Exploring the Performance of VI Architecture Communication Features in the Giganet Cluster LAN

Hermann Hellwagner, Markus Lachowitz, Matthias Ohlenroth

Institute of Information Technology, University Klagenfurt
Universitätsstraße 65–67, A–9020 Klagenfurt, Austria

Abstract. *The Virtual Interface (VI) Architecture standard was developed to satisfy the need for a high-throughput, low-latency communication system required for cluster computing. This paper presents the results of a performance study of one VI Architecture hardware implementation, the Giganet cLAN (Cluster LAN). The focus of the study is to assess and compare the performance of different VI Architecture data transfer modes and specific features that are available to higher-level communication software like MPI, in order to aid the implementor to decide which VI Architecture options to employ for various communication scenarios. Examples of such options include the use of send/receive vs. RDMA data transfers, polling vs. blocking to check completion of communication operations, multiple VIs, completion queues, and scatter capabilities of VI Architecture.*

1 Introduction

The performance of parallel applications running on clusters depends on the implementation of the nodes and the LAN or SAN (system area network) that acts as the communication system. Conventional communication systems like Fast Ethernet and legacy TCP/IP protocol stacks are widely used, especially for systems with limited budgets. But due to high software processing overhead (e.g. [13]), their communication performance is not sufficient for all application types. Applications with high communication frequency and small messages may suffer from the high latency of such solutions.

For a modern SAN with several Gbit/s hardware throughput, it is crucial to eliminate or minimize this software overhead along the critical path of send/receive communication operations. Appropriate techniques to reduce software overhead have been pioneered by fast communication systems like Active Messages [5], Fast Messages [12], and U-Net [6].

These techniques and systems form the basis on which the VI Architecture has been developed [2][4]. VI Architecture, described briefly in Section 2, is not intended for application programmers; instead, it provides a set of communication concepts and operations suitable to implement efficient higher-level communication libraries, for instance MPI.

There are a number of variants and specific features of the VI Architecture operations that the programmer of communication libraries has to judiciously choose from in order to achieve high communication performance in various scenarios. These choices include:

- use of send/receive vs. RDMA (remote direct memory access) data transfer modes;
- use of polling vs. blocking mechanisms to test the completion status of communication operations;
- the number of virtual interfaces (VIs) to use for a given connection;
- use of completion queues (CQs) to simplify testing completion of communication operations;
- use of advanced features of the VI Architecture like scatter facilities.

Often, the performance implications of design choices like these are not obvious. Therefore, we have performed a series of experiments in order to gain some insight into the performance behavior of these features.

The results of these experiments as well as our conclusions are presented in this article. The paper is organized as follows. Section 2 briefly reviews the important features of the VI Architecture standard. Section 3 introduces the cluster platforms for the experiments and the communication benchmarks used and presents the performance results. Section 4 addresses related research. Our conclusions are given in Section 5.

2 The VI Architecture

The VI Architecture standard [2][4] defines a set of concepts and primitives that allow an application to perform protected communication operations directly from the user level without involving the operating system (OS). VI Architecture provides point-to-point connections, the endpoints of which are implemented as *virtual interfaces* (VIs). A VI consists of a send queue, a receive queue, and a notification mechanism called doorbell.

The VI Architecture model comprises a VI consumer part and a VI provider part (Figure 1). The *VI consumer* consists of an application and a communication library like MPI that uses VI services via a lower level library (VI user agent in the figure). The *VI provider* consists of the NIC hardware and a kernel level driver component (VI kernel agent). This component is responsible for handling protection-related functions like opening and closing connections, for registering the application's memory regions (e.g., message buffers) with the NIC for communication purposes, and for address mapping of these message buffers. Communication requests like send, receive, and RDMA bypass the OS interface and interact directly with the NIC via a VI. Initiating a communication operation requires the VI consumer to prepare a *descriptor* of the work to be done, post it on the respective queue of a VI, and inform the NIC about the new communication request using the doorbell mechanism. The NIC will execute the communication request according to the information provided in the descriptor. After communication, the NIC will record completion status information in the status field of the descriptor.

