EXPLORING THE PERFORMANCE OF THE MYRINET PC−CLUSTER ON LINUX

Roberto Innocente − Olumide S. Adewale

ABSTRACT

Both the Infiniband and the virtual interface architecture (VIA) aim at providing effective cluster communication. However, the specification of the former does not define APIs. It contains an abstract description of the protocol verbs. The dependence of an implementation on the various features of the hardware, firmware, and software are not defined in the Infiniband architecture specification making it difficult to implement now. VIA is a standard mechanism architecture independent from physical layer to deliver performance directly to applications. Existing implementations of VIA come in three flavours, one of these implementations based on the Myricom's Myrinet intelligent NIC is examined in this paper. The paper further presents a series of experiments on the communication performance of three different Myrinet PC−cluster on Linux. The experimental results obtained show that while there is a substantial difference of performance between Myrinet 2 and Myrinet 2000 boards, there is no significant minimum latency difference between the two borads because of the overhead of the PCI bus. It was further observed that there is no significance difference in performance on the different platforms tested using Myrinet 2000 board.

Keyword: Myricom, Myrinet, VIA, Infiniband, VI−GM

1. INTRODUCTION

In a cluster environment, a number of commercial−off−the shelf computers are combined to act as a single, microprocessor system. The number of cluster implementations is poised to explode. High−availability systems are used to achieve 4 − to 5 − 9 up time[1]. They focus upon reliability and used a shared storage among other technologies to permit processes to migrate from one failed system to failover system. Clusters for scalability are no longer just for university research. Traditionally considered scientific solutions, these are being used extensively in applications, such as oil research, integrated circuit design, genome mapping, protein folding, turbo fan design, automobile design, nuclear weapons simulation, galaxy simulation, climate simulation, and earth/ocean modelling. And load−balancing solutions, which can achieve reliability and scalability benefits, are common place with web applications.

The performance of parallel applications running on clusters depends on the implementation of the nodes and the LAN or SAN that acts as communication system. Emerging distributed and high performance applications require large computational power as well as low latency, high bandwidth and scalable communication subsystems for data exchange and synchronous operations. In the past few years, the computational power of desktop and server computers has been doubling every eighteen months. The raw bandwidth of network hardware has also increased to the order of Gigabit per second. However, the performance of underlying hardware is not delivered to the applications due to the traditional architectures of communication subsystems. The traditional networking protocols for inter−processor communication and I/O usually need heavy kernel involvement and extra data copies in the communication critical path. TCP/IP is the most popular one due to its flexibility and layered structure. But due to high software processing overhead on host arising from the cost of making system calls, demultiplexing, protocol stacks and copying data between address spaces and buffer; their communication performance is not sufficient for all application types. Specifically, protocol stacks require that all network access be through the operating system, which contributes a significant overhead to both the transmission and reception paths. Applications with high communication frequency and small messages may suffer from the high latency of such solutions.

For a modern SAN with several Gbps hardware throughput, it is critical to eliminate or minimise this software overhead along the critical path of send/receive communication operations. To overcome the inherent overheads caused by this software processing, significant research efforts have been carried out, which among others include zero−copy and user−level network protocol schemes.

Zero−copy means that applications are given direct buffer management at the network layer. Unfortunately, this optimisation is awkward to implement with the current BSD sockets API specification. The semantics of the BSD socket interface states that the user data buffer could be reused immediately after the system call, that is write, has fully copied the data to the system buffer. However, with zero−copy protocol, the only data buffer is the user data buffer. This implies that the user data buffer may not be ready for re−use immediately. To preserve the existing semantics, a common approach is to have memory pages containing the data marked as Copy−On−Write (COW) by the memory management unit. This essentially prevents the application from writing to the pages. If the application attempts to overwrite the data, the operating system creates a copy of the original data and transparently maps the copy into the user virtual address space. After the data has been acknowledged the COW mapping can be released. Avoiding the copy at the receiver side is much more complicated since the network adaptor requires knowledge about the protocol states in order to determine where the data should be placed. In addition, memory pages to which data is to be placed must be locked to prevent it from being paged out to disk. Smith and his colleagues outline a variety of approaches for implementing the zero−copy protocols [2].

The user level network protocol scheme, also known as operating system−by−pass, exposes a lower level abstraction of the network device than is normally provided. Applications are given their own virtual interface so that they can manipulate the network queue buffers for transmission and reception. Bypassing the conventional operating system mechanisms allows applications to reduce the latency and per−packet overhead of using the network. This is because the overhead associated with the system calls is avoided and the extra data copies are minimised. In addition, better communication scheduling can be achieved since the network traffic can be predicted more accurately with either mathematical or probabilistic models.

