VIA Communication Performance on a Gigabit Ethernet Cluster.*

Mark Baker[1] Paul A. Farrell[2] Hong Ong[3] Stephen L .Scott[4]

[1] School of Computer Science,
University of Portsmouth,
Portsmouth, UK, PO1 2EG.
Mark.Baker@computer.org
[2] Dept. of Mathematics and Computer Science,
Kent State University,
Kent, OH 44242, USA.
farrell@mcs.kent.edu
[3] Dept. of Mathematics and Computer Science,
Kent State University,
Kent, OH 44242, USA.
hong@mcs.kent.edu
[4] Oak Ridge National Laboratory
Oak Ridge, TN 37831-6367, USA
scottsl@ornl.gov

Abstract. As the technology for high-speed networks has evolved over the last decade, the interconnection of commodity computers (e.g., PCs and workstations) at gigabit rates has become a reality. However, the improved performance of high-speed networks has not been matched so far by a proportional improvement in the ability of the TCP/IP protocol stack. As a result the Virtual Interface Architecture (VIA) was developed to remedy this situation by providing a lightweight communication protocol that bypasses operating system interaction, providing low latency and high bandwidth communications for cluster computing. In this paper, we evaluate and compare the performance characteristics of both hardware (Giganet) and software (M-VIA) implementations of VIA. In particular, we focus on the performance of the VIA send/receive synchronization mechanism on both uniprocessor and dual processor systems. The tests were conducted on a Linux cluster of PCs connected by a Gigabit Ethernet network. The performance statistics were collected using a local version of NetPIPE adapted for VIA.

1 Introduction

The advent of high performance microprocessors coupled with high-speed interconnects has made clusters an attractive platform for parallel and distributed

* This work was supported in part by NSF CDA 9617541, NSF ASC 9720221, NSF ITR 0081324, by the OBR Investment Fund Ohio Communication and Computing ATM Research Network (OCARnet), and through the Ohio Board of Regents Computer Science Enhancement Initiative

computing. There are many research institutes and academic departments involved in building low cost Beowulf-class clusters to fulfill their computing needs at a fraction of the price of a traditional mainframe or supercomputer.

Interconnecting the nodes in a cluster requires network hardware and software that is scalable, reliable, and provides high bandwidth and low latency communications. Gigabit Ethernet [1], among the other high-speed networks, can in principle provide the required performance needed for cluster computing. However, the improved performance of Gigabit Ethernet is not realized at the application layer of the network. This is due primarily to the overheads incurred in communicating between processor, memory, and I/O subsystems that are directly connected to a network. In particular, this communication overhead is caused by the time accumulated when messages move through different layers of the TCP/IP stack in the operating system. The source of these overheads is multiple memory copies and use of the operating system for receiving and transmission of packets [2]. In the past, the end-to-end Internet protocol overhead did not significantly contribute to the poor network performance, since the latency was primarily dominated by the speed of the underlying network links. However, Gigabit Ethernet technologies coupled with high-speed processors now mean that the overhead of the Internet protocol stack is the dominant bottleneck in the performance of this technology.

Many research projects have been proposed to address network performance issues. Examples of these projects include U-Net [3], Fast Messages [4], and Active Messages [5]. More recently, the Virtual Interface Architecture (VIA) [6] has been developed to standardize efforts in this area. VIA defines mechanisms that will bypass the intervention of the operating system layers and avoid excess data copying during sending and receiving of packets. This effectively reduces latency and lowers the impact on bandwidth. Since the introduction of VIA, several software and hardware implementations of VIA have become available. Examples include Giganet [7], ServerNet-II [8], Myrinet [9], and M-VIA [10].

This paper evaluates and compares the performance of both hardware and software implementations of VIA technology on a Linux-based cluster connected by a Gigabit Ethernet Network. The layout of this paper is as follows: Section 2 gives an overview of the VIA technology. Section 3 gives an overview of NetPIPE and the implementation of NetPIPE-VIA. Section 4 presents the test environments used. Section 5 discusses the results from our performance tests. Finally, the conclusions and future work are presented in Section 6.

2 VI Architecture Overview

The VI Architecture is an industry trade effort to provide low latency and high bandwidth message-passing support over a System Area Network (SAN). Figure 1 illustrates the VIA design. The VIA design consists of two major components: the *VI Provider* and the *VI Consumer*.