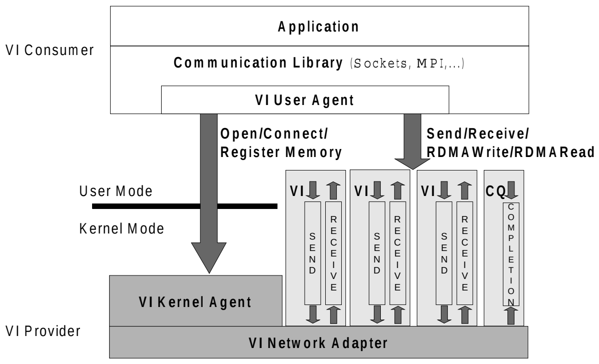

Figure 1: VI Architecture model

The VI Architecture standard offers a number of options to execute communication operations. First, an asynchronous *send/receive* model is provided. Initiating a communication and testing its completion status are separate actions, as described above. The second model is called *remote direct memory access* (RDMA) which denotes one-sided communication and is asynchronous as well. For example, a RDMA write operation transfers the message directly into the receiver's message buffer without notifying the receiver about the message transfer. The memory region (virtual address and memory handle) to be written to must have been disclosed to the writer prior to this operation.

A VI consumer (application) has two options to check the completion status of communication operations, namely via non-blocking and blocking calls. These calls are typically used by an application to *poll* on the completion of a communication (on user level), or to *wait and be interrupted* after its completion (which involves the OS), respectively. Clearly, the latter is more expensive, but allows the CPU to perform useful activity in lieu of busy waiting.

An application may own multiple VIs and

many communication operations may be outstanding on each VI. In such a case, the application may need to spend considerable time on checking its VIs for completed communication activities. To avoid this, VI Architecture specifies the concept of *completion queues*. CQs collect completion notifications for multiple VIs. They enable the VI consumer to check completion status in a single location and then directly access the completed descriptor in a VI.

Another feature that appears to be attractive to be used by communication software on top of VI Architecture is its *scatter capability*. A scatter operation can be effected in send/receive mode by specifying, for instance: (1) a single data segment in the send descriptor describing the length, memory handle, and virtual address of a single large send buffer; and (2) multiple data segments in the receive descriptor holding the lengths, memory handles, and virtual addresses of multiple smaller receive buffers. The NIC at the receiving end of the connection then autonomously scatters the data to the receive buffers, given that they provide sufficient aggregate space. This behavior may be more convenient for the programmer than to explicitly have a receiver process store the data into multiple destination locations.

3 Performance Evaluation

3.1 Platforms and Benchmarks

The performance experiments were run on two cluster platforms. One series of tests used two Intel Pentium^{R} 200 MHz-based machines equipped with the Intel 430HX chipset. The second series ran on two machines equipped with a Intel Pentium^{R} III 450 MHz processor and the Intel 440BX chipset. Both configurations have a Giganet cLAN (Cluster LAN) network, more precisely the GNN1000 cluster adapter which implements VI Architecture in hardware [7]. The cluster nodes run the Windows NT 4.0 (SP 3) operating system.

Latency and bandwidth figures for the different platforms and communication options were obtained using well-known microbenchmarks. In addition, in order to test more realistic communication patterns, we adopted one of the memory system and communication benchmarks proposed by Stricker et al. [15][9]. Specifically, we report the results for copy and communication operations with strided stores of the data into the target buffer which models matrix transposition, for instance.

3.2 Latency

Latency was measured using ping-pong tests (based on `viptest`) with four-byte packets. We evaluated the send/receive model with polling and blocking synchronization and the RDMA write model without immediate data. Synchronization in this context denotes how a VI consumer detects, or is notified of, the completion of a communication operation (send or receive). The ping-pong test in the RDMA write model is realized by having the receiving party check a write flag in its local memory that is set by the sending party upon completion of the RDMA write. Subsequently, the parties change their roles. Receive descriptor processing is not involved, therefore. Table 1 summarizes the round-trip latency results. Only the cases with identical synchronization styles at both the sending and receiving sides are shown.

The polling mechanism is significantly faster than the blocking mechanism on both platforms. This is due to the software overhead for system calls, context switching, and interrupt processing associated with the blocking synchronization style. The numbers indicate that this software overhead is about 20 μs for the Pentium^{R} and less than 10 μs for the Pentium^{R} III machine. The RDMA tests outperform their send/receive counterparts because descriptor processing does not occur on the receiver side. The RDMA write with blocking synchronization on the Pentium^{R} III is surprisingly fast. An analysis (under Linux) of the system calls performed by the VI library revealed that not all of the wait calls (`ioctls`) do become effective, on average yielding a low overhead value per wait operation.