Examples of the early user−level network protocols used in these user−level networks are U−Net, Fast Message (FM), Active Message (AM), GM [3, 4, 5, 6]. These protocols tried to exploit the specificities of parallel computing on clusters in order to propose more efficient and dedicated subsystems. As early as 1992 in the context of massively parallel processor, AM was proposed to integrate communications and computations by the use of callbacks triggered by the message reception rather than using send/receive primitives and their asynchronous variants. The U−Net project dealt with the issue of protected user level access to network interface in general purpose operating systems. The FM library uses some concepts of AM and was first to show that the communication performance of a commodity network hardware could reach the communication performance of much more expensive proprietary machines. Moreover, the MPI port on top of FM was one of the efficient MPI implementation on commodity hardware, which contributed to close the gap with parallel machines. The GM library was developed by Myricom to provide a commercial quality implementation trying to mix the best of these systems with the requirements of a more general− purpose system. In these systems, the involvement of operating system kernel is minimised and the number of data copies is reduced. As a result, they can provide much higher communication performance to application layer.

In the recent past, the design and development of these user−level network efforts lead to an industry standard known as VIA [7, 8]. It is a standard mechanism aims to deliver high performance directly to applications and to eliminate overhead and inefficiencies common to operating systems and network stacks. It defines a generic system architecture that is independent from the physical layer. The seeds of the VIA were grown in academic research. Several research efforts including the FM project at the University of Illinois, the portals project at Sandia National Laboratories and the University of New Mexico, and U−Net at Cornell University were independently investigating optimised communications for clustered systems. All these efforts shared a common theme: pin down the communications buffers in DMA−capable memory, and let NIC read and write directly from and into that memory. Rather than handling the data through several software layers in the protocol stack, the operating system sets up the DMA buffers and gets out of the way. The user and the NIC communicate directly through the DMA buffers, completely bypassing the operating system on message sends and receives. This is the same fundamental mechanism used in VIA. The VIA is significant largely

because it is an industry standard backed by Compaq, Dell, Intel and others, and is thus the future for enterprise servers.

The main objective of VIA is to move the operating system out of the time critical communication path. For this, a VIA NIC provides applications direct access to the network through so called Virtual Interfaces (VIs). VIs are connected to each other in a point−to−point fashion. The VIA is based on descriptor processing. It comprises two work queues, one for send descriptors and one for receive descriptors, and a pair of appendant doorbells. In order to start a data transfer the sender prepares a descriptor that contains the virtual address of the data to be sent, puts it on the send work queue and rings the doorbell. This is referred to as posting a descriptor. Thereupon the NIC starts processing this descriptor, that is, it reads the data from the user buffer via direct memory access (DMA) and sends it through the network to NIC hosting the peer VI. By this time the receiver must have posted a receive descriptor pointing to the destination buffer. When the message arrives the NIC takes the next descriptor from the head of the specified VI's receive queue and writes the data, again by DMA, into memory right into the user buffer. The work queues reside in user memory and the doorbells are mapped into user address space as well, thus there is no kernel call needed to start a data transfer. The completion of a descriptor can be checked either by polling its status or by blocking the process by means of a kernel trap. VIA also provides a remote DMA (RDMA) mechanism where the sender specifies both the local and the remote address. Fig. 1 shows the relationship between VIA components while Fig. 2 illustrates a VI.

Another characteristic of VIA is that all memory, which is to be used to hold descriptors or data buffers must be registered in advance. That means that all involved memory pages are locked into physical memory and the addresses are stored in the NIC's translation and protection table.

More recently, InfiniBand architecture [9] has been proposed as an industry standard that provides availability, reliability, scalability, and performance for server I/O and inter−server communication. In InfiniBand, computing nodes and I/O nodes are connected to the switched fabric through Channel Adapters. It provides a verbs interface which is a superset of VIA. This interface is used by the host operating systems to communicate with host channel adapters. It also provides many novel features: three different kinds of communication operations such as send/receive, RDMA, and atomic; multiple transport services such as reliable connection; and different mechanisms for quality of service such as service levels and virtual lanes. In addition to providing scalability and high performance, InfiniBand also aims to meet applications' need for reliability, availability and serviceability. The Infiniband specification not only describes the physical interconnect and the high−level management functions but also a wide range of functions from simple unreliable communication to network partitioning, with many options. The specification defines line protocols for message passing. The specification does not however define APIs; it contains an abstract description of the functions that must be present, as a collection of proto verbs. The dependence of an implementation on the various features of the hardware, firmware, and software are not defined in the Infiniband architecture specification, which makes its objective difficult to be realised through standard means for now. Therefore, this paper focuses on the VIA.

Existing implementations of the VIA come in three flavours. Native implementations, such as Emulex's cLAN VIA [10], potentially offer the highest level of performance, but require custom NIC hardware. The VIA programming interface can be provided on existing traditional networks by using software emulation. M−VIA [11] consists of a user−level library and loadable kernel module for Linux, and supports VIA over Ethernet. The third approach is to use an intelligent NIC. Intel's proof−of−concept [12] implementation and Berkeley VIA [13] both make use of Myricom's Myrinet − a gigabit−class, programmable, user−level accessible NIC.

Myrinet provides low−latency, high−bandwidth, end−to−end communication between two processes in the cluster. These are achieved by giving a user process a direct access to the network interface, avoiding copies of data and by−pass the operating system in a fully protected fashion through its GM message passing system.

