Semi-User-Level Communication Architecture

Dan Meng, Jie Ma, Jin He, Limin Xiao, Zhiwei Xu
Institute of Computing Technology
Chinese Academy of Sciences
P.O. Box 2704
Beijing 100080, P.R.China
{*md, majie, jhe, xlm, zxu*}*@ncic.ac.cn*

Abstract

This paper introduces semi-user-level communication architecture, a new high-performance light-weighted communication architecture for inter-node communication of clusters. Different from traditional kernel-level networking architecture and user-level communication architecture, semi-user-level communication architecture removes OS kernel from its message-receiving path while reserves an OS trapping on its message-sending path. No interrupt handling is needed. This new communication architecture doesn't support user-level access to network interface. It provides good portability, security, and support for heterogeneous networking environment and usage of large memory. Semi-user-level communication architecture has been implemented on a SMP workstation cluster system called DAWNING-3000, which is interconnected through Myrinet. Communication performance results are given and overhead distribution is analyzed.

1. Introduction

With high scalability and performance/cost ratio, clusters have been becoming a mainstream architecture of high-performance computing systems. Most ASCI machines are clusters except ASCI Red. Nowadays, clusters have been widely used in the areas of technical computing, Internet service, and database applications. However, to achieve good application results, the cluster communication subsystem must deliver high performance[1].

With the rapid growth of network hardware performance, communication software becomes the bottleneck of performance improvement. Traditional kernel-level networking architecture, like TCP and UDP, places all protocol processing into OS kernel. As a result, the critical path of a message from its source to destination has included expensive operations, such as several crossings of the operating system boundary, plenty of data copying at both ends, and interrupt handling.

In response to this performance problem, user-level communication architecture has been proposed. It entirely removes the operating system from the critical communication path, providing direct user-level access to network interface and avoiding excessive data copying. These features reduce the software overhead significantly, bringing communication latency and bandwidth close to the hardware limits[3][13].

However, direct user-level access to network interface results in several problems as follows:

- Portability. User-level access to network interface need the operating system support, that is, virtual memory mapping. Although many popular operating systems, such as Linux and Solaris, provide mmap system call to implement the mapping of memory and registers on network interface into application process user space, some not. IBM AIX is such an example. Any user-level communication architectures implemented on Linux or Solaris can not be ported to AIX. Because of lack of virtual memory mapping support, user-level communication architecture can not be implemented on AIX.

- Protection. Since user-level architectures give users direct access to the network interface, memory and registers on the network interface are visible to user processes. OS can not be relied on to check network transfers. This results in security problem. As superservers, clusters are being widely used in Internet service and database applications. Multi-user and multiprogramming must be support, and security must be guaranteed. In user-level communication architecture, virtual-memory system is used to restrict each user access to a different part of network interface memory. It is the network interface's responsibility to check user's requests and validate all user-specified parameters. To

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

do so, the network interface must cache some important data structures and keep consistent with host. This therefore increases the complexity of communication protocol and processing workload of the network interface.

- Heterogeneous network environment. In user-level communication architecture, difference of the network interface may result in different implementation of communication library. So it is difficult to support heterogeneous network environment.

- Address translation and usage of large memory. The use of DMA transfers between host and network interface memory introduces virtual-to-physical address translation. Because user processes are untrusty in user-level communication architecture, network interface is involved. VMMC and U-Net implement a page-table cache on the network interface. Since network interfaces are usually equipped with only a small amount of memory and relatively slow processors, the address translation efficiency will be affected, especially when each node of clusters provides large capacity of memory. In large SMP or NUMA clusters, each node is equipped with large capacity of memory. The capacity increase of memory on the network interface can not keep pace with host, so the efficiency of the cache will decrease.

In order to implement high-performance light-weighted secure communication protocol on IBM AIX, an operating system that does not support virtual memory mapping, we design the semi-user-level communication architecture. It removes OS kernel from message-receiving path. There is an OS trapping on message-sending path. Access to network interface can only be done through kernel. User-level access to the network interface is not allowed. Network transfer checking and address translation of DMA-able buffer is carried out in host OS kernel. Therefore semi-user-level communication architecture provides a better way to resolve the problems discussed above.

The semi-user-level communication architecture has been implemented on a cluster of SMP workstations called DAWNING-3000 interconnected with Myrinet. The communication software is called BCL. Its communication performance has been evaluated and latency overhead distribution has been analyzed. The performance analysis shows that, compared with user-level communication architecture, communication latency increases by approximately 22% while bandwidth is the same.

