計算機アーキテクチャ 99－18
(1993. 3. 12)

An Overview of Asura's Network with Simulation Results

David FRASER and Takashi TANAKA

Office of Computer Business
Kubota Corporation
2-47, Shikitsuhigashi 1-chome
Naniwa-ku, Osaka, 556-91

E-mail: {david, takashi}@kocb.astem.or.jp

Asura's Inter Cluster Network connects its clusters into a closely coupled system. In order to reduce latency, the ICN is wide enough that commands can be completed in one cycle and multiple transmissions can occur in parallel. The bandwidth of the ICN scales as more clusters are added to the system with each link having a bandwidth of 400 Mbytes per second. The ICN uses general point-to-point links that allow many different network topologies to be used with Asura. This paper describes the ICN and its Network Interface. Simulation results show that the ICN, together with Asura's global cache can effectively reduce the average latency of memory accesses as seen by the CPU.

Keywords: Parallel, network, bandwidth,
network latency, memory hierarchy, cache hierarchy

ＡＳＵＲＡのネットワーク構造とそのシミュレーション

David FRASER、田中高士

（株）クボタ
コンピュータ事業推進室
〒５５６－９１
大阪市浪速区敷津東１丁目２番４７号

ＡＳＵＲＡのインター・クラスター・ネットワーク（ＩＣＮ）は複数のＡＳＵＲＡクラスターを密に結合する。ネットワーク遅延軽減のために、ＩＣＮは通信コマンドを１サイクルで実行し、かつ、複数のデータ転送を並列に実行するように設計されている。また、ＩＣＮの各リンクは 400MB/s と高速で、クラスタ台数の増加に応じた台数効果をもたらすに十分なバンド幅を提供する。ＩＣＮは任意のノード間の通信を行なえるため、ＡＳＵＲＡ上での様々なアプリケーションで使用される、さまざまな通信トポロジーを容易に実現できる。本稿では、ＩＣＮとそのネットワーク・インタフェイスについての概説を行なう。さらに、シミュレーションにより、ＩＣＮの平均遅延時間と、ＡＳＵＲＡのグローバル・キャッシュ・ヒット時の性能向上についても報告する。

キーワード：　並列、ネットワーク、バンド幅、
ネットワーク遅延時間、階層メモリ、階層キャッシュ。

1 Introduction

Over the past decade many different architectures have been proposed for large-scale multiprocessor computers. Some early machines have been shared memory machines, either based on a bus architecture such as the Alliant, used in the Cedar project [KUC86], or based on an expensive multistage interconnection network such as the BBN Butterfly [BRO87]. Other machines, such as the Intel iPSC hypercube [ARL88] or a cluster of workstations, use distributed memory and communicate by message passing. Recently the two types have been combined into distributed shared memory multiprocessors such as DASH [LEN92], the KSR1, and SESAME [WIT92]. This class of computers is based on processing nodes, each with its own memory, thus the memory is physically distributed and the network bandwidth and resources scale as more nodes are added. However, each processor in the system sees the memory as a single block and can access it as shared memory.

Asura is the name of the large-scale, distributed shared memory, scalable multiprocessor architecture that is presently under research at Kubota Corporation. The project aims to develop a new computer architecture that includes an operating system, a parallel compiler, and a hardware prototype. Figure 1 shows an overview of Asura's architecture.