The technical steps toward making Myrinet a reality included the development of robust communication channels with flow control, packet framing and error control; self−initialising low−latency, cut−through switches; host interfaces that can map the network, select routes, and translate from network address to routes, as well as handle packet traffic. Finally, it streamlined host software that allows direct communication between user processes and the network.

In this paper, a high−performance VIA implementation Myricom's Myrinet is examined. The paper further investigates the performance differences between two generations of Myrinet on three different PC−cluster configurations and how they affect application performance by means of a benchmark suite. From the experimental results obtained, it shows while there is a substantial difference between the two generations considered, there is no significant minimum latency difference between the two Myrinet generations because of the PCI bus. It was futher established that there is no significant difference in performance on the different platforms tested using Myrinet 2000 switch.

The rest of the paper is structured as follows: Section two gives a brief overview of the Myrinet 2000 technology, GM and VI−GM. Section three deals with the experimental setup and the results obtained from the experiments which are presented in form of charts. Section four touches briefly some related work on comparative analysis of high−performance interconnect systems while Section five concludes the paper.

2. AN OVERVIEW OF MYRINET 2000, GM AND VI−GM

While a detail description of the Myrinet 2000 technology can also be found in [14]; a brief overview of the technology is provided here followed by that of VI−GM and GM.

Myrinet−2000, developed by Myricom, is a proprietary network technology. Myrinet is a switched, Gigabit per second network that is widely used in Beowulf clusters and embedded system. A Myrinet−2000 network [14] is composed of crossbar switches and network adaptors. Network adaptors are connected to the switches through point−to−point duplex links and the crossbar switches can be interconnected in an arbitrary topology.

The basic building block for the Myrinet−2000 switches is a 16−port crossbar. Currently the maximum number of hosts supported within a single unit (enclosure) is 128, using 24 16−port crossbar switches. The topology used to interconnect these switches is a full−bisection Clos network. By applying the same Clos network principle, Myrinet−2000 can scale up to 8192 hosts that can potentially offers a network bisection bandwidth in the order of Terabits per second. Data packets are wormhole routed from one network adaptor to another through a series of crossbar switches enabling a latency of approximately half a microsecond. Flow control is achieved by inserting STOP and GO control bytes into the opposite channel of a link by the receiver side to stop or restart data transmission on the sender side. As a result, Myrinet would not normally drop packets unless the receiver fails to drain the network. Error control is accomplished by computing an 8−bit cyclic−redundancy check (CRC−8) on the entire packet including packet header. The CRC−8 is then carried in the packet trailer and recomputed in each network stages. Thus, data packet entering a host interface with a non−zero CRC−8 indicates transmission errors.

The Myrinet−2000 network adaptor is a programmable communication device that provides an interface to the SAN. It consists of three major components (Fig. 3) namely a custom VLSI (LANai) chip, a synchronous static RAM memory, and a PCI Bridge and a DMA controller.

Like other modern network adaptors, Myrinet adapters have a programmable network interface processor known as LANai9. LANai9 is a 32−bit RISC processor that operates at up to 133MHz for the PCI64B interfaces, or at up to 200MHz for the PCI64C interfaces. Using the Myrinet Control Program (MCP), which is stored in the onboard SRAM, LANai9 controls the data transfer between the host and the network (through the host and packet interface), performs data

buffer management (through memory interface), and maintains network mapping and monitoring. The benefit of a programmable network processor is that it enables researchers to explore many protocol design options.

The onboard Myrinet−2000 SRAM size ranges from 2Mbytes to 8 Mbytes and operates at the same clock speed as LANai9, i.e., at up to 133 MHz for the PCI64B interfaces or at up to 200MHz for the PCI64C interfaces. This suggests that the maximum attainable bandwidth is approximately 1,064 Mbytes/s and 1,600 Mbytes/s for the PCI64B and PCI64C respectively. The SRAM is accessible from both the onboard local bus and the external system bus. Both local bus and external system are both 64−bit wide but the local bus is clocked at twice the chip−clock speed, permitting two local bus memory accesses per clock cycle.
To increase the data transfer rate, a Myrinet−2000 network adaptor is equipped with three DMA engines. Two DMA engines are associated with the packet interface: one for receiving packets and one for sending packets. The third DMA engine is used for data transfer between the SRAM and the host system memory through the host interface. Like most systems that support DMA, the on board memory can be mapped into user space and is thus accessible directly to user processes. This mapping is performed in two steps: the operating system first maps the NIC address space into kernel space and then, on a request by user, the kernel region is mapped into user space. This memory mapping technique is commonly known as "memory pinning". In order to support zero−copy APIs efficiently, the DMA operations can be performed with arbitrary byte counts and byte alignments. Additionally, the DMA engine also computes the IP checksum for each transfer and provides a "doorbell" signalling mechanism that allows the host to write anywhere within the doorbell region, and have the address and data stored in a first−in−first−out queue in the local memory.

