Performance Evaluation of a Hardware Implementation of VIA

Xin Liu
June 8, 1999

Department of Computer Science and Engineering
University of California, San Diego
xinliu@cs.ucsd.edu

1 Introduction

Virtual Interface Architecture (VIA) is the industry effort to standardize communication paradigms for System Area Networks (SAN). Its design focuses on providing low-latency, high-bandwidth, end-to-end communication of two processes in the cluster. To achieve this aim, VIA defines a set of functions, data structures and associated semantics to access the network interface directly, bypassing the operating system in a fully protected fashion. Currenttly there are a few hardware implementations of VIA, such as the GigaNet, ServerNet and Finisar. Performance evaluation of hardware implementation is necessary to study the design of VIA. In this paper, we compare the performance characteristics of GigaNet with those of Myrinet to study its advantage as a high speed network for cluster systems.

We use a testbed consisting of 32 node HP NetServer cluster, each node having 2 PentiumII 450MHz processors and installing NT Terminal Server 4.0. All the nodes are connected with Myrinet and GigaNet GNN1000. We implement Fast Message and MPI above GigaNet so that we can use the existing parallel benchmarks and real applications to study its performance and compare the results with those from FM/Myrinet.

This paper starts with a discussion of the architecture of VIA and FM, and then introduces why and how we implement FM on VIA. Then we study the performance of FM/GigaNet in terms of in terms of latency, bandwidth, and scalability. And we also analyze the internal behavior of GigaNet and compare it with Myrinet. We conclude with the overall evaluation of GigaNet and VIA.

2 Background and Motivation

2.1 Virtual Interface Architecture (VIA)

Virtual Interface Architecture is the industry effort to standardize communication paradigm for System Area Networks (SAN). At its heart is the concept of an user-level network architecture that that provides application processes a virtual view of a network interface to enable user-level access to high-speed communication devices. It removes the operating system from the critical path of communications while still providing full protection. The key components of VIA include Virtual Instance, Message Descriptor, and Send/Receive Queue.

GigaNet GNN1000 is a host adapter that implements VIA and provides reliable communication, in-order delivery, and support for RDMA-writes.

2.2 Fast Message (FM)

Fast Messages (FM) is a low-level messaging layer designed to deliver the underlying network's hardware performance to the application, even for small messages. For example, FM 2.1 on Myrinet delivers 41 megabytes/second for messages as short as 256 bytes with even higher peak performance. FM makes bandwidth accessible, without requiring changes in applications (or protocols) to increase the message size.

Fast Messages (FM) is also designed to enable convenient and high performance layering of other APIs and protocols atop it. As such it provides key guarantees: reliable, in order delivery, and host-network decoupling as well as a composable interface: efficient gather/scatter, receiver rate control, and per-packet multithreading which make it easy to build higher level interfaces based on FM. Some of the interfaces that have been built include MPI, Shmem Put/Get, and Global Arrays. Because the FM messaging layer is a simple, efficient set of primitives, it is appropriate for implementors of a language runtime or communications library or even the target of a compiler, in addition to application programmer use.

2.3 Motivation for implementing FM on VIA

The main reason for implementation is that VIA does not provide a simple, efficient programming interface for user application. It provides some very low-level functions to operate the VI instance and send and receive messages, but the users programs have to work on buffer management, flow control and connection management. The second reason is that VIA does not provide enough functions for parallel programming. VIA only provides end-to-end communication and does not provide collective communication based on VIA. The end-to-end communication semantics is also not suitable for traditional SPMD parallel computing. User programs have to work on these issues themselves, which is tedious and error-prone. A runtime library above VIA is necessary to make it a user-friendly programming interface. Here we choose Fast Message as the high-level user interface because of its simple and flexible API and variant applications.

2.4 Implementation of Fast Message on VIA (FM/VIA)

We implement Fast Message 2.1 based on VIA. FM/VIA keeps the same programming interface as FM/Myrinet and the majority semantics. In FM/VIA, we just take VI as a powerful data link layer and provide reliable, in order, and flow control services on it. All FM applications can run directly on FM/VIA without any modification.