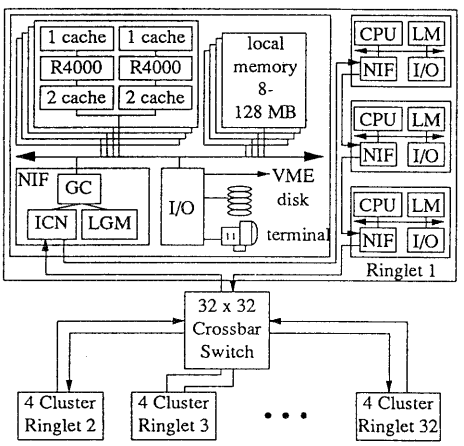

Figure 1: An Overview of the Asura System

Asura can be configured with from 2 to 128 clusters. For the prototype, each cluster is based on a Titan computer [NAI92] which contains four CPU boards with 2 processors per board, memory boards, an NIF (Network Interface) board, and an I/O board. A graphics board can also be included. Each CPU board contains two Mips R4000 microprocessors, each with it's own primary and secondary cache. Each memory board can contain up to 128 Mbytes of memory and is used as private local memory by the system. The NIF board contains a 32 Mbyte global cache, 16 Mbytes of global memory, and an interface to the ICN (Inter Cluster Network), used to connect the clusters. The global cache serves to hide the latency of accesses to global memory in remote clusters and to isolate the bus protocol of the cluster from the ICN protocol. This is a key feature of Asura since it allows us to optimize the intra cluster network (for the prototype machine, this is the Titan bus) for fine grained parallel processing and to optimize the ICN network for coarse grained parallel processing. Asura's global cache uses a version of the *Synopse* protocol [ARC86]. Global data can be in any of three states: *dirty*, *shared*, and *invalid/uncached*. The global memory on the NIF board is referred to as that cluster's local global memory. Logically this is identical to the global memory located in other clusters (remote global memory) but since accesses to it don't pass through the ICN they are faster (see [GLE92]). This fact can be used by the software to place data to be used by a cluster in the cluster's local global memory to optimize performance.

Section 2 gives an overview of the ICN. Section 3 examines the performance of the ICN based on the Verilog model of the NIF board. The effect of varying some of the ICN parameters is discussed in section 4 and some concluding remarks are included in section 5.

2 An Overview of the Inter Cluster Network

It is the goal of the ICN to provide a high bandwidth, low latency, scalable network to connect Asura's clusters into a tightly coupled system. In order to do that it is necessary to ensure that the latency of accesses through the network do not keep the CPUs waiting idle for too long a time. The ICN is based on a *register insertion ring* [TAN81] network. Basically each node has an input buffer and an output buffer and it may transmit whenever it's input buffer is empty. This scheme is very simple and it allows us to implement a very fast network interface. In the case of accesses to line locking variables and synchronization variables latency must be kept to a minimum and hotspots should be minimized. The ICN is sixteen bytes wide and all synchronization and locking commands require eight bytes therefore these commands do not acquire any added latency due to their length and in fact two commands can be sent in the same clock cycle as long as they are not destined for the same cluster. Since most traffic in the ICN will be the result of cache coherency operations the network must be able to efficiently transfer cache lines. The global cache uses a line size of 1 Kbyte. This line size was chosen after simulations showed that the

large global cache and the use of software control to prefetch and write back cache lines can effectively hide the extra latency needed to load a large cache line. As well, since eight CPUs use the same global cache a line fetched in response to an access from one CPU will also implicitly prefetch data needed by other CPU's in the same cluster.

The ICN specification does not require any specific network topology. Instead the interface to the ICN consists of one input link and one output link. For the first prototypes of Asura a ring topology will be used to connect up to eight clusters in a simple, low cost network. For those applications that demand more processing power and therefore more clusters a crossbar switch will be developed to connect directly to clusters or to connect ringlets of four clusters each. The full, 128 cluster version of Asura will consist of thirty-two, four cluster ringlets connected by a crossbar switch. This means that at the most, a message will have to pass through eight clusters and the crossbar switch twice to access any cluster in the system. Figure 2,a shows a four cluster version of Asura. Figure 2,b shows the full 128 cluster topology.

The ICN is 160 bits wide [1] and is further split up into two, eighty bit wide subnets. It is clocked at 25 MHz. Each subnet is then further divided into eight bytes (72 bits) for command and data transfer and one byte for implementing the network's bandwidth allocation and priority mechanism. In this paper only the command and data transfer portion will be dealt with. The bandwidth allocation and priority mechanism was adapted from the IEEE SCI (Scalable Coherent Interface) standard and is described in [IEEE91]. When used to issue commands and acknowledgements the following information is transmitted; the type of command (read, write, coherent, invalidation, acknowledgement, etc), the amount of data (none, 1, 2, 3, 4, 8, or 1024 bytes), the issuing and receiving unit (cluster, global cache, global memory), 39 address bits, network priority information (see[IEEE91]), and parity bits.