The host processor can also access the Myrinet−2000 SRAM through the programmable input/output (PIO) interfaces. With PIO, the host processor reads the data from the host memory and writes it into the Myrinet−2000 SRAM. This mode of data transfer typically results in many PCI I/O bus transactions. Although Myrinet−2000 PCI64 interfaces are capable of sustained PCI data rates approaching the limits of the PCI bus, the network performance greatly depends on the data transfer rate of the host’s memory and PCI−bus implementation.

All Myricom software for the PCI64 family of interfaces is based on the GM MCP and the GM API. GM is a lightweight message−based communication system for Myrinet featuring low CPU overhead, low−latency and high−bandwidth. Reliable delivery is provided between end−points called ports. Two levels of priority are supported, and message order is preserved for messages of the same priority between the same sending and receiving ports. Messages must be sent from and received into DMAable memory.

The software components of GM include a library, a driver and the NIC firmware called the Myrinet MCP. The library links with the client and provides the API. It includes functions for opening ports, allocating DMAable memory, sending and receiving messages, etc. The former operations are carried out with the help of the driver, but no driver involvement is required for message sends and receives. These are carried out through direct interaction between the client and the MCP as described below.

The MCP consists of four state machines: SDMA, RMDA, SEND and RECV. When a client wishes to send a message, a send descriptor describing the message is written to a memory−mapped location in the NIC. The SDMA state machine detects the descriptor, builds a corresponding send token and inserts the token into an appropriate send queue. Further, it polls send queues, prepares packets and DMAs data into transmit buffer. The SEND state machine transfers packets from the transmit buffer to the network.

In GM, queues are hierarchically arranged. At the highest level is a circular connection queue, which has an entry for each remote node with which communication is active. Each connection entry points to a circular queue of sub−ports that are communicating over that connection. Each sub−port entry points to a queue of send tokens. Some tokens have been serviced completely and

are awaiting acknowledgements, others that have not be completely serviced are called sendable tokens, At any time, one connection entry and a corresponding sub−port entry are designated as the head entries. After one packet dispatch corresponding to a sendable token from the head sub−port, the sub−port queue for the head connection is rotated and so is the connection queue. This allows fair network access. The RECV and RDMA state machines similarly co−operate for message receives.

VI−GM is a fully compliant implementation of the VIA. It supports the whole API, for all reliability levels. This implementation is a user−level port on top of GM, the low−level communication interface for Myrinet, supported by Myricom. As the main resource used by VI−GM is memory, it supports a very large number of VI objects like VIs, completion queues, etc.

No configuration is required in VI−GM, only the VI−GM connection manager should be initially running when VI−GM applications are executed (if they need to establish connections, but as VI is connection−oriented, it's almost a requirement). Customers do not need to manage GM ports, VipOpenNic will try to open the first GM port available (starting from port 2).

VI−GM is completely thread−safe, and multiple VI−GM threads spawned from the same process share the same GM port. VI−GM does not use any signals and does not overload any function from the C library. VI−GM supports a very large number of nodes (65000+) and a virtually unlimited number of local threads (the number of NIC handles is limited to comply with the Intel conformance suite).

3. EXPERIMENT ENVIRONMENTS

The experiment environment consists of three different cluster configurations. The first cluster is made of Pentium III@550MHz nodes with Myrinet 2 switch, the second cluster comprises of Pentium III@933 MHz nodes with Myrinet 2000 switch while the third cluster is made of Pentium 4@1.7GHz nodes with Myrinet2000 boards and switch. These nodes are equipped with two Intel Pentium P4 (Willamette) CPUs with 256k L2 cache, running at 1.7 GHz, with an Intel 860 chipset, RIMM PC800 memory and 2 PCI 64 bits / 66 MHz slots. The Myrinet boards are PCI64C with 2M of RAM and a Lanai 9.2 processor running at 200 MHz. The switch is a 16 port Myrinet 2000 switch (SW16M).

The GM software we used is a proprietary protocol and API used by Myricom on their boards. As it comes, it is configured to use at most 256 bytes tranfers between the chipset and the Myrinet board. With the 64−bit PCI bus, as it exists on our nodes, this is not sufficient to exploit the available bus bandwidth (it will perform just 32 data transfers before to relinquish the bus). With these short burst transfers, it is only able to measure a PCI bandwidth of only 147MB per second. By increasing the PCI burst length to 512 bytes (64 transfers), which is being possible by patching the "gmarch.c" file in the directory drivers/linux/gm, a PCI bandwidth of 227 MB/s reading from the board and of 315 MB/s writing to the Myrinet board was obtained.

The one−way bandwidth and latency of both Myrinet 2 and Myrinet 2000 between any two hosts were measures using the standard benchmark suite. The comparison of the one−way experimental results obtained for the bandwidth and latency of both Myrinet 2 and Myrinet 2000 are shown in Fig. 4 and Fig. 5 respectively. The various experimental results of the one−way experimental results obtained for the bandwidth and latency of Myrinet 2000 on different platforms are shown in Fig. 6 through Fig. 9.

4. RELATED WORK

There has been a substantial previous work on the comparative analysis between various high−performance interconnect systems [16, 17, 18] but none of these comparative analysis has been between generations of the same high−performance interconnect systems as the work reported in this paper.