The *VI Provider* is a combination of a media dependent *VI Network Interface Card* (NIC) and the *VI Kernel Agent*. The VI Kernel Agent is a software compo-

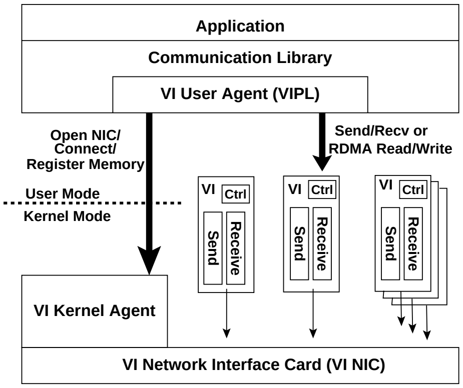

Fig. 1. VI Architecture

nent that includes the device driver for the VI NIC and a set of kernel routines needed to perform privileged operations such as connection management and memory registration on behalf of applications.

The *VI Consumer* is an application process that communicates using the *VI Provider Library* (VIPL) primitives. The software component that implement the VIPL is known as the *VI User Agent*. It provides a protected and directly accessible interface, know as *Virtual Interface* (VI), to the network interface. Each VI represents a communication endpoint and pairs of such VIs can be connected to form a communication channel for bi-directional point-to-point data transfer. A VI Consumer can have multiple VIs.

Each VI is associated with a send queue and a receive queue (also known as work queue). For data transmission, the sender and the receiver post packet descriptors to its work queue. A mechanism known as a *doorbell* is used to notify the VI Kernel Agent that a descriptor has been added to a work queue. A doorbell is a control register associated with each work queue which is mapped into the virtual address space of the application process that owns the VI.

A VI Consumer must register the memory region of its packet descriptors and data buffers with the VI Kernel Agent before being used for communication. Memory registration takes the expensive operations of mapping memory and doing virtual to physical translation out of the critical performance path. As a consequence, this eliminates the need for copying data from the user buffer space to the kernel buffer space, which is typically used in traditional communication models.

The VI Architecture supports send/receive and remote direct memory access (RDMA) read/write types of data movements. The current revision of the VI architecture specification defines the semantics of a RDMA Read operation but does not require that the network interface support it.

3 NetPIPE Benchmark and Implementation of NetPIPE-VIA

NetPIPE [11] is a network protocol independent performance evaluation tool developed by Ames Laboratory. The design of NetPIPE consists of a protocol independent driver, and a set of well defined communication APIs.

The device independent driver implements a ping-pong like program which increases the transfer block size from a single byte to large blocks until transmission time exceeds 1 second. Specifically, for each block size c, three measurements are taken for block sizes $c - p$ bytes, c bytes and $c + p$ bytes, where p is a perturbation parameter with a default value of 3. This allows examination of block sizes that are possibly slightly smaller or larger than an internal network buffer.

The communication APIs implement the protocol specific module. Currently, NetPIPE supports TCP, PVM, and MPI communication protocols. For our performance evaluation, we implemented a VIA communication protocol module for NetPIPE.

The set of NetPIPE communication APIs needed for the protocol specific module includes those for establishing a connection, closing a connection, sending and receiving data, and performing synchronization. We implemented all the communication APIs using the VIPL library.

To keep the implementation simple, NetPIPE-VIA creates a pair of VI endpoints per connection. A fixed number of send and receive packet descriptors are pre-allocated and each descriptor has a fixed size of registered (pinned) memory, which is equal to the maximum data buffer size supported by the VI Provider. The descriptors are chained together to form a ring. To send a message, NetPIPE-VIA gets a descriptor from the send ring and posts the descriptor to the send queue. After the completion of a send operation, the descriptor is inserted back into the ring. VIA requires packet descriptors to be posted on the receive queue before any message arrives. Otherwise, the message will be lost. Therefore, NetPIPE-VIA pre-posts all the receive descriptors before the reception of messages occurs. Whenever a packet arrives, it gets a descriptor out of the receive queue, processes the packet, and posts the descriptor back to the receive queue again.