A wide variety of commands are supported on the ICN. *Coherent* commands read or write 1 Kbyte blocks of data. If the command is *exclusive* then the data will be loaded into the global cache in the *dirty* state, otherwise it will be loaded *shared*. *Non-coherent* commands transfer 1, 2, 3, 4, or 8 bytes of data that have been marked uncacheable. An *invalidate* operation is provided to invalidate global cache lines. Special commands are also provided for synchronization purposes (see [SAI92]).

The ICN uses a split protocol. Every command is acknowledged, even if no data is returned. This in necessary to determine when a synchronization command has completed globally. The format of acknowledgements is similar to commands except that an *ack* bit is set.

[1] In future versions it will be possible to transmit the 160 bits on four parallel fiber optic links but this is not included in the first prototype.

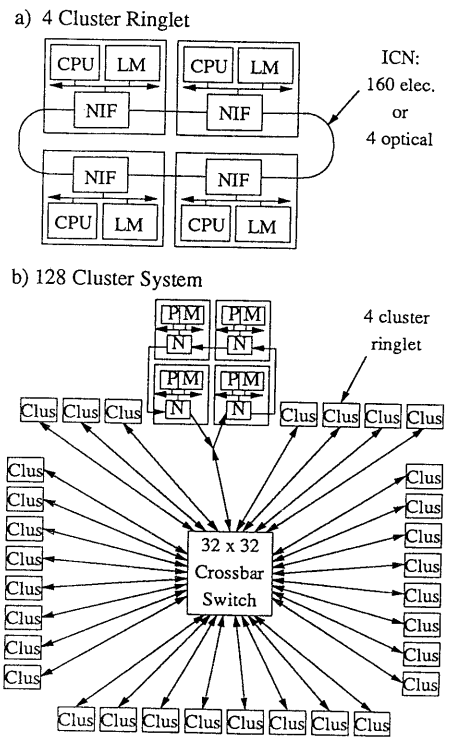

Figure 2: Examples of ICN Topologies that can be used with Asura

For non-coherent commands involving data the data is sent in the same clock cycle as the command but on the other subnet. In the case of block transfers the first cycle contains the command with the property information for the line, which describes what type of coherence control protocol to use with the line [MOR92], on the other subnet. The 1 Kbyte line is then transferred in the next 64 consecutive clock cycles.

The ICN's network interface is contained on the NIF board located in each of Asura's clusters. A block diagram of the NIF board is shown in figure 3. It can be seen that the NIF board consists of six main modules. The global memory contains 16 Mbytes of global memory, the coherence directory, support for line locking and COLB [GOO91] based barrier synchronization. The global cache is a 32 Mbyte shared cache for the eight CPUs in the cluster. It is non-blocking in that up to eight accesses to the cache can *miss* before other accesses to the cache are blocked. This ensures that a miss by one CPU's access will not block the other seven. The communication controller provides a non-blocking path between the previous

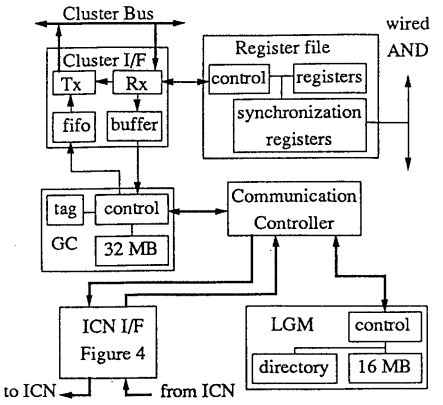

Figure 3: Block Diagram of the NIF board

three modules and is based on a crossbar switch architecture. It offers a bi-directional link to both global memory and the global cache and a uni-directional link both to and from the ICN interface module. The cluster interface connects the global cache to the cluster's bus. Wire based synchronization and status registers are contained in the register file. The ICN interface sends and receives data from the ICN and is described in the next paragraph.