This paper is organized as follows. In section 2, we introduce DAWNING-3000 and give an overview of BCL. In section 3, we present the principle of semi-user-level communication architecture and make comparison with other two architectures. In section 4, we discuss the implementation of BCL on DAWNING-3000. In section 5, we give communication performance of BCL and analyze the affection of communication architecture to communication performance, especially communication latency. In the final two sections, we present related work and draw conclusion.

2. Background

2.1. DAWNING-3000 Superserver

Dawning3000[15] is a distributed memory high-performance computer with SMP cluster architecture and message passing programming model. The system provides 70 nodes with 280 microprocessors, 168 Gbyte memory, 630 Gbyte internal disks, and 3 Tbyte external disks, achieving 403.2 Gflop/sec peak performance. It has 64 computing nodes and 6 service nodes, and each node is a 4 CPUs SMP system running IBM AIX4.3.3.

The computing node is used for high performance computing and data processing applications and uses 375 MHz Power3-II microprocessor from IBM. The service node is used for I/O service, login service, network service, and database service adopting PowerPC RS64-III microprocessor technique from Bull. Usually the service node needs to expand the disk storage and the network interface. The mapping of applications to nodes could be flexible according to the user's requirement.

Nodes of DAWNING-3000 are interconnected with system area network, internal network, external network, and detection network. The system area network (SAN) has two choices: Myrinet or custom-designed nWRC 2-D mesh. Myrinet M2M-PCI64A NICs and M2M-OCT-SW8 switches are used. It provides 1.28Gb/sec per channel bandwidth. The key technique of nWRC 2-D mesh is a routing chip called nWRC1032. The chip uses the wormhole routing technique and works at 40MHz. It has 6 data channels with 32 bits data for each path. The network interface (NIC), called PMI960, is a 33 MHz, 32 bits PCI adapter with an Intel i960 microprocessor as the DMA engine and communication co-processor. We design the NIC interface with the routing chip and the control program running on the i960. The link in the 2-D mesh is a kind of 2 inches cable from AMP.

As depicted in Figure 1, the communication software of DAWNING-3000 are divided into three layers, that is, low-level light-weighted communication protocol BCL (Basic Communication Library), middle-level communication library EADI-2 (Extended Abstract Device Interface-2), and high-level parallel programming software including PVM, MPI, HPF, JIAJIA[8], and a parallelizing compiler AutoPar.

The foundation of DAWNING-3000 communication software is BCL. Like BIP[9] and VIA, BCL is a light-

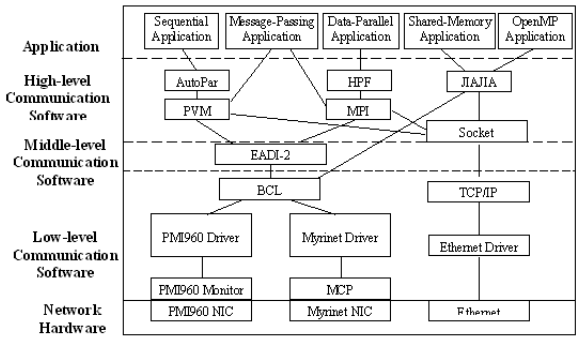

Figure 1. DAWNING-3000 Communication Software Architecture

weighted high-performance communication protocol. It consists of three components: firmware on NIC, device driver in host OS kernel space and communication library in user space. We will present BCL in detail in coming section. DAWNING-3000 implements PVM on a middle-level form communication library EADI-2. ADI[7] is a standard defined to support the implementation of MPI. EADI-2 extends ADI-2 to fulfil the requirements of PVM implementation. EADI-2 is implemented as an independent library, not a component of MPI code as before. Compared with implementing PVM directly using BCL, this method simplifies the implementation of PVM. PVM can inherit the optimization in EADI-2 and is easy to port to different platforms and low-level communication protocols.

2.2. Overview of BCL

BCL is a VIA-like low-level high-performance communication interface. BCL defines port for inter-process communication. Each process can create only one port to communicate with others. The pair of node number and port number is the unique identifier of a process. Each port has a sending request queue, a receiving buffer pool and the corresponding event queues (sending event queue and receiving event queue). Receiving buffer pool is composed of several channels.

