIMES FPGA simulator. The results showed that a modest number of processor cores are sufficient to implement the required functionality for next generation storage network speeds. This makes the embedded multiprocessor implementation an attractive solution, considering the ease of design and flexibility that a programmable solution offers. Given the advantage of silicon area, power requirements, and performance scalability of a multi-core architecture such as CyclopsE, much higher bandwidths can be accommodated with the use of larger numbers of cores. Future work could involve the prototyping of more complex protocols such as TCP/IP or iSCSI.

Acknowledgments

The authors would like to thank John R. Flanagan for his technical insights and support of our prototyping activity. The authors also would like to thank our sponsors from various government agencies and CAPSL group members in the University of Delaware for their support.

References

[1] DeCusatis, C.M., Stigliani, Jr., D.J., Mostowy, W.L., Lewis, M.E., Petersen, D.B., Dhondy, N.R., Fiber Optic Interconnects for the IBM S/390 Parallel Enterprise Server G5. *IBM Journal of Research and Development*, vol. 43, no. 5/6, September/November 1999, pp. 807-828

[2] Stigliani, Jr., D.J., Bubb, T.E., Casper, D.F., Chin, J.H., Glassen, S.G., Hoke, J.M., Minassian, V.A., Quick, J.H., Whitehead, C.H. IBM eServer z900 I/O subsystem, *IBM Journal of Research and Development*, vol. 46, no. 4/5, July/September 2002, pp. 421-446.

[3] Georgiou, C.J., Salapura, V., Denneau, M., A Programmable Scalable Platform for Next Generation Networking. *Network Processor Design, Issues and Practices*, vol. 2, Chapter 2, Morgan Kaufmann Publishers, San Francisco, California, 2004, pp. 9-28

[4] Wolf, T., and Franklin, M.A., Design Tradeoffs for Embedded Network Processors. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS)* (Lecture Notes in Computer Science), vol. 2299, pp. 149-164, Karlsruhe, Germany, April 2002. Springer Verlag

[5] Hirofumi Sakane, Levent Yakay, Vishal Karna, Clement Leung, Guang R. Gao, DIMES: An Iterative Emulation Platform for Multiprocessor-System-On-Chip Designs, *International Conference on Field-Programmable Technology, FPT'03*, University of Tokyo, Japan, Dec. 15-17, 2003.