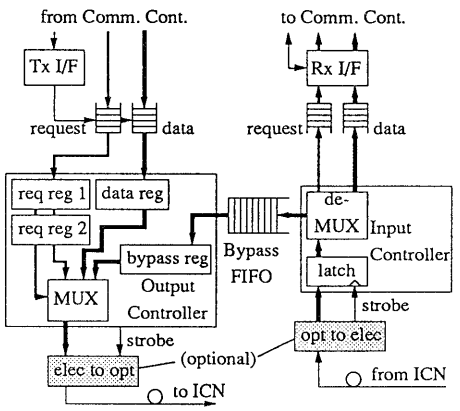

Figure 4: Block Diagram of the ICN Interface

The ICN interface is shown in figure 4. It consists of and *input controller*, an *output controller*, a *receiver interface*, a *transmit interface*, a *bypass fifo*, a *receive fifo*, and a *transmit fifo*. Data is latched in from the ICN and is decoded. A demultiplexer is then set up to route the data. If it is addressed to the cluster, it is written into the proper receive fifo. There are separate fifos for data, and command and acknowledgement information. When the receive interface (Rx I/F in figure 4) detects that the command and acknowledgement fifo is not empty it reads a command and starts to arbitrate for the communication controller. If necessary data is read, and it is sent together with the command to either the global memory or the global cache. The receive interface is pipelined so that it can send a new command or acknowledgement to the communication controller every clock cycle. Data that is not addressed to the cluster is written to the bypass fifo. When the communication controller has information to be sent on the ICN it writes the information into the transmit fifo with the help of the transmit interface. The output controller is constantly monitoring the transmit fifo and the bypass fifo. If there is anything in the bypass fifo it will be transmitted immediately. If the bypass fifo is empty then a command will be sent from the transmit fifo. The output controller is pipelined so as to be able to send a new command every ICN clock cycle. The output controller will also place two commands in each cycle if it is possible.

3 Performance Analysis of the ICN

The final judge of Asura's memory architecture will be the time that it takes to run applications on it. In the case of the ICN this will depend to a large extent on how well it can hide the latency of accesses to memory that are either uncachable, such as line locking or synchronization commands, or miss in the global cache. There are two components that must be examined to determine the performance of Asura's ICN: The latency due to the cluster and the latency due to the ICN. The latency due to the cluster is known since both the actual hardware and a hardware description model written in Verilog of the Titan exist. For the ICN a Verilog behavioral model was written and simulations performed to determine its performance. The factors that we are interested in are the bandwidth between the CPU and the different levels of memory and the latency of accesses. Of particular interest is the time to get data to the CPU in the case of a cache miss. Unless otherwise stated, all of the results are based on a four cluster system connected in a ring topology. The following abbreviations will be used: *CPU* for the R4000 microprocessor, *1-c* for the on-chip primary cache, *2-c* for the secondary cache, *LM* for the local memory in each cluster, *GC* for the global cache, *LGM* for the global memory located in the same cluster as the CPU, and *RGM* for global memory located in another cluster.

B/w and	CPU 1-c	1-c 2-c	2-c LM/GC	GC LGM	GC RGM
Clock(MHz)	100	50	16	25	25
Width(B)	4	16	8	16	16
BW(MB/s)	400	800	128	400	400

Table 1: A Summary of the Bandwidth between the Different Levels of the Memory Hierarchy

	MTrans/s	MB/s
Cluster Bandwidth		
Coherent Read	3.94	126
ICN Bandwidth		
Word reads per link	24.60	98.4
Block Reads per link	0.38	387.5

Table 2: Cluster and ICN bandwidth

3.1 Bandwidth

Bandwidth is the capacity of the network to move data between different levels of the memory hierarchy. Table 1 sums up the bandwidth between the different levels of the Asura hierarchy. The entries in the table show peak performance in Mbytes per second.