For each measurement, the protocol independent driver determines the size of the data block either linearly or exponentially depending on a user specified command line option. Hence, the memory buffer for a data block of size c is dynamically allocated at run time. In order to achieve zero-copying and avoid the extra overhead of pining and unpining the memory buffer for each data block, NetPIPE-VIA pre-allocates and pre-registers a pool of memory buffers. All memory requirements of the independent protocol driver are satisfied from this memory pool. This also keeps the memory management in NetPIPE-VIA simple.

When transmitting a large data block, the message will be fragmented in order to fit into a descriptor's data segment. This implies that multiple descriptors are needed to either transmit or receive large messages. Consequently, flow control is required to prevent the sender from overflowing the receiver's pre-posted receive descriptors. NetPIPE-VIA implements a simple flow control scheme. On the sender side, it continues to transmit until either the entire data block c is sent or the number of sends reaches the maximum number of pre-posted descriptors of the receiver. For the latter case, the sender waits for a "continue" message from the receiver before sending more packets. On the receiver side, it continues to receive packets until either the entire data block c is received or it reaches the maximum number of pre-posted descriptors. If the receiver runs out of pre-posted descriptors, it stops receiving and waits for all receive requests to complete. Then, it sends a "continue" message to inform the sender to continue to send more packets.

4 Testing Environment

The testing environment for collecting VIA performance data was a cluster of 32 dual processor Pentium III 450MHz PCs using the Intel 440BX chipset. The PCs were equipped with 256MB of 100MHz (PC100) SD-RAM. The PCs were connected together through a Foundry FastIron 10/100/1000Mbps switch using SysKonnect *SK-9821* NICs. Four of these PCs were also connected together through a Giganet cLAN5000 switch using cLAN1000 NICs. In addition, two PCs were connected back to back using Packet Engine *GNIC-II* NICs. All the NICs were installed in 33MHz/32bit PCI slots. The PCs were running the Red Hat 6.2 distribution with the 2.2.16 Linux kernel. For testing the software implementations of VIA, M-VIA v1.0 was installed. All performance results were collected using the NetPIPE-VIA benchmark tool.

5 Performance Results

In VIA terminology, the term synchronization is used to refer to the process by which a VI Consumer detects, or is notified of, the completion of a communication request. For each of the Gigabit Ethernet NIC mentioned earlier, we evaluated the performance characteristics of the polling and blocking synchronization schemes for a uniprocessor system and a dual processor system. In particular, we present and compare the point-to-point latency and throughput performance of both hardware and software implementation of VIA. For completeness, we have also included the TCP performance[5]. Latency is measured by taking half the average round-trip time for a 1 byte transfer. The throughput rate is calculated from half the round-trip time for a data block of size c bytes.

5.1 VIA Latency Discussion

Table 1 summarizes the latency performance for the various NICs on uniprocessor and dual processor systems. One obvious observation is that the TCP latency, on both the uniprocessor and dual processor systems, is at least 50% higher

[5] The LAN emulation driver *lanevi* was used to collect TCP performance for the cLAN 1000 NIC

Table 1. Latency Performance in μ secs

	Uni-Processor			Dual Processor		
	Socket Read/Write	VI Send/Recv		Socket Read/Write	VI Send/Recv	
NIC	TCP	Poll	Block	TCP	Poll	Block
CLAN	58	9	9	116	10	45
GNIC-II	57	19	19	94	42	42
SK-9821	63	29	29	101	51	53

than the VIA latency regardless of which synchronization schemes are used. This highlights that VIA can, in practice, deliver the low latency needed for communication intensive applications. In general, the latency of hardware implemented VIA (cLAN) is at least 40% lower than software implemented VIA (GNIC-II and SK-9821). With hardware support, the VI Kernel agent is able to offload much of the processing to the network card resulting in lower processing overhead.

For the uniprocessor system, the latency for both synchronization schemes is almost identical for all NICs. For a dual processor system, the latency using the polling scheme is slightly less than the blocking scheme. However, it is more obvious for the cLAN NIC. The primary cause for such a difference is due to the implementation of the doorbell mechanism in the VI Kernel agent. cLAN's doorbell mechanism is supported directly in hardware as a true memory mapped doorbell. cLAN's VIPL uses the `ioctl` system call to provide the time-sensitive services. On the other hand, M-VIA's VIPL uses `fast_traps` system calls to emulate the doorbell mechanism. In Linux kernel 2.2.16, there is a big variance when timing both the `ioctl` and `fast_trap` calls. However, when we take the average of the overhead for each system call, it yields lower overhead per wait operation on a dual-processor system then on a uniprocessor system. However, this effect is less on a uniprocessor system resulting in little or no difference in latency.