When a process (sender) wants to send a message to another process (receiver), the sender should compose a send request and put it into the send request queue. Destination is specified by its node number, port number and channel number in the request. The receiving channel should be ready before the message arrived the receiving side. A receive event will be generated when a complete message arrived. On the send side, a send event will be put into the send event queue when a sending operation is over.

There are three types of channels: system channel, normal channel and open channel. System channel is designed to transfer small messages. Each process has one system channel. Every system channel has a buffer pool, which is initialized when the process starts. It is organized as a FIFO queue. When a short message is arrived, it is automatically put to the first free buffer in the buffer pool. The incoming message will be discarded if there is no free buffer in the pool. After the receiver gets the message, the buffer will be returned to the buffer pool.

Normal channel is designed to transfer messages with rendezvous semantics. Each port has a set of normal channels. The receiving process needs to post a data buffer in its user space and bind the buffer to the receive channel before the sending process start transmission. A receive event will be generated when a message has been received.

Open channel is designed to perform RMA operations. Each port has a set of open channels. Once a user-specified buffer is bound to an open channel, other processes are able to read/write memory areas within the corresponding buffer.

3. Principle of Semi-User-Level Communication Architecture

Figure 2 illustrates the principle of the semi-user-level communication implemented by BCL over Myrinet. BCL defines port for inter-process communication. Similar to VIA and InfiniBand[2], each port of BCL consists of a pair of work queues: send request queue and receive request queue. Each work queue has a corresponding completion queue. A receiving request must be posted before corresponding sending operation begins.

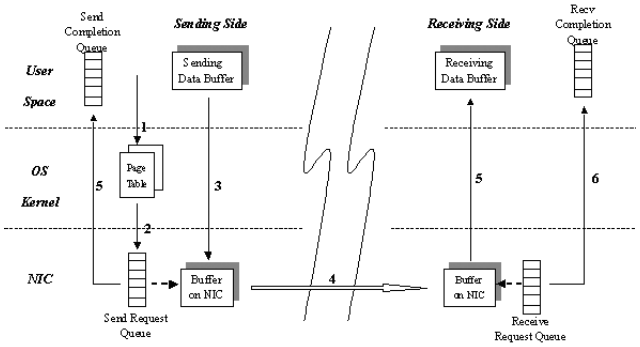

Figure 2. Semi-User-Level Communication Architecture in BCL

When a sender process issues a send request, it traps into OS kernel. In kernel space, it searches pin-down buffer page table and completes virtual-to-physical address translation and pin-down operation for sending data buffer if search-missing. Then the sender fills in send request queue

located in NIC memory with send request descriptor, including physical addresses and lengths of all physical pages consisted of the sending data buffer, and return back to its user space. Once finding the send request descriptor, MCP running by LANai on NIC initiates data transfer from sending data buffer in user space to intermediate buffer on NIC directly using DMA. Then the data is transferred to destination NIC buffer through Myrinet physical link. MCP on destination NIC DMAs the arriving data to receiving data buffer in receiver process' user space based on the buffer information in receive request descriptor posted before. Finally, MCP on both sides DMA send/receive completion events into corresponding completion queue in user space to notify the sender/receiver, which poll on the completion queue using BCL primitives.

Table 1 compares these three communication architectures in three important aspects: numbers of OS trapping and interrupt handling and location accessing NIC on communication critical path.

Table 1. Comparison of Three Communication Architecture

	Kernel-Level Communication	User-Level Communication	Semi-User-Level Communication
OS Trapping	2	0	1
Interrupt Handling	2	0	0
Location Accessing NIC	Kernel Space	User Space	Kernel Space

Compared with user-level communication, semi-user-level communication has only one more kernel trapping on its critical communication path. Although its communication latency maybe a little bit more than user-level communication, semi-user-level communication has following advantages:

1. Portability. It can be implemented on all commercial UNIX operating systems without need of special system support for virtual memory mapping (mmap).

2. Security. In semi-user-level communication, NIC can only be accessed in host kernel space. Important data structures, such as pin-down buffer page table, are stored in OS kernel. Network transfer checking is carried out in kernel. This guarantees system security.

3. Support for heterogeneous network environment. Because NIC is transparent to process user space, binary code written in BCL or PVM and MPI implemented on top of it can run on any combination of networks supporting BCL protocol. Applications written in BCL need not be recompiled. This is especially useful for applications running over a cluster of clusters[11]. Currently, BCL support Myrinet and custom-designed nWRC 2-D mesh.