Since each cluster is based on a Titan computer the specifications of the cluster bus cannot be easily changed and the ICN should be designed to match the cluster bus. In order to see how well the present ICN design matches the cluster bus the bandwidth of the cluster bus will be compared to the bandwidth of the ICN for Asura system configurations of different numbers of clusters. The case of coherent (block) reads to RGM will be used as an example since the command will travel both the cluster bus and the ICN.

Table 2 shows the actual system bandwidth for various commands. The cluster bus in eight bytes wide and clocked by a 16 MHz clock. This gives a peak bandwidth of 128 MB/s. In the case of a read, for each word of data returned, a read request must be issued. However, the cluster bus has separate *address/data* cycles and *data return* cycles that can be overlapped so that there is no loss of bandwidth. As well, the memory board spends 1.6% of its time in DRAM refresh cycles so the total bandwidth is 126 MB/s. This is the same whether or not the read is local or remote since the cluster bus uses a split transaction protocol and for a normal load from RGM there is no transaction generated on the cluster bus of the remote cluster. The *word read* value for the ICN assumes one word being transferred per cycle. The block read assumes one cycle for property information followed by 64, 16 byte transfers to transfer a 1 KB line. The results are summarized in figure 5. It can be seen that for all cases that the bandwidth of the ICN exceeds that of the cluster bus. Therefore it may be possible to decrease the performance of the ICN to match its bandwidth to that of the cluster bus but this would adversely affect latency as is discussed in section 4

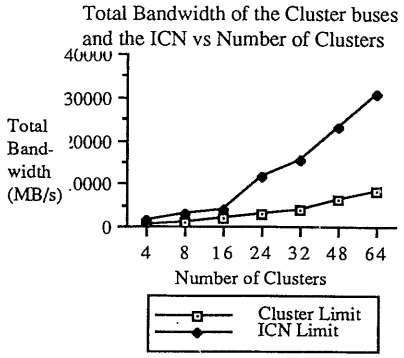

Figure 5:

3.2 Latency

Latency is the time that a CPU must wait from the time that it issues a request until, in the case of a read, it receives the data, or in the case of a write, it may proceed with the next request. Asura's CPUs are Mips R4000s which use an internal 100 MHz clock. The high speed of the clock means that the memory architecture is much more important than in a system with a slow CPU since the same delay in seconds will mean many more idle cycles in the system with the fast clock than in the slow one.

In the latency measurements to follow only the case of a cache refill is examined. This is an important case since a CPU always stalls on a read miss and will be idle until the data is fetched. Therefore the values used indicate the number of CPU (100 MHz) clock cycles between when the CPU issues the request to when it actually receives the data. In all the graphs in this section the following abbreviations are used: *LM* and *GC* are the latency involved in reading 32 bytes from local memory and the global cache. *LGM*, and the various *RGM* accesses involve first loading a new 1 Kbyte line from global memory into the global cache and then supplying a 32 B line to the secondary cache. *S* and *d* mean that the line is already loaded in the *shared* or *dirty* state in another global cache. *L* and *r* indicate that the other cache is in the same (local) cluster, or in another (remote) cluster, as the global memory.

Figure 6 shows the latency of coherent reads when the access misses at various levels of the memory hierarchy. The results are further divided by what part of the memory hierarchy is causing the delay. Though accesses to the global cache are supposed to complete in the same amount of time as those to local memory it can be seen that, even though it takes less time to access the global cache than the local memory, the cluster interface adds a delay of 78 clock cycles. This

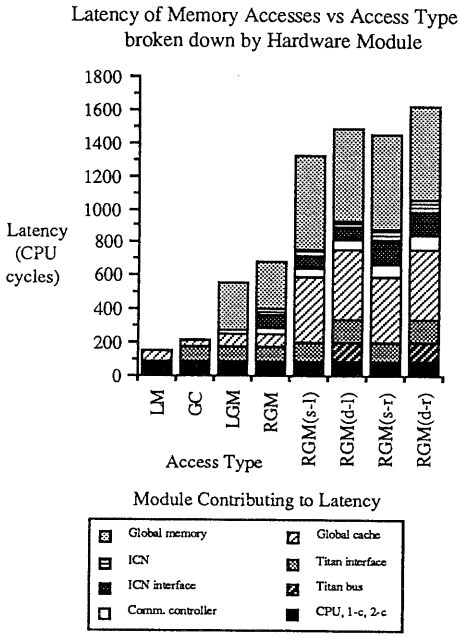

Figure 6:

is because the cluster interface buffers requests from the cluster bus while the global cache is busy dealing with the global memory or the ICN interface (See figure 3). The next point to notice is the delay due to global memory (and the global cache when it must write a line back to global memory). This latency is due almost entirely to the size of the global cache line as it takes about 280 cycles to read a 1 Kbyte line from global memory. The ICN interface adds 72 cycles of latency for each transaction and 18 cycles for each cluster that is bypassed in the ring. Therefore in the full Asura configuration of 128 clusters the total delay due to the ICN will be $72 + (18 \times 6) = 180$ cycles plus the delay of the crossbar which will be at least $72 \times 2 = 144$ cycles. This means that the delay caused by adding eight clusters to a ring network is the same as that caused by using a crossbar switch. In other words, for less than eight clusters, a ring topology will provide faster response than a crossbar topology.

Another point to remember is that in a well behaved program that there will be good locality of references. Therefore if it is necessary to fetch a 1 Kbyte line from global memory it should implicitly prefetch several 32 B intra cluster cache lines. In table 3 the best and worst case latencies to retrieve one word are shown. The best case values assume that if four words must be fetched from local memory to supply the CPU with one word that the next three accesses will hit in the

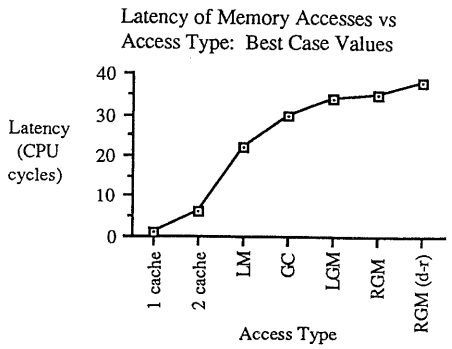

Figure 7:

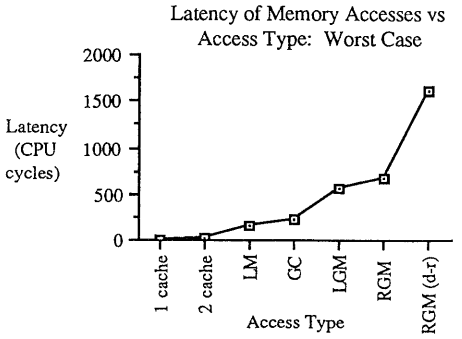

Figure 8:

primary cache or that if 1 Kbyte of data must be fetched from global memory then the next 31 $((1024/32) - 1)$ misses to that line will hit in the global cache. The worst case values assume a miss on each access.

It can be seen in figure 8 that while the latency for the worst case, indicating that every access misses, rises sharply if there is a miss in the global cache due to the time required to read a 1 Kbyte line from memory. However for the best case it can be seen in figure 7 that though the average latency for accesses that miss in the global cache is not that much higher than for accesses that hit in the global cache. This is because of the prefetching effect of the large global cache line. In the best case, for each miss in the global cache the next 31 accesses will hit. Also it is seen that a limit is reached that is below 40 CPU cycles per word.

4 Discussion

From the results of the simulations the difficulty of keeping Asura's CPUs supplied with data can be seen. In this section the latency added by each

Operation	Best Case		Worst Case	
Read from	MB/s	Clk	MB/s	Clk
1-c	400.0	1	400.0	1
2-c	69.6	6	20.0	20
LM	18.2	22	2.7	150
GC	13.1	30	1.8	218
LGM	12.6	32	0.7	556
RGM	12.4	32	0.6	684
RGM(d-r)	11.1	36	0.2	1618

Table 3: Average Time per Access on a Cache Miss

module will be discussed and improvements suggested.