Table 1: GNN 1000 round-trip latency for send/receive and RDMA write

Communication and synchronization model	Pentium	Pentium III
Send/receive, polling	15.3 μs	17.5 μs
Send/receive, blocking	84.7 μs	49.1 μs
RDMA write, polling	14.8 μs	16.5 μs
RDMA write, blocking	56.0 μs	24.9 μs

3.3 Throughput

We measured one-way throughput by having the sending process send out data at the maximum possible rate and having the receiver finally respond by a single confirmation message to complete the test. Again, results were obtained for both send/receive and RDMA write communication models. Figure 2 summarizes the results obtained for polling and blocking modes together with the traditional send/receive-style communication. The polling synchronization method is much faster than the blocking method on both machines. For small messages, the throughput on both machines is comparable because the performance is limited by the NIC hardware. For large messages, the Pentium^{R} III machine is approximately 200 MBit/s faster due to a better PCI chipset implementation; the advantage of the polling synchronization method becomes slightly smaller. Using the socket interface (TCP send/receive), the performance does not exceed approx. 110 MBit/s for both machines. The performance provided by the RDMA write model (Figure 3) is comparable to the traditional send/receive-style communication. Both methods can achieve approximately 830 MBit/s throughput on a Pentium^{R} III system.

3.4 Strided Copy Performance

Strided copy tests are based on the memory system performance benchmarks proposed by Stricker et al. [15]; these benchmarks have

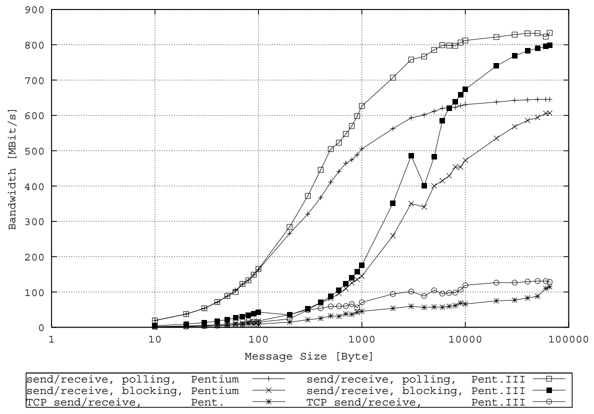

Figure 2: One-way throughput using the send/receive communication model

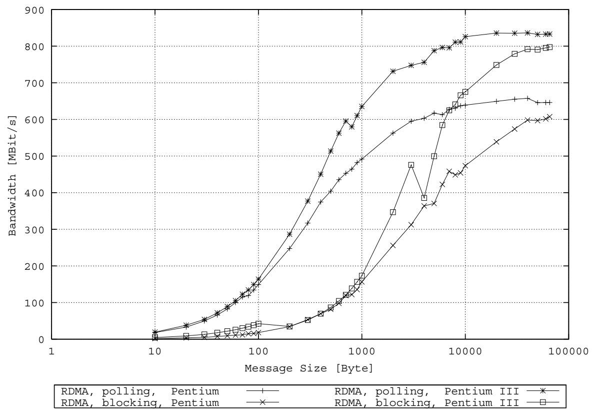

Figure 3: One-way throughput using the RDMA write communication model

also been adapted to test distributed shared memory systems and, thus, their underlying interconnects, e.g., Scalable Coherent Interface (SCI) and Cray T3D [9]. In the strided copy benchmark, contiguously stored blocks of double-precision floating-point (FP) numbers are spread over the target buffer, which may be located on a different node. The distance between two buffer entries depends on the organization of the buffer and is called the *stride*. Stride 1 simulates copying memory regions contiguously. Performance values for strides greater than 1 are influenced by the memory and cache organization and the properties of the communication system.

The strided copy algorithm was implemented in three different ways. First, the performance of the memory system was evaluated using a local version of the algorithm. A second form combines this test with a message passing step: the memory region is transferred over the network as a contiguous block and then stored away locally on the receiving node with the required stride. A third version combines both communication and strided copy into one communication request using the capability of the VI Architecture to distribute received data over a list of memory blocks: for each double-precision FP number to be received (as one element in a block that has been sent contiguously), a separate target address is specified in the receive descriptor; the NIC performs the strided store as part of the receive operation.

Figure 4 compares the throughput achieved for different strides on the Pentium^{R} III cluster. All tests handle blocks of approx. 8 kByte. (Buffer usage depends on the stride size.) Only the local strided copy version is influenced by the stride value. This version achieves a peak throughput in excess of 3 GBit/s, indicating that the local memory system is not a bottleneck. The second form of strided store achieves a peak throughput of 585 MBit/s. In contrast, bandwidth for communication integrating the strided store (version three) is nearly constant at the low rate of 22 MBit/s and is determined by the latency of the communication system; in this case, data segment processing by the NIC appears to be the bottleneck because each store of a double-precision FP value is encoded into one data segment of a receive descriptor. Clearly, an eight-byte value associated with a single data segment is too fine-grained to be processed efficiently by the NIC. It is therefore interesting to see if and how performance can improve when scattering applies to coarser-grained blocks of data. Figure 5 depicts the results of such an experiment, where the size of the individual blocks to be scattered by the NIC increases from 8 to 1024 bytes and 32 data segments are used in the receive descriptor; i.e., the overall amount of data transferred increases from 256 bytes to 32 kBytes. Increasing the data block size for scattering