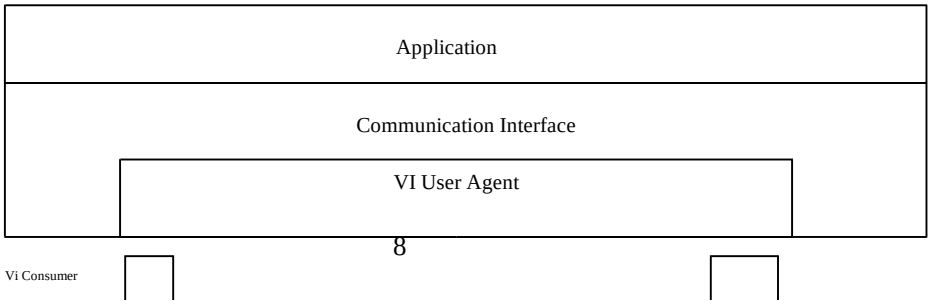
5. CONCLUSION

One of the approaches for implementing VIA is through the Myricon's Myrinet high−performance interconnect system. Myrinet is a new type of local−area network based technology used for packet communication and switching within massively parallel processors. In this paper, a comparative analysis of the performance of two different Myrinet PC−clusters was presented on three differenct PC−clusters. The result obtained show that while there is a substantial difference of performance between Myrinet 2 and Myrinet 2000 boards, there is no significant minimum latency difference between the two Myrinet generations because of the overhead of the PCI bus. The GM software we used is a proprietary protocol and API used by Myricom on their boards. As it comes, it is configured to use at most 256 bytes tranfers between the chipset and the Myrinet board. With the 64−bit PCI bus, as it exists on our nodes, this is not sufficient to exploit the available bus bandwidth (it will perform just 32 data transfers before to relinquish the bus). With these short burst transfers, it is only able to measure a PCI bandwidth of only 147MB per second. By increasing the PCI burst length to 512 bytes (64 transfers), which is being possible by patching the "gmarch.c" file in the directory drivers/linux/gm, a PCI bandwidth of 227 MB/s reading from the board and of 315 MB/s writing to the Myrinet board was obtained.

REFERENCES

1. Anderson R., Lee M. and Chapin S. J., Unravelling the mysteries of clustering, Oses and Netwrok Services, October 2000.
2. Smith J., Brendan C. and Traw S., Giving applications access to Gb/s networking, IEEE Network, July 1993, 7(4) pp. 44−52

3. von Eicken, T.; Culler, D. E., Goldstein, S. C. and Schauser, K. E., Active Message: a mechanism for integrated communication and computation, in proceeding of the 19th International Symposium on Computer Architecture, Gold Coast, Australia, May 1992, pp. 256 − 266
4. Basa, A; Buch, V; Vogels, W and von Eicken, T, "U−Net: a user−level network interface for parallel and distributed computing", 29(5): pp. 40 − 53 15th ACM Symposium on Operating Systems Principles, December 1995
5. Pakin S., Lauria M. and Chien A., High performance messaging on Workstation: Illinois Fast Message (FM) for Myrinet, in Proceeding of the International Conference on Supercomputing, 1995, URL: http://www.supercom.org/sc95/proceedings/567_SPAK/SC95.pdf
6. Myricom, Inc. "the GM message Passing System," http://www.myri.com/scs/GM/, 2000
7. Compaq, Intel and Microsoft, Virtual Interface Architecture Specification, http://www.viarch.org/, 1997
8. Speight E., Adbel−Shafi H., and Bennett J. K., Realising the performance potential of the virtual interface architecture, in Proceeding of the International Conference on Supercomputing, 1999, URL: http://www.csl.cornell.edu/~espeight/papers/ics99.pdf
9. Infiniband Trade Association, Infiniband architecture specification release 1.0, October 2002, URL: http://www.infinibandta.org
10. Emulex cLAN, http://wwwip.emulex.com/ip/products/clan1000.html
11. M−VIA Project, http://www.nersc.gov/research/FTG/via/
12. Berry, F; Deleganes, E and Marie, M, "the virtual interface architecture: proof−of−concept performance results," ftp://download.intel.com/design/servers/vi/via/via_proof.pdf
13. Bounadonna, P; Geweke, A and Culler, D. E., "an implementation and analysis of the virtual interface architecture", Department of Electrical Engineering and Computer Science, University of California, Berkeley, Technical Report
14. Myricom Inc., Guide to Myrinet−2000 switches and switch network, version 27, 2001, URL: http://www.myri.com/myrinet/m3switch/guide/myrinet_2000_switch_guide.pdf
15. Myricom, Inc., "VI−GM: Virtual Interface Myrinet," http://www.myri.com/scs/VI−GM/doc/pdf
16. Chen, Y; Wang, X; Jiao, Z; Xie, J; Du, Z; and Li, S, "MyVIA: a design and and implementation of the high performance virtual interface", to appear in IEEE Cluster2002 Conference
17. Riddoch, D; Pope, S and Mansley, K, "VIA over cLAN network," AT&T Laboratories−Cambridge, Technical Report 2001.14, 2001
Seifert, F; Balkanski, D; and Rehm, W, "comparing MPI performance of SCI and VIA", Conference Proceeding of Sci−Europe 2000, 7 pages, August, 2000