A miss in the primary cache will cause the CPU to sit idle for the 20 cycles that it takes to fetch a cache line from the secondary cache. In the case of a miss in the secondary cache the overhead of the CPU board is 92 clock cycles. In designing the ICN and NIF board this is the base from which the design started from.

The cluster I/F contributes 78 clock cycles of delay to send and receive a 32 byte line which amounts to 13, 16 MHz cycles. Of these, 9 of the cycles are needed to deal with the 4 read requests and 5 data return cycles. The other 4 cycles include arbitrating for the cluster bus and the delay of the buffer between the cluster interface and the global cache. By coupling the design of the cluster interface and the global cache more closely it should be possible to overlap these 4 cycles with the activity of reading the cache line. This would leave a delay of 54 CPU cycles.

The latency due to the global cache and the global memory is primarily due to the large global cache line size. The global cache contributes approximately 25% of the total latency and the global memory 40%. To decrease the line size would shorten the delay but as seen in table 3 much of this delay can be hidden if there is a high locality of reference among data accesses. Decreasing the line size would actually increase the average latency as seen by the CPU for applications that exhibited behavior close to the ideal case as discussed below.

The ICN adds approximately 6% to 16% to the delay or 108 to 216 CPU cycles. The bright side of this is that for the full sized system this reaches a limit of about 330 cycles due to the use of the crossbar switch. Unfortunately, perhaps the only way to reduce this latency is to increase the clock speed of the ICN which is limited by the the speed of commercial FIFO integrated circuits.

4.1 The Affect of Varying the ICN Parameters on Latency

In this section the effect on latency of varying the size of the global line size and the width of the ICN are examined. The global line size is of interest because, as shown in table 3, for applications that do not follow the *best* case latency would be very high for accesses that miss in the

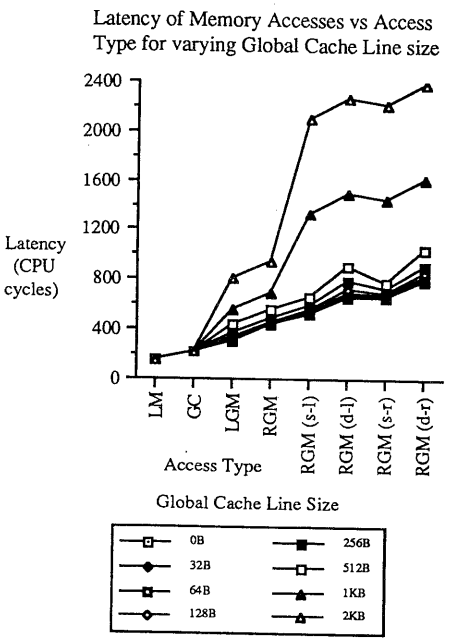

Figure 9:

global cache and that 53% to 72% of the latency is due to the cache line. Figure 9 shows the affect of varying the line size on the latency. It can clearly be seen that there is a large jump in latency when the cache line size changes from 512 bytes to 1 Kbyte. This would imply that it may be advantageous to halve the global cache line size.

The width of the ICN is of interest more from an economical point of view. The 16 byte width of the ICN makes it difficult to build the ICN interface due to the large number of integrated circuit pins needed. In fact, the ICN interface will probably have to be made in a *bit-slice* fashion. Reducing the width of the ICN would increase latency and reduce bandwidth but would greatly reduce costs and increase ease of manufacturing. Figure 10 shows the effect on latency of the width of the ICN. The graph only shows the effect for block transfers and since the transfer time is dominated by the line size it is apparent that a high ICN bandwidth is needed to transfer a large line in a reasonable amount of time. If it proves reasonable to decrease the line size and/or increase the clock speed then it may prove possible to decrease the line width and therefore gain the benefits of a narrow network. In the case of synchronization and command messages, messages are only 8 bytes wide so they are little affected by the width of the ICN.