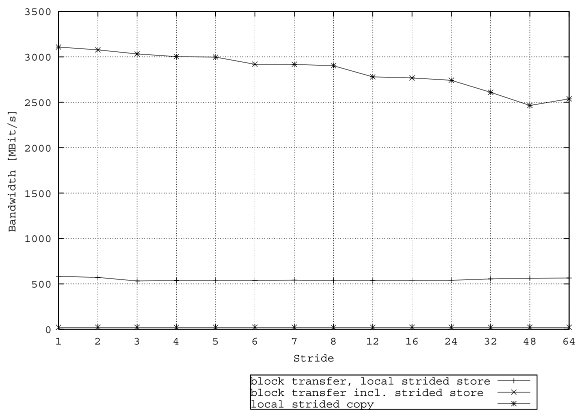

Figure 4: Block transfer with strided stores (on the Pentium^{R} III system)

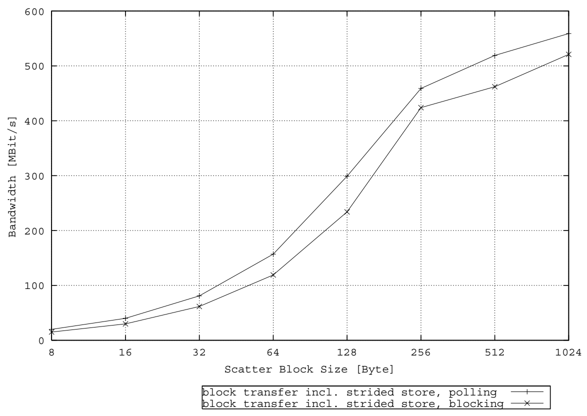

Figure 5: Automatic scattering with varying block sizes (on the Pentium^{R} III system)

rapidly increases throughput, up to a value of about 560 MBit/s under a polling synchronization scheme; the results for the blocking mode are slightly worse. Varying the stride size did not significantly impact performance. A comparison with Figure 2 shows that data scattering at the receiving end costs about one third of the maximum throughput even for large scatter blocks. Moreover, the scatter support provided by the VI NIC (third algorithm) is inferior to explicit scattering under program control (second algorithm); cf. Figure 4. Therefore, from a performance point of view, the use of the scattering facilities of the Giganet VI Architecture implementation cannot be recommended.

Table 2: Throughput results using multiple VIs and RR vs. CQ descriptor completion identification (on the Pentium^{R} machines)

N	Round Robin	Completion Queue
1	602 MBit/s	590 MBit/s
10	596 MBit/s	590 MBit/s
50	567 MBit/s	590 MBit/s
100	566 MBit/s	589 MBit/s

3.5 Completion Queues

Completion queues record completed descriptors in a single location and allow to directly branch to VI work queues associated with completed communication requests. This simplifies communication software and should lower the overhead of checking the completion status of communication operations when multiple VIs are in use. To analyze the efficacy of this concept, we compared the use of CQs to a hand-crafted round robin (RR) descriptor processing strategy. The test program creates N connections (on N VIs) between two communicating processes and distributes communication requests randomly among them. Table 2 presents the results obtained using 2048-byte packets on Pentium^{R} machines.

The CQ processing overhead is independent of the number of VIs. This overhead is approximately 2% compared to the RR strategy with one VI. The advantage of CQs becomes more obvious when more than 10 VIs are active. The overhead caused by the RR strategy increases with the number of connections. CQs should be considered when implementing communication software with multiple active connections. When only a small number of connections is maintained, CQs might be used because their performance impact is small.

4 Related Work

A number of investigations of VI Architecture implementations and, more specifically, of the Giganet cLAN implementation have been reported in the literature. Prototype implementations (e.g., over Myrinet) and their performance results are described in [4], [1], and [11], for instance. Higher-level communication layers (e.g., TCP, RPC, and MPI) over Giganet's cLAN and their performance behavior are introduced in [14] and [3], among others. Performance results for various communication layers and applications are also available on the Giganet Web site [8]. In contrast to these analyses, our work goes into more details of various VI Architecture features and their performance implications. Our work also extends the performance results of [9] by providing data for the strided copy benchmarks (direct-deposit data transfers to remote memory in the terminology of [9]) for the Giganet cLAN interconnect.