4. Efficient and secure way for virtual-to-physical address translation. In semi-user-level communication architecture, address translation is done in host kernel, and the physical addresses of a message buffer are transferred to NIC through PIO. This simplifies the design and implementation of communication protocol. It is an efficient way to implement address translation, especially for SMP or NUMA nodes with tens or even hundreds of CPUs and large capacity of memory.

4. Implementation on DAWNING-3000

BCL is a low level communication software used on DAWNING-3000. It has two versions. One is implemented on Myrinet and the other is implemented on Dnet. The upper levels can make full use of the hardware performance via BCL. BCL supports point to point message passing. All other collective message passing should be implemented in the higher level software.

4.1. Semi-user level Architecture

One of the most important requirements to BCL is that it can be easily ported to different platforms, including AIX/RS6000 and Linux/PC.

4.1.1 Basic architecture

BCL is divided to three layers: user level library, host kernel module and control program in NIC (MCP, Message Control Program).

Figure 3 is the internode communication architecture of BCL. BCL library provides a set of APIs. Applications linked with BCL library can use these APIs to communicate with each other. In fact these APIs are only the covers of some ioctl() syscall subcommands provided by BCL kernel module.

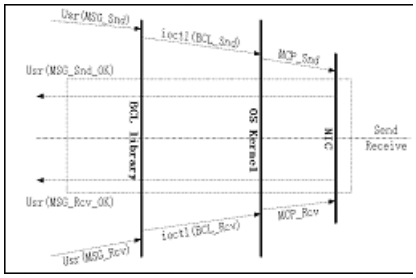

Figure 3. Inter-node communication in BCL

BCL kernel module posts operation requests to the request queues on NIC's local memory. These requests include the sending requests and other requests, such as initialize/close communication port request and initialize

channel requests. Kernel module also implements some functional operations, which need to be executed in the kernel environment. Such operations include the host memory pin/unpin operation and host virtual memory address to bus memory address conversion.

The NIC has a local processor, which can control the NIC to transfer packets by using its serveral DMA engines. In BCL, MCP controls all the inter-node packet transfers. MCP completes a sending operation by reading send request in the card's local memory, sending/receiving message with DMA engines and informing user process the completion.

By the approch, the message receive/send finish event will be directly sent to user process space. So the user process need not trap into kernel mode to check the status of BCL messages. However in order to make the security check, the BCL message sending and making ready for message buffer still need switch into kernel mode. The mechanism used by BCL combined the merits of user-level and kernel-level communication protocols. We named the mechanism BCL uses the "semi-user level" protocol.

4.1.2 SMP support

Because the DAWNING-3000 super server is a cluster of SMP nodes, BCL provides another communication path for different processes in the same node.

There are several ways to move data from one process to another process within one node. The traditional way is to move data as the same way as between nodes. Process A first transfers the data to NIC (Network Interface Circuit) by DMA. Then NIC transfers them back to process B. While the memory copy bandwidth is much higher than DMA bandwidth, a good solution is to use shared memory to implement intra-node communication. Process A first copies the data to a shared memory area. Then process B copies them out. Another way is to move data directly from user space to user space.

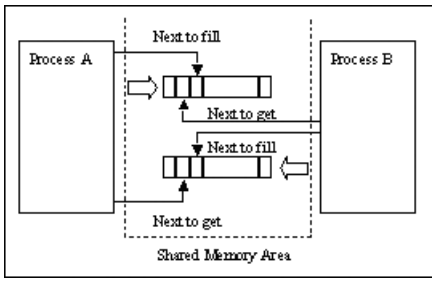

Figure 4. Intra-node communication in BCL

BCL uses shared memory based intra-node communication. The internal buffer queue is used to transfer message from one process to another process within a node. This queue consists of a list of buffers. Each pair of processes has two queues. (Figure 4). To ensure the message sequence, BCL uses the sequential number to decide whether the operation should continue or not. Shared memory based intra-node communication needs an extra memory copy than the direct memory copy solution. BCL reduced the extra overhead by using the pipeline message passing technique.