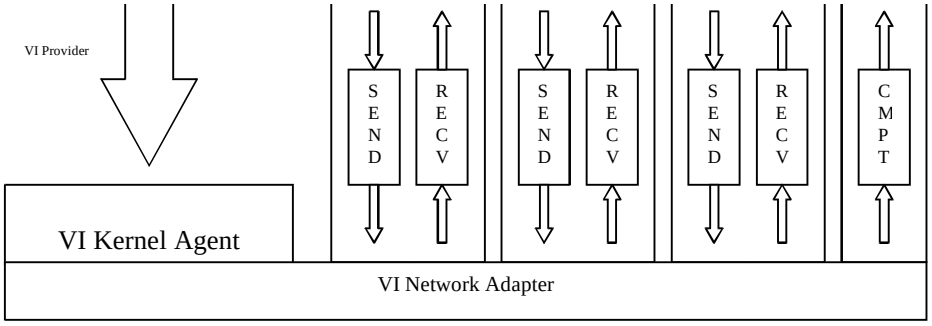


Fig. 1 − Virtual Interface Architecture

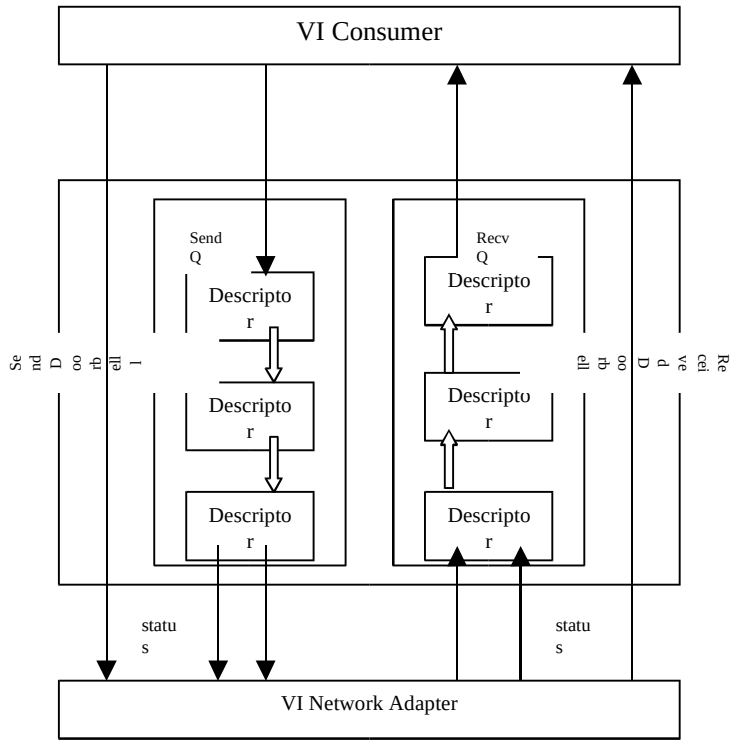


Fig. 2 − VI