For M-VIA, the latency for both VIA synchronization schemes on the dual processor system is higher when compared to the uniprocessor system. This is because M-VIA uses the Linux `spinlock` primitives to protect the data structure of each VI. For an SMP, the overhead of using `spinlock` primitives to perform mutual exclusion across CPUs is higher than in the uniprocessor case. This overhead comes from disabling interrupts when making `spinlock` calls to lock the VI data structure. Although it is expensive, the `spinlock` mechanism is safe as compared to other implementations. On the other hand, the latency for cLAN on the dual processor system is only about 1μ second higher than the uniprocessor system. This is because cLAN uses the Linux `pthread` primitives to provide protected access to the VI data structure. It is relatively cheap in terms of CPU cost and does not require the calling thread to disable interrupts.

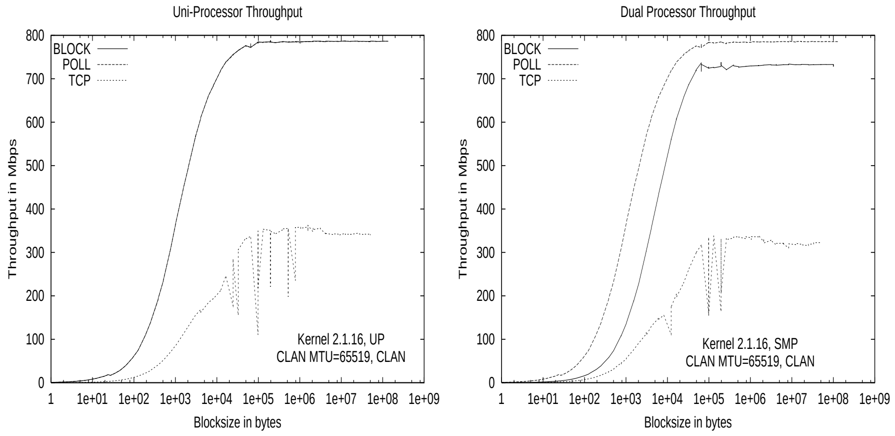

Fig. 2. cLAN: Send/Recv Mode

5.2 VIA Throughput Discussion

Figure 2 shows the communication performance using cLAN. For a uniprocessor system, cLAN achieves maximum throughput of approximately 362Mbps for TCP and 788Mbps for both polling and blocking synchronization schemes. For a dual processor system, cLAN achieves maximum throughput of approximately 339Mbps, 738Mbps, and 768Mbps for TCP, the blocking scheme, and the polling scheme, respectively. The advantage of using the polling scheme is more obvious for the dual processor system. Recall that cLAN uses a software layer to emulate TCP transmission. The actual transmission still uses the cLAN VI Kernel agent and the VIA-aware hardware. One would expect TCP to perform well in such a configuration. However, we observe that there were many severe dropouts in the cLAN TCP graph especially for larger data blocks. This could be interpreted as the overhead involved in maintaining the VI's work queues, the cost of the memory copy from the user's data buffer to the packet descriptor's data buffer, and the overhead of the cLAN's TCP emulated device driver (lanevi) in fragmenting the data block to 1500 bytes (the default maximum transmission unit (MTU) specified in the Ethernet standard 802.3).

Figure 3 shows the communication performance using the GNIC-II. For a uniprocessor system, the GNIC-II achieves maximum throughput of approximately 258Mbps for TCP and 510Mbps for both polling and blocking synchronization schemes. For a dual processor system, the GNIC-II achieves maximum throughput of approximately 261Mbps for TCP and 273Mbps for the polling scheme and blocking scheme, respectively. In the dual processor system, the VIA performance drops drastically whereas the performance of TCP increases slightly.