4.1.3 Portability

All the system-related operations are hidden in the kernel module, so the user level library and control program in NIC need no modification when porting to different platforms. Besides, Applications, which used the BCL user level library, can be ported to different platforms without modification. Also, the difference of heterogeneous network environment is hidden in the kernel module.

4.2. Security mechanism

Kernel-level communication protocols, such as TCP/IP, the communication main path involves OS kernel trapping and interrupts handling. All these overheads cause high end-to-end message latencies.

User-level communication such as allows applications directly access the network interface cards (NIC) without operating system intervention on both sending and receiving sides. Messages are transferred directly to and from user-space by the network interface card while observing the traditional protection boundaries between processes. It reduces the communication overhead by avoiding kernel traps on the data path.

However, in commodity systems, security is the most important requirement. User level communication protocol exposes all the control data structures to user space so that any mistake or malice operation will cause the system broken. Kernel level communication protocol protects the important data structures in kernel space.

To provide a secure communication approch, BCL forces the communication request from applications to pass some necessary security checks in kernel module and control program layers. Because a malicious user can bypass the security check in user library by using some debug utilities, BCL basicaly delays those checks in kernel module.

The parameters checked include application process ID, communication buffer pointer, and communication target and so on. With this safeguard mechanism BCL assures all processes using it will safely send and receive messages, never destroy kernel data structures.

Besides, shared memory based intra-node communication can make the system more reliable. Any mistake or malice operation during a directly inter-process memory access can cause the target process crashed. By using shared

memory, the sender process can only destroy the shared area. It won't affect other processes' space. This guarantees the independence of each process.

5. Performance Analysis

The semi-user level communication protocol BCL is implemented on DAWNING-3000 Super Server. The processor overhead for sending a message is approximately 7.04μs, and for receiving a message is approximately 1.01μs. No trapping into kernel environment makes the receiving operation much faster.

All the tests are done on DAWNING-3000, which consists of 70 IBM270 workstations. Each node is a 4-way 375MHz Power3 SMP, which is running IBM AIX4.3.3. Myrinet M2M-PCI64A NICs are used on each node. These nodes are interconnected by M2M-OCT-SW8 switches.

5.1. Inter-node transmission and reception timings

The timeline for transmission of a BCL message is shown in Figure 5. The timings were obtained using the timing functions provided by operating system and using repeated executions of parts of the code. The timeline corresponds to the transmission of a BCL message. A timing analysis of the BCL code shows that the processor overhead required to push a message into the network is approximately 7.04μs and another 0.82μs is required to complete the sending operation. Since the PCI IO performance is low on our test bed (about 0.24μs to write a word to NIC and 0.98 to read a work from NIC), filling sending request consumed more than half of the time.

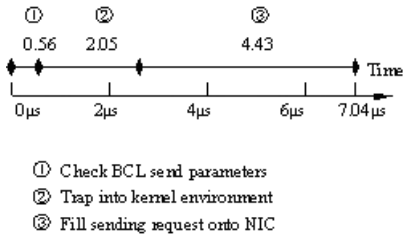

Figure 5. Transmission timeline for a BCL message

Figure 6 shows the timeline for reception of a BCL message. Not trap into kernel environment makes the reception operation much faster. It only needs to check the data structures in user environment and to decide whether a new message is arrived.

Combined the Transmission and reception timeline, Figure 7 shows the one-way latency timeline of a 0-length BCL

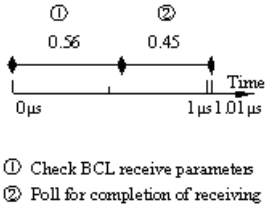

Figure 6. Reception timeline for a BCL message

message. About one third of the overhead is used to transfer message from NIC to network (stage 4) because NIC control program need to process the reliable protocol and perform re-transmission when timeout. Stage 5 and 6 are inevitable since the physical layer need transfer message via network and put data from card to host memory.

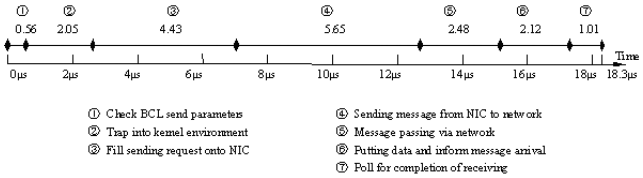

Figure 7. One-way latency timeline for a 0-length BCL message

Compared with user level communication protocol, BCL added stage 2 and stage 6 to the communication path. Extra overhead required in semi-user level communication protocol is approximately 4.17μs, which is about 22% of the total message transfer time. This extra overhead won't affect bandwidth since only 4.17μs is added to 898μs transfer time when transfer a 128KB-length message.