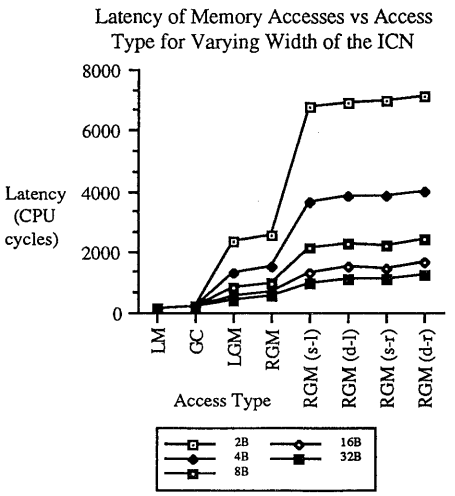

Figure 10:

5 Conclusion

Asura's NIF board implements its ICN and interfaces it to its cluster. The ICN runs at 25 MHz and is 16 bytes wide. The bandwidth of the ICN is 400 Mbytes per second and provides slightly higher bandwidth than the combined intra cluster busses of the cluster. The latency of accesses is dominated by the time it takes to read the large, 1 Kbyte global cache line. However, for programs that exhibit good locality of reference, much of this delay can be hidden, due to the implicit prefetching effect of the large line. By decreasing the line size the latency can be greatly reduced but for well behaved programs this could actually increase the overall latency of accesses. In order to decrease the cost of manufacturing and make the hardware smaller the width of the ICN could be decreased but this would have to be coupled with some adjustment to make up for the increase in latency. This paper presented the results of the first prototype design and it will be refined in the future based on this and further studies.

References

[ARC86] Archibald, James and Jean-Loup Baer, Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. *ACM Transactions on Computer Systems*, Vol. 4, No.4, November 1986, pp 273-298.

[ARL88] Arlauskas, Ramune, *iPSC/2 System: A Second Generation Hypercube.* ACM, 1988.

[BRO87] Brooks III, E. D., A Butterfly Processor-Memory Interconnection for a Vector Processing Environment, *Parallel Computing*, 4:103-110, 1987.

[GLE92] Gleeson, Tim, *Memory Hierarchy in Asura: Discussion document.* an unpublished work of Kubota Corporation, July 1992.

[GOO91] Woest, Philip J. and James R. Goodman, *An Analysis of Synchronization Mechanisms in Shared-Memory Multiprocessors.* International Symposium on Shared Memory Multiprocessing, pp 152-165, April 1991.

[IEEE91] *SCI: Scalable Coherent Interface; Logical, Physical and Cache Coherence Specifications.* Draft for Sponsor Ballot Review, IEEE, New York, N.Y., January 1991.

[KUC86] Kuck, David J., Edward S. Davidson, Duncan H. Lawrie, and Ahmed H. Sameh. Parallel Supercomputing Today and the Cedar Approach. *Science*, 231(2):967-974, 1986.

[LEN92] Lenoski, Daniel E., *The Design and Analysis of DASH: A Scalable Directory-Based Multiprocessor.* Technical Report No. CSL-TR-92-507, Computer Systems Laboratory, Stanford University, February 1992.

[MOR92] Mori, Shin-ichiro et al. *A Distributed Shared Memory Multiprocessor: ASURA - Overview and Memory Architecture* -. Submitted to the International Conference on Supercomputing 93.

[NAI92] Naito, Jun, Kazuki Joe, Hiroaki Matsuno and Hiroyuki Nitta, *Performance Evaluation of the ASURA Cluster*, Technical Report 92-ARC-97-10, IPSJ, Dec. 1992.

[SAI92] Saito, Hideki et al. *The Event Correspondent Cache Coherency Scheme and Its Application to Barrier Synchronization*, Technical Report No. 92-ARC-95-2, IPSJ, 1992.

[TAN81] Tanenbaum, Andrew S., *Computer Networks*, Prentice-Hall, Tokyo, 1981

[TIT90] Titan *Hardware Reference Manual*, an unpublished work of Stardent Computer Inc., 1990.

[WIT92] Wittie, Larry D., Gudjon Hermannsson, and Ai Li, Eager Sharing for Efficient Massive Parallelism. Extended version of Parallel Processing Conf. 1992 paper. *First International Symposium on High-Performance Distributed Computing.* September 1992