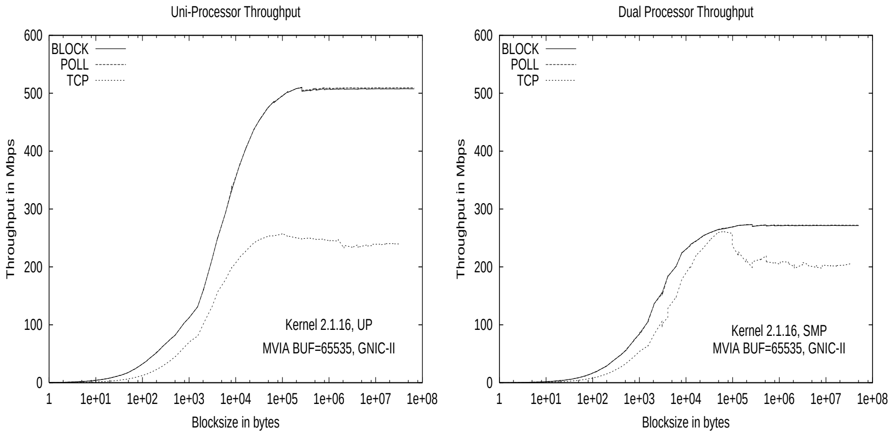

Fig. 3. GNIC-II: Send/Recv Mode

Figure 4 shows the communication performance using the SK-9821 NICs. For the uniprocessor system, SK-9821 achieves maximum throughput of approximately 296Mbps for TCP and 520Mbps for both polling and blocking synchronization schemes. For dual processor system, SK-9821 achieves maximum throughput of approximately 260Mbps for TCP and 269Mbps for polling scheme and blocking schemes, respectively.

The poor VIA throughput performance on the dual processor system is related to the implementation of the doorbell mechanism in M-VIA as explained above. In general, the throughput performance using the polling mechanism is slightly better (about $1 - 3\%$) than the blocking mechanism for all NICs under both systems. However, this effect can only be observed when the data block size is large, e.g $> 4Mbytes$.

5.3 Effect of Hardware MTU on Throughput Performance

In M-VIA, the GNIC-II and SysKonnect NICs hardware MTU is 1500 bytes. This is because the Gigabit Ethernet standard still limits the MTU to 1500 bytes. In [12], it has been observed that a larger MTU improves TCP throughput.

To confirm that hardware MTU will also improve VIA performance, we tested the SK-9821 NIC using an MTU of 9000 bytes. Since the Foundry switch does not support MTU greater than 1500 bytes, we connected two PCs back to back using the SK-9821 NICs. Figure 5 shows the VIA communication performance as well as the TCP performance. For the uniprocessor system, the SK-9821 achieved maximum TCP throughput of approximately 574Mbps with latency of 54μ secs. The M-VIA performance attains maximum throughput of roughly 632Mbps using the blocking scheme and 648Mbps using the polling synchronization scheme. The latency for both schemes is approximately 24μ secs. This represents an increase of a factor of 2 for TCP throughput and approximately a 20% increase