5.2. One-way Latency and Bandwidth

The first tests are raw BCL point to point communications. Latency and bandwidth are measured on our DAWNING-3000. Both inter-node and intra-node communications are tested. Figure 8 and Figure 9 show the inter-node communication performance. The minimal latency is 2.7μs within one node and 18.3μs between nodes. The bandwidth is 391MB/s within one node (with the affect of cache) and 146MB/s between nodes. And the half-bandwidth is reached with less than 4KB message.

Table 2 shows the comparison of three protocols. GM is a message-based communication system for Myrinet, which is designed and implemented by Myricom. GM doesn't provide special support for SMP. Only inter-node communication performance data are given in the table. The range for

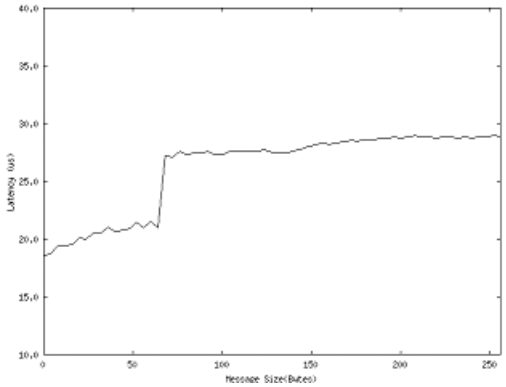

Figure 8. Inter-node latency of BCL

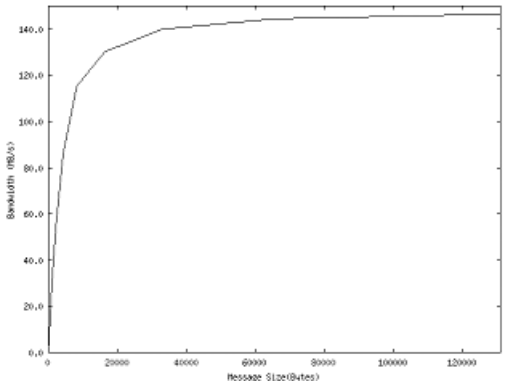

Figure 9. Inter-node bandwidth of BCL

Table 2. Comparison of different communication protocols

Protocol	Latency(μs)		Bandwidth(MB/s)	
	Intra-node	Inter-node	Intra-node	Inter-node
GM	—	13.37—21	—	>140
AM-II	3.6	27.5	160	32.8
BIP-SMP	1.8	5.7	160	126
BCL	2.7	18.3	391	146

GM's one-way short-message latency on a wide variety of hosts is from 13.37 to 21μs. And the peak bandwidth is over 140MB/s. BCL reaches almost the same performance and provides a more reliable (using kernel level communication) and more complex (special support for SMP) protocol.

AM-II (Active Messages) is similar to a remote procedure call (RPC) mechanism. Compared with AM-II, BCL has a better latency in both intra-node and inter-node communication. It is meaningless to compare the bandwidth of these two protocols since AM-II needs an extra memory copy when transfer a message while BCL doesn't. BCL reaches a much higher bandwidth.

BIP (Basic Interface for Parallelism) is a low level message passing system developed by the Laboratory for High Performance Computing in Lyon, France. It has a very low latency. But it doesn't provide the functionality of flow control and error correction. Its bandwidth is lower than that of BCL.

Table 3. Performance of BCL and MPI/PVM over BCL

	Latency(μs)		Bandwidth(MB/s)	
	Intra-node	Inter-node	Intra-node	Inter-node
BCL	2.7	18.3	391	146
MPI over BCL	6.3	23.7	328	131
PVM over BCL	6.5	22.4	313	131

Table 3 shows the performance of MPI and PVM over BCL. The minimal latency of MPI over BCL is 6.3μs within one node and 23.7μs between nodes. The bandwidth is 328MB/s within one node (with the affect of cache) and 131MB/s between nodes. The minimal latency of PVM over BCL is 6.5μs within one node and 22.4μs between nodes. The bandwidth is 313MB/s within one node (with the affect of cache) and 131MB/s between nodes.