Connection Management Because VIA is connection oriented, we set up connections between each pair of processes at the initialization and then a FM application can use these connections to transfer data as if there is no connection at all. While FM/Myri uses the node ID and context number pair to identify a process, FM/VIA uses a network address and a discriminator pair to locate a process and setup connection. After initialization, the FM runtime just uses one connected VI for each process of the application.

Buffer Management All the VI instances in a process share a pre-allocated Send Descriptor queue and each descriptor has fixed-size registered memory. To send out a message, the FM runtime gets one descriptor from the free send queue, copies the data into the registered memory, and then posts the descriptor to Send Queue. After the complement of sending, the descriptor is inserted into the free send queue again. At the receiver side, FM runtime allocates and post receive descriptors to the Receive Queue of each VI instance. When a message arrives, we get the receive descriptor out of the Receive Queue, process the message, and post the descriptor back to Receive Queue again. Each receive buffer has fixed size like the send side.

In contrast to sharing send buffers, each VI instance has to allocate its own receive buffers. This is because VIA requires that the receive descriptors must be posted before any messages arrive. One drawback of this approach is that it consumes too much memory and wastes most of it when the number of processes in an application is large. The other limitation is that the size of the packet is fixed, and we cannot use the largest message size (65,519bytes) that VI can provide. We also cannot use the RDMA to transfer

huge data, and we need an additional data copy in user space (from user data buffer to pinned buffer).

3 Basic Evaluation of VIA based on the FM/VIA

To evaluate the GigaNet, we study the performance of raw GigaNet, FM/Giga and FM/Myrinet, and we also study the MPI performance based on FM/Giga and FM/Myri, this is an indication of how much hardware performance can be delivered to applications. Finally, we also study the performance of LINPACK benchmark and some real applications. Figure 1 is the topology of our 32-node HP NetServer testbed.

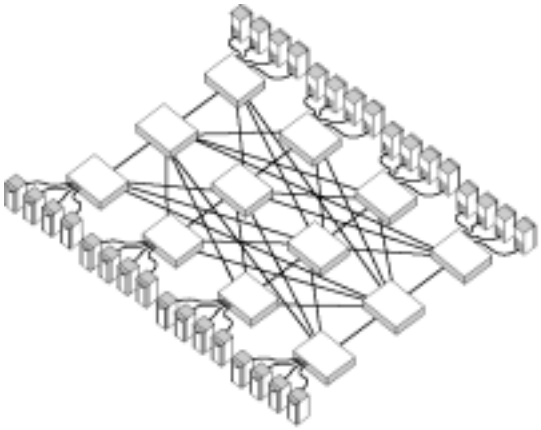

Figure 1 the topology of 32-node testbed

3.1 Performance of raw GigaNet, FM/GigaNet and FM/Myrinet

Latency : Figure 2 shows the point-to-point latency of Raw GigaNet, FM/GigaNet and FM/Myrinet. These systems have the similar performance, and both of their latency for small messages is about 10 usec. While FM/GigaNet provides more flexible API on the raw GigaNet, it only adds a little overhead to the raw GigaNet.