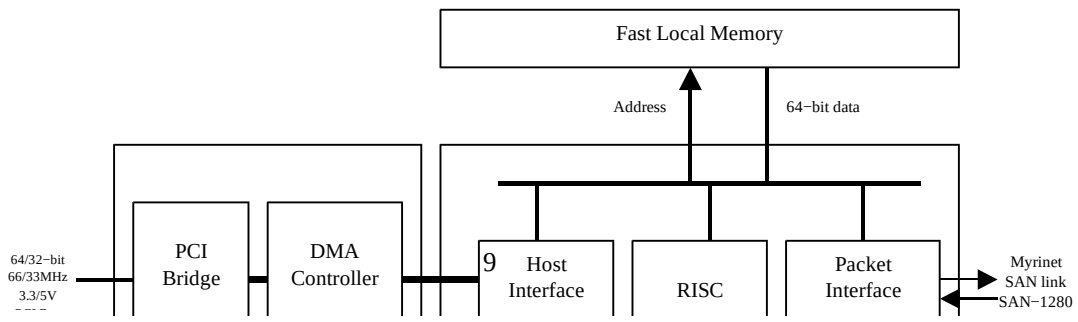


Fig. 3− LANai9 Architecture

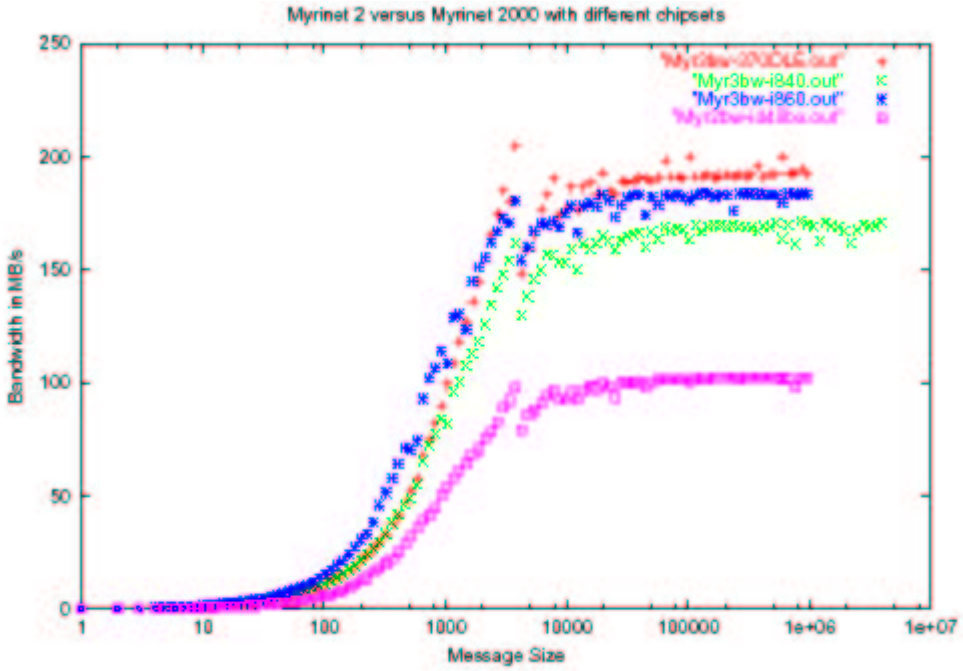
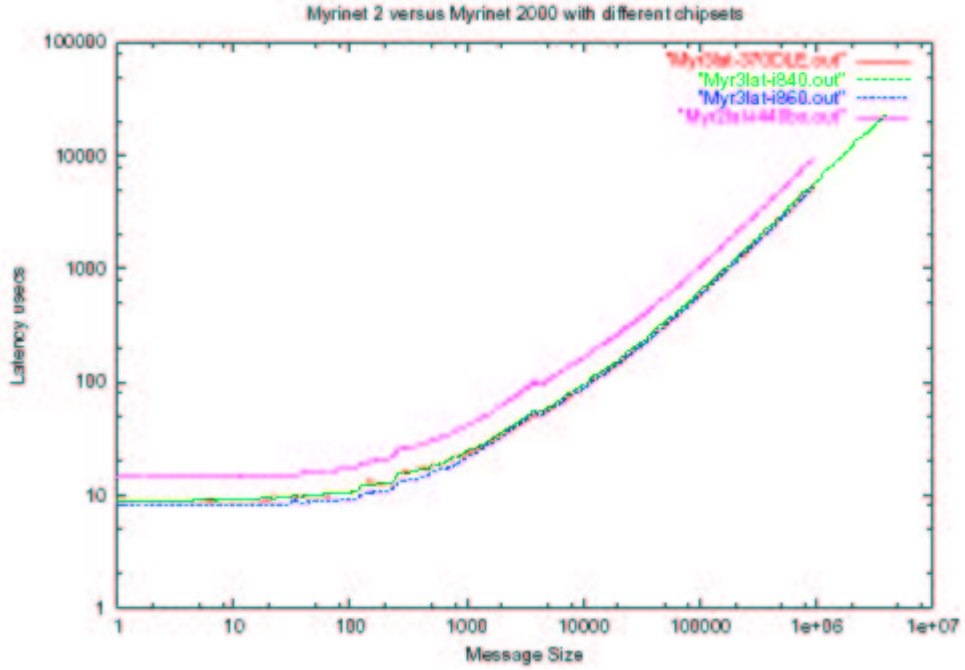