5 Conclusions

In this paper, we investigated the performance implications of using various communication models and specific features of VI Architecture, more precisely its implementation incorporated in the Giganet cLAN (GNN1000 adapter cards). The results can aid an implementor of higher-level communication software in deciding which features to use or avoid.

The send/receive and RDMA write communication methods provide the same performance in terms of throughput, given that the same synchronization method for testing completion of communication operations is used (polling vs. blocking, i.e., interrupt-based notification). In terms of latency, however, our results indicate that the choice of the synchronization method is very important: polling-based completion checking avoids the overheads of interrupt processing and context switching of the blocking synchronization (less than 10 μs and about 20 μs, respectively, on our two platforms), yielding round-trip latency figures several times lower than with blocking. It must be noted, though, that the choice of polling vs. interrupt-based notification involves several other aspects and trade-offs (e.g., [10]).

The use of multiple VIs and a completion queue (CQ) appears to be advantageous. Our results indicate that multiple VIs provide slightly higher performance than a single VI; the use of a CQ generates an overhead of only about 2% for the case of a single VI and pays off when more than 10 VIs are active. Given that a CQ can simplify communication software, its use can be recommended.

The scatter capability of the send/receive model that we investigated using a more realistic and complex strided copy benchmark turned out to yield disappointing throughput, more than an order of magnitude worse than the simple method to transfer a contiguous block and scatter it locally on the receiving node. Increasing the size of the data blocks to be scattered, rapidly increases the performance of automatic scattering, yet does not reach the maximum throughput that can be achieved when scattering is performed explicitly under program control. The superiority of the simple, explicit method is consistent with the results reported in [9].

References

[1] P. Buonadonna, A. Geweke, D. Culler. An Implementation and Analysis of the Virtual Interface Architecture. *Proc. High Performance Networking and Computing Conference 1998 (SC'98)*, Orlando, FL, Nov. 7-13, 1998. http://www.supercomp.org/sc98/papers/.

[2] Compaq Computer Corp., Intel Corporation, Microsoft Corporation. *Virtual Interface Architecture Specification.* Version 1.0. December 1997. http://www.viarch.org.

[3] R. Dimitrov, A. Skjellum. An Efficient MPI Implementation for Virtual Interface (VI) Architecture-Enabled Cluster Computing. *Proc. Third MPI Developer's Conference.* March 1999.

[4] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, March/April 1998.

[5] T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. *Proc. 19th Int'l. Symp. on Computer Architecture.* ACM Press 1992.

[6] T. von Eicken, A. Basu, V. Buch, W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. *Proc. 15th ACM Symposium on Operating System Principles.* ACM Press 1995.

[7] Giganet, Inc. *GNN1000 High Performance Host Adapter User Guide.* September 1998.

[8] Giganet, Inc. *cLAN Performance.* http://www.giganet.com/products/performance.htm.

[9] C. Kurmann, T. Stricker. A Comparison of Three Gigabit Technologies: SCI, Myrinet and SGI/Cray T3D. In: H. Hellwagner, A. Reinefeld (eds.), *SCI: Scalable Coherent Interface. Architecture and Software for High-Performance Compute Clusters*, LNCS 1734, Springer Verlag 1999.

[10] K. Langendoen, R. Bhoedjang, H. Bal. Models for Asynchronous Message Handling. *IEEE Concurrency*, April-June 1997.

[11] National Energy Research Scientific Computing Center (NERSC). *M-VIA: A High Performance Modular VIA for Linux.* http://www.nersc.gov/research/FTG/via.

[12] S. Pakin, V. Karamcheti, A. Chien. Fast Messages: Efficient, Portable Communication for Workstation Clusters and MPPs. *IEEE Concurrency*, April–June 1997.

[13] C. Papadopoulos, G. M. Parulkar. Experimental Evaluation of SUNOS IPC and TCP/IP Protocol Implementation. *IEEE/ACM Transactions on Networking* **1**(2), April 1993.

[14] H. V. Shah, C. Pu, R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. *Proc. CANPC'99* (Third Int'l. Workshop on Communication and Architectural Support for Network-Based Parallel Computing), LNCS 1602, Springer Verlag 1999.

[15] T. Stricker, T. Gross. Global Address Space, Non-Uniform Bandwidth: A Memory System Performance Characterization of Parallel Systems. *Proc. of the ACM Conference on High Performance Computer Architecture (HPCA-3)*, San Antonio, TX, February 1997.