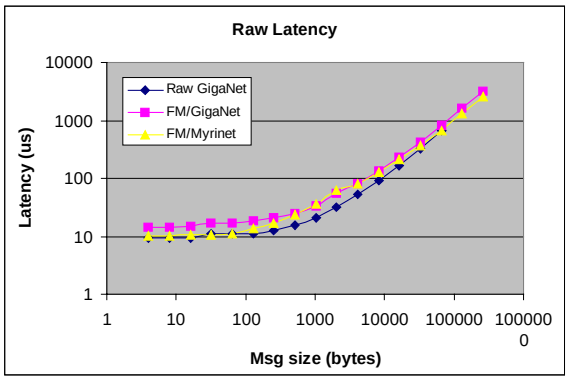

Figure 2 Raw Latency

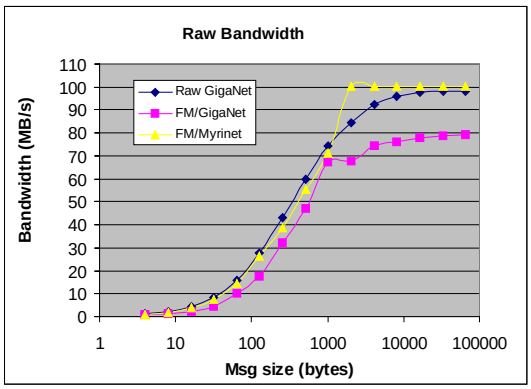

Figure 3 Raw Bandwidth

Bandwidth: The point-to-point bandwidth test measures the rate at which large amounts of data can be copied from one endpoint to another endpoint across the network. We can see that the raw GigaNet and FM/Myrinet have the same bandwith, but FM/GigaNet's maximum bandwidth is limited to about 80MB/s. The reason is that FM/GigaNet has an extra data copy from registered buffers to user buffers. As we will explain later, this copy is usually necessary for all VIA applications, but the raw GigaNet bandwidth test ignores this overhead.

3.2 MPI performance

As we have seen from the above latency and bandwidth test result, both GigaNet and MyriNet have small latency and high bandwidth at low level. But it is also a key problem that how much network performance can be delivered to application level. We choose the MPI as the typical application interface and study its performance to show what performance the common applications can expect from GigaNet and Myrinet.

Figure 4 and Figure 5 show the latency and bandwidth of MPI on FM/Giga and MPI on FM/Myrit. Like the MPI on FM/Myri, the MPI on FM/Giganet can deliver almost 70-90% performance to higher level. To justify our approach of using GigaNet, we also present the performance result from MPI/PRO for GigaNet. The latency of MPI/PRO is a little larger than FM/Giga and the bandwidth of MPI/PRO is much smaller than FM/Giga. Since the MPI/PRO is implemented directly on GigaNet, we expect better performance than that of FM/Giga. The result shows that FM is an effective and efficient enhancement of VIA Interface.

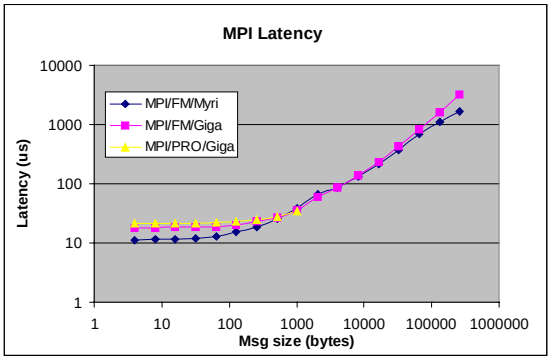

Figure 4 MPI latency

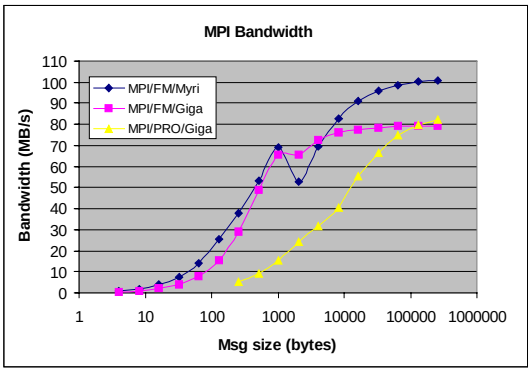

Figure 5 MPI bandwidth

3.3 LINPACK benchmark

The LINPACK benchmark is a benchmark to test the overall performance of a parallel machine in terms of Megaflops, millions of floating point operations per second (Mflop/s). The LINPACK benchmark features solving a system of linear equation, Ax=b with parallel Gaussian elimination. To achieve good performance, both the processor capability and communication overhead are important. Figure 6 shows the LINPACK benchmark result from 1 to 32 processors for both FM/Myri and FM/Giga. The whole system shows 5GFLOP/s, and each node has about 160MFLOPS/s. Both of them exhibit almost linear increase of Megaflops, and this shows both of them have good scalability. Although the FM/Myri has better communication performance than FM/Giga, the former's LINPACK benchmark result is a little worse than FM/Giga's. This is due to that FM/Giga has less CPU overhead than FM/Myri.