Fig. 4 − Bandwidth Comparison

Fig. 5 − Latency Comparison



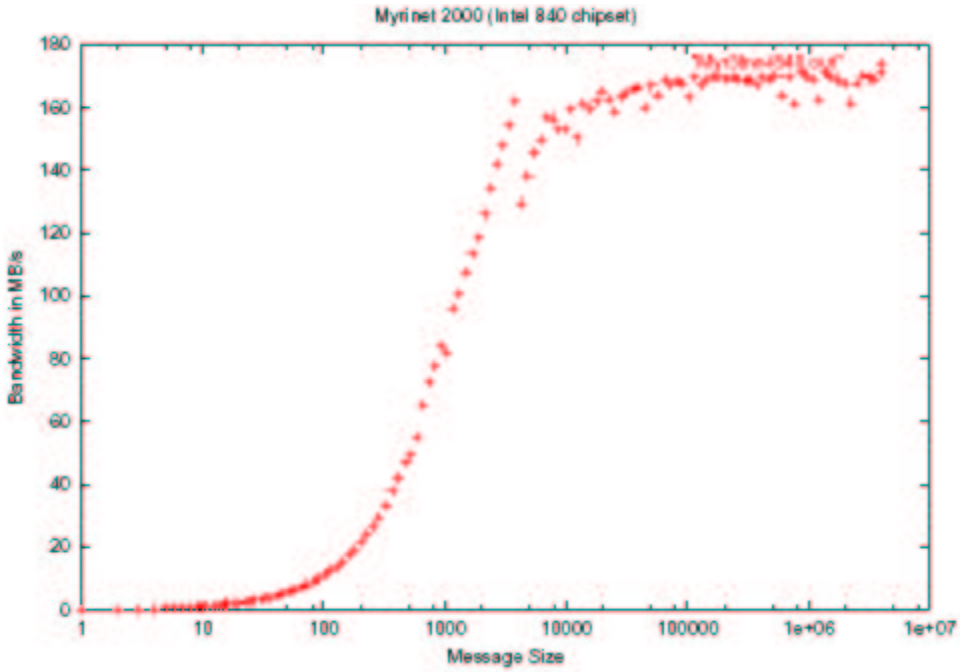


Fig. 6 − Bandwidth with Intel 840

Fig. 7 − Bandwidth with Intel 860

Fig. 8 − Latency with Intel 840 Chipset

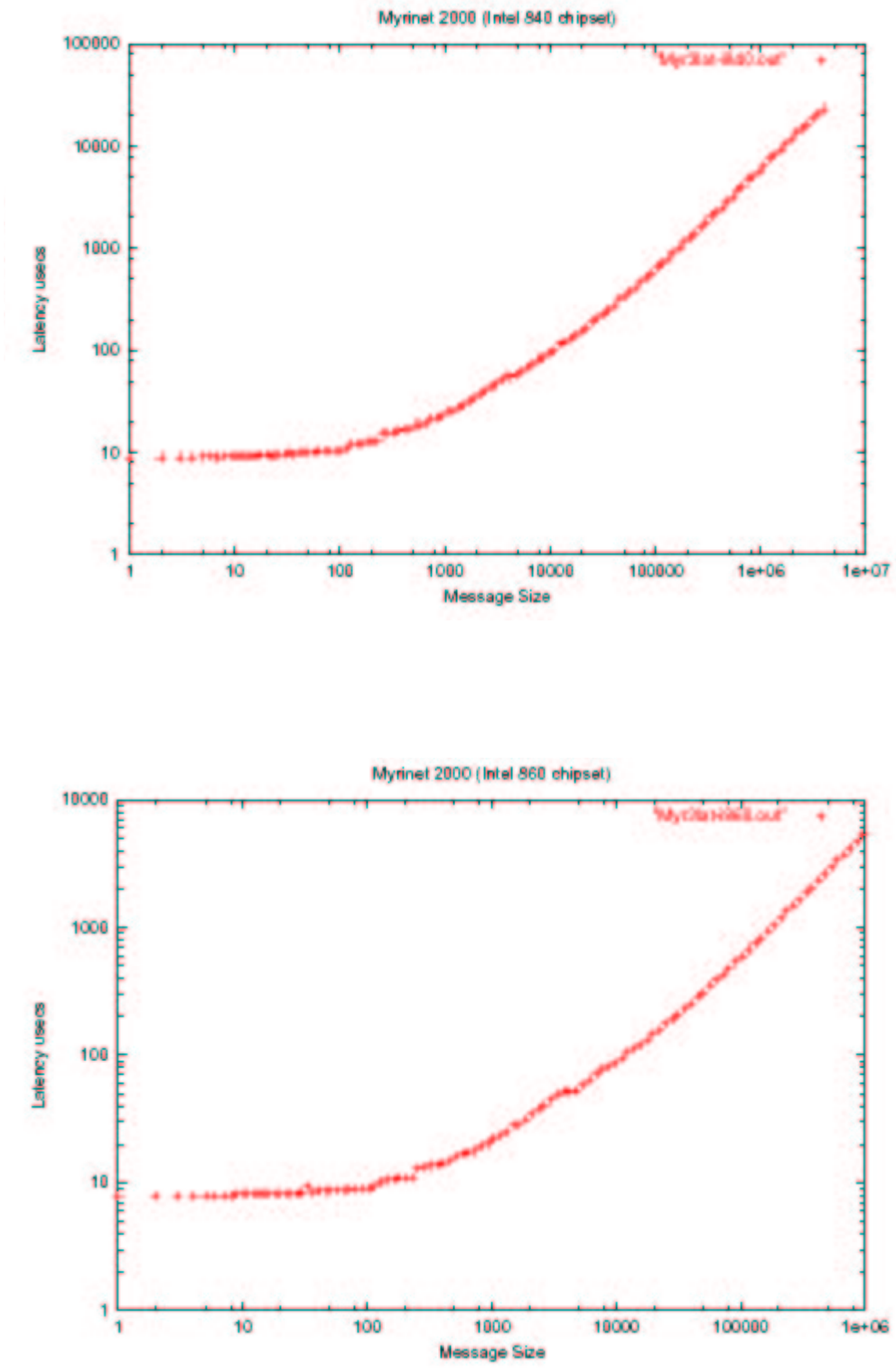


Fig. 9 − Latency with Intel 860Chipset