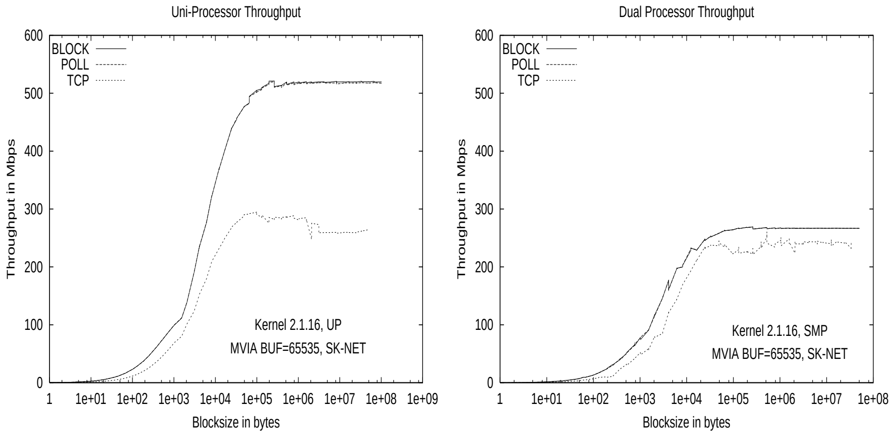

Fig. 4. SK-9821: Send/Recv Mode

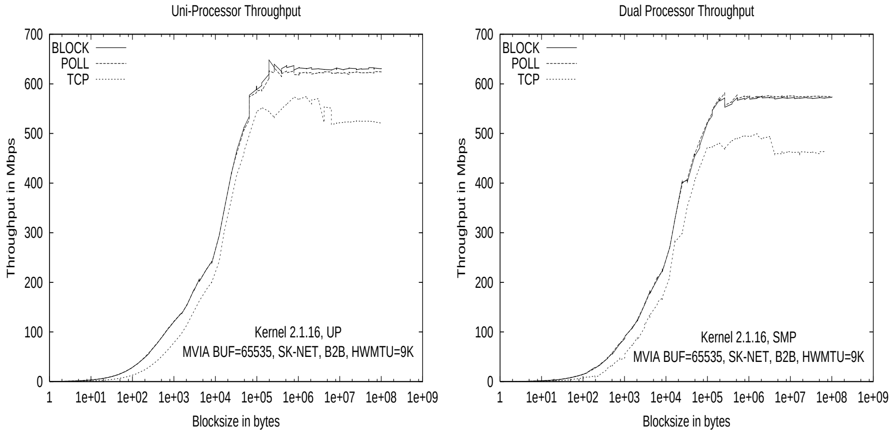

Fig. 5. SK-9821: Send/Recv Mode with MTU=9000

for the VIA throughput when compared to using MTU of 1500 bytes. Moreover, the latency has also decreased.

For the dual processor system, the SK-9821 achieves a maximum TCP throughput of approximately 504Mbps with latency of 93μ secs. The M-VIA maximum throughput is approximately 535Mbps for both the polling and blocking schemes. The latency of the polling scheme has increased by 3μ secs. However, the latency of the blocking scheme remains unchanged. As compared to MTU of 1500 on the dual processor system, the M-VIA throughput has improved by approximately 200Mbps.

6 Conclusion

This paper has presented the performance of both hardware and software implementation of VIA on uniprocessor and dual processor systems. From the performance figures, we have verified that the VI architecture can give higher throughput and lower communication latency to applications. We have also confirmed that eliminating the TCP protocol stack provides higher throughput performance. VIA technology can take advantage of the VIA-aware hardware. For instance, cLAN achieves higher throughput and lower latency when compared to a software-only implementation such as M-VIA. The effectiveness of polling and blocking synchronization schemes depends on the implementation of the doorbell mechanism. Further investigation of other VIA features such as multiple VIs and RDMA is warranted. Moreover, we suspect that using multiple VIs on a SMP system may yield higher throughput.

References

1. Gigabit Ethernet Alliance: Gigabit Ethernet Overview, `www.gigabit-ethernet.org` (1997)
2. R. P. Martin, A. M. Vahdat, D. E. Culler, T. E. Anderson: Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. Proc. $24th$ Annual Int'l Symp. on Computer Architecture (ISCA). (1997)
3. T. Von Eicken, A. Basu, V. Buch, W. Vogels: U-NET: A User Level Network Interface for Parallel and Distributed Computing. Proc. of the 15^{th} SOSP. (1995)
4. S. Pakin, M. Lauria, A. Chien: High Performance Messaging on Workstation: Illinois Fast Message (FM) for Myrinet. Proc. of Supercomputing'95. (1995)
5. Richard P. Martin, Amin M. Vahdat, David E. Culler, Thomas E. Anderson: Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. ISCA 24. (1997)
6. Compaq Computer Corp., Intel Corp., Microsoft Corp: Virtual Interface Architecture Specification version 1.0, `developer.intel.com/design/servers/vi/` (1997)
7. Giganet Inc.: cLAN Performance, `www.giganet.com/products/performance.html`
8. E. Speight, H. Abdel-Shafi, J. K. Bennett: Realizing the Performance Potential of the Virtual Interface Architecture. Proc. of Supercomputing'99. (1999)
9. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, W. Su: Myrinet - A Gigabit per second Local Area Network. IEEE Micro. (1995)
10. National Energy Research Scientific Computing Center: M-VIA: A High Performance Modula VIA for Linux, `www.nersc.gov/research/FTG/via/`
11. Q. O. Snell, A. R. Mikler, J. L. Gustafson: NetPIPE: Network Protocol Independent Performance Evaluator. Ames Lab., Scalable Comp. Lab., Iowa State. (1997)
12. Paul A. Farrell, Hong Ong: Communication Performance over a Gigabit Ethernet Network. IEEE 19^{th} Proc. of IPCCC. (2000)

This article was processed using the LaTeX macro package with LLNCS style