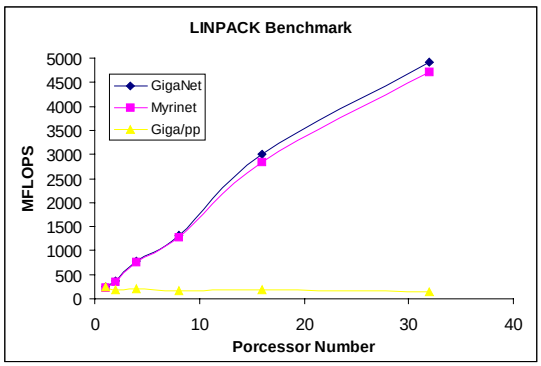

Figure 6 LINPACK benchmarks

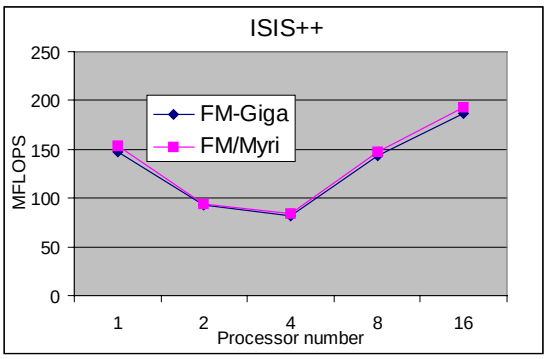

Figure 7 ISIS++

3.5 NAS Parallel Benchmarks and ISIS++

The Numerical Aerodynamic Simulation (NAS) Parallel Benchmarks (NPB) were developed to evaluate the performance of parallel computing systems for workloads typical in NASA and the aeronautics community. The objective of the Parallel Benchmark effort is to measure and report performance of the latest commercially available high performance computing (HPC) systems. In our study, we use the MPI implementation of NPB2.3. The results are show in Table 1.

ISIS++ library, an object oriented library of scalable iterative sparse linear system solution methods. For parallel execution, ISIS++ uses the distributed memory (SPMD) programming paradigm with inter-process communication handled by MPI. ISIS++ includes a collection of matrix and vector classes which provide mathematical functionality (in terms of products and norms, etc.) so that higher level source code (e.g., solvers and preconditioners) can remain constant when moving between serial and parallel platforms. The results are showed in Figure 7.

Similar to the result of LINPACK benchmark, FM/Giga and FM/Myri get the same performance on both of these two programs.

		bt	cg	ep	mg	sp	ft	is	lu
FM/Myri	1	64.06	30.35	1.1	52.13	42.74	44.41	3.47	60.24
	2		60.23	2.45	103.89		88.38	7.1	119.53
	4	237.34	117.24	3.99	201.23	162.34	174.39	14.02	231.39
	8		235.34	5.47	394.45		337.89	26.34	478.42
	9	514.78				350.98			
FM/Giga	1	64.43	30.42	1	51.65	42.45	44.49	3.88	59.98
	2		60.73	2.34	100.45		88.23	7.7	118.46
	4	235.83	118.34	3.94	196.34	161.34	173.9	14.23	228.34
	8		231.45	5.23	389.29		327.96	27.89	459.73
	9	534.89				347.28			

Table 1 the NPB Performance

3.6 Conclusion

From these basic evaluations based on application performance, we can find that GigaNet has similar performance to Myrinet, and it enables the use of commodity servers to build low cost, high performance clusters. Through the use of VIA, GigaNet delivers ultra low latency, high throughput, and point-to-point connectivity with minimal CPU overhead. This hardware performance can be delivered to higher level applications through a carefully designed interface like FM.

4 Detailed Studies of GigaNet and VIA

The last section deals mainly with application level. To give a better understanding of VIA and GigaNet, we study the internal behavior of GigaNet and analyze some scalability and memory management issues related to VIA.

4.1 Connection management

Because VIA is connection-oriented, users must setup the connection between two endpoints before communicating. Because the connection setup and tear-down need kernel access, to harness the benefits of user-level network, we must take them out of the critical path. This means that in a real application, all the connections are setup before the real task and remain connected until the end of the application. This is exactly the approach that FM/VIA takes.

Fully connecting all the possible endpoints in an application brings up the scalability issue. If there are N processes in an application, then there are totally N^2 connections in the system.

When N is large (for example, 1024), to maintain theses connect is a large CPU burden. In addition to this CPU overhead, it also need more switchers to keep them full connected. Now each GigaNet NIC supports 1024 connections, and the 8-port GigaNet switcher supports 8192 connections.

To calculate how many connections can be setup at the same time in a system, let us assume that:
N: the number of all processes in a system
P: the number of physical nodes (maybe SMP)
C: context number per node
So we have N=P*C.
First, we should support the connections of all the processes in a node through a Link. We get:
$C*C + C*(N-C) = C*N <= 1024$
Second, we splits all the processes into two partitions, we can calculate the Links necessary to fully connect all the processes through the cut. It is
$x*(N-x) <= N*N/4$, where x is the process number in a partition

From these two equations, we can figure out, when

C=2, N = 512, we need 64 links
C=4, N = 256, we need 16 links
C=8, N = 128, we need 4 links

We can find that to keep the system full connected, we need considerable switchers. For example, in our 32-node cluster system, we use 12 8-port switchers (almost 3 nodes require a switcher). When the system become larger, the situation becomes worse. Of course, the GigaNet switcher implementation is not good enough, but the connection-oriented VIA is the key reason for this limitation. In the connectionless Myrinet, each endpoint can send a message to the other endpoint directly, and there is no overhead to setup and maintain the connection. Of course, we need some mechanism to protect and use the memory efficiently.

4.2 Memory Management
The send and receive buffers of VIA are organized through the packet descriptors. Descriptors are of variable length and virtually contiguous. Each descriptor consists of a single control segment followed by one or more data segments. The data segments can be used to scatter/gather operation of data.

Memory Copy or Register on the Fly In order to enable memory protection and virtual address translation, all the memory for receive and send descriptors must be registered (pinned). There are two choices at this point: one is the register/unregister on the fly, the other is to use the pre-registered buffer pool. To decide which is a suitable approach, we must study the cost of both mechanisms. The cost for pin/unpin memory is much higher than memory copy for small messages and a little lower for large messages (Figure 6). The effects of the memory copy and pin/unpin overhead are important to the overall performance. Figure 7 and Figure 8 show their latency and bandwidth respectively. As we expected, the pin/unpin operation increases the latency for small messages greatly and the memory copy limits the bandwidth for the large messages.

The ideal solution is to use the later for small messages and the former for large messages (>16KB). But it is difficult to mix them in an application. Since most of the messages in real applications are small, we choose the fix size pre-allocated buffer pools in FM/Giga. Here we get tradeoff between latency and bandwidth. The additional copy for each message decreases FM/Giga bandwidth to about 80MB/s. But this tradeoff is reasonable for most applications. Another reason to give up registering memory on the fly is that VIA requires all the receive

buffers to be posted before the arriving of messages. User programs must pay great efforts to synchronize the receiver and sender and make sure that the receive buffers are post in the head. We think it is unpractical for user to use this feature efficiently.

To achieve a real zero copy communication and avoid the overhead of a pin/unpin operation, applications should custom the memory management. The idea is that applications setup a registered buffer pool and all the memory allocations are satisfied from this pool. This approach is an application-aware interface or we have to wrapper the memory management system calls. It is only suitable for some system-level middlewares, such as DCOM, but not for most ordinary applications.

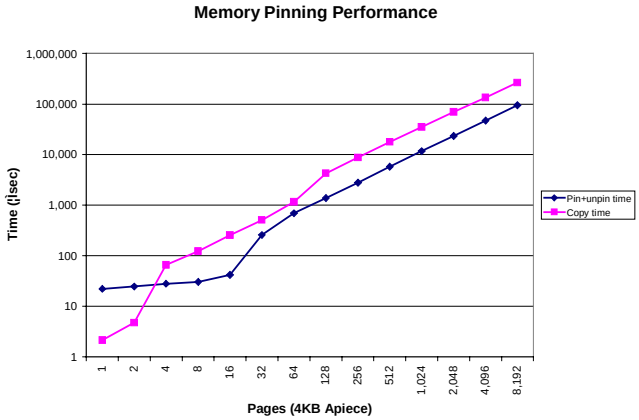

Figure 7 Memory Pining Performance

There are some other overheads for operating descriptors: the DMA used by NIC to read and write the descriptors and the doorbell mechanism of notification. Based on these observations, and also because all receive descriptors must be post before the arriving of messages, it is suitable for the applications to allocate and register the buffers before the real task. As a result, the scatter/gather function is not very useful, since all the data must be copied in/out the registered buffers. Unless the application allocates memory directly from the registered memory, and this will complicate the memory management.

Since we must allocate receive buffers for each VI in a process, so the receive buffers are linear to the number of processes in the system. This is comparable to the requirement in FM/Myri, which also allocate receive buffers for each possible senders in the system.

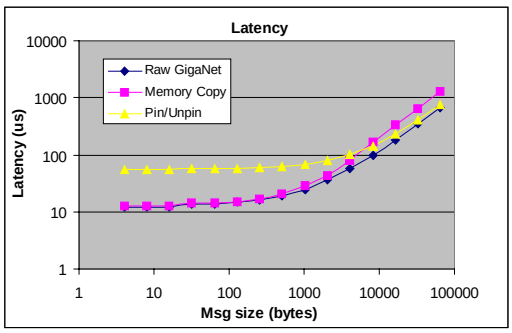

Figure 8

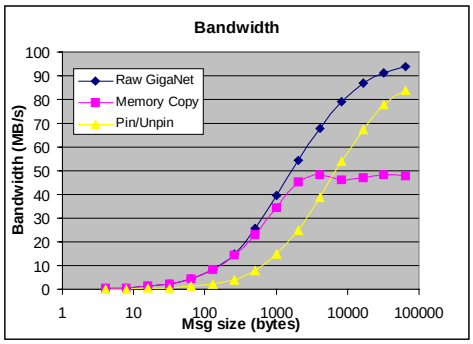

Figure 9

4.3 Scalability of VIA

During the development of FM/Giga, we found that the scalability of VIA is not good enough. The first limitation comes from the connection management as we mentioned in section 4.1, and the limitation of GigaNet switchers makes this problem worse. In fact, the memory management (in section 4.2) is also another limitation. The second limitation comes

from the send/receive queues management, because the descriptors are located in the host memory, and during the send and receive messages, so the NIC must access these descriptors through the memory bus. This will increase the bus transactions, which in fact are the bottleneck of high-speed networks.

5 Summary

In this paper we make a performance evaluation of a hardware implementation of VIA. The evaluation focuses on three points: application performance, connection and memory management, and scalability issues.

We study and compare the performance of FM/Giga, FM/Myri and MPI/Giga, which is an MPI implemented directly on VIA. We study the latency, bandwidth, and also the LINPACK benchmark and some real applications. The conclusion of this study is that GigaNet can provide low-latency and high bandwidth to applications. And FM on VIA is an efficient enhancement to VIA.

Through the analysis of connection and memory management, we conclude that the scalability of VIA is not good. The VIA is connection-oriented and it creates a big overhead to maintain the full connection of all the endpoints in a large system, because the total number of connections is in the order of square of number of endpoints. This limits the scalability of VIA applications. The overhead for buffer management is also not trifling. Because VIA must maintain a receive buffer queue for each endpoint in the system and they can not the receive buffers. The first problem is that it requires a large amount of memory and the second problem is that a host must post and remove the descriptor from the NIC for each message. The overhead also limits the scalability.

Funding Ack

The research described is supported in part by DARPA orders #E313 and #E524 through the US Air Force Rome Laboratory Contracts F30602-96-1-0286 and F30602-97-2-0121, and NSF Young Investigator award CCR-94-57809. It is also supported in part by funds from the NSF Partnerships for Advanced Computational Infrastructure -- the Alliance and NPACI. Support from Microsoft, Hewlett-Packard, Myricom Corporation, Intel Corporation, Tandem Computers, and Platform Computing is also gratefully acknowledged.