5.3. Discussion

It is evident from the above performance figures and tables that the semi-user level architecture reduces the reception overhead much more. It not only avoids the overhead to trap into kernel environment, but also avoids interfering of NIC's operation. I/O device will have a low performance when lots of I/O accesses occur during a DMA operation. Message sending operation also benefits from the semi-user level architecture. Compared with user level communication, only 22% extra overhead is added to the communication path.

The one-way latency timeline shows that the operation on NIC consumes more than half of the overhead. The reason is that BCL performs data checking and guarantees reliable transmission in the on-card control program. Half of the overhead is inevitable to transfer messages between nodes. And the other 5.65μs is to perform the reliable transmission. To reduce the protocol overhead is a way to improve the communication performance.

Another time consuming operation is to fill the sending request onto NIC. This is limited by the I/O performance of the PCI bus. A good motherboard can improve the I/O performance heavily. Host CPU frequency limits the parameter checking and trap operation's overhead. A faster CPU will reduce these overheads.

Bandwidth of BCL is almost reaching the hardware limitation. The peak performance of Myrinet switch is around 160MB/s. BCL reaches 91% of the physical bandwidth. The extra overhead added by semi-user level is only 4.17μs which is about only 0.4% of the total time when transfer a

128KB-length message. It doesn't affect the final performance of bandwidth.

6. Related Work

In general, there are three distinct communication architectures: kernel-level networking, user-level messaging and semi-user-level architecture. Traditional kernel-level networking architecture, like TCP and UDP, brings expensive overhead due to place all protocol processing into operating system kernel. User-level messaging has been studied in a number of projects, such as U-Net[6], Shrimp[5], Fast Message[10] , HP Hamlyn[14] and Parastation[12]. To approach higher communication performance close to the hardware limits, the user-level communication architecture usually implements direct access to the custom network interface, by dint of the support of some specific operating systems. To a certain extent, the high performance by user-level messaging is obtained at the cost of sacrificing portability and protection. Direct access to network interface not only needs the operating system support, but also tightens the coupling between communication software and network hardware. In addition, removing kernel from the communication pathespecially the message-sending path, drastically tears the security shelter of the operating system.

Due to the use of DMA transfers between host and network interface memory, address translation becomes a crucial problem in any messaging architectures. Normally, operating system do not export virtual-to-physical mapping to users, so users cannot pass physical addresses to the network interface. The common approach is that user pages are pinned and unpinned dynamically so that DMA transfers can be performed directly to those pages. The main implementation problem is that the network interface needs to know the current virtual-to-physical mapping of individual pages. However, network interfaces do not store information for every single virtual page since they are usually equipped with only a small amount of memory and their processors are relatively slow. Different methods are implemented to solve this problemBIP[9] and LFC[4] let users pin their pages and obtain the physical address of these pages from a kernel virtual-to-physical translation module, VMMC-2[5] and U-Net let the network interface cache a limited number of valid address translations which refer to pinned pages, and BCL just lets a simple kernel module carry out all the address translation functions.

Supporting heterogeneous network environments is important requirement for communication architectures. PM2[11] is a high performance communication middle layer, which can supports both Myrinet and Ethernet. Binary code written in PM2 may run on any combination of those networks without re-compilation. This characteristic is just as our BCL.

7. Conclusion

This paper describes a new cluster communication architecture, called semi-user-level communication. With this scheme, there is an OS trapping on the message-sending path. All accesses to the network interface can only be done through the kernel. User processes cannot directly access the kernel space or the network interface. Network transfer checking and address translation of DMA-able buffer is carried out in the kernel. This scheme brings about several benefits:

1. Portability. It can be implemented on all commercial UNIX operating systems without the need of special system support for virtual memory mapping (mmap).

2. Security. In semi-user-level communication, the network interface can only be accessed in host kernel space. Important data structures, such as the pin-down buffer page table, are stored in OS kernel. Network transfer checking is carried out in kernel. Thus system security is enhanced. This is especially important in a superserver system, where the underlying communication system needs to support not only message passing in scientific computing applications, but also high-speed TCP/IP communication and cluster file systems, in a multi-user, multi-process environment.

3. Support for heterogeneous network environment. Because the network interface is transparent to process user space, binary codes written in BCL, or PVM and MPI programs implemented on top of it, can run on any communication networks supporting the BCL protocol. Applications written in BCL need not be recompiled.

4. Efficient and secure address translation. In the semi-user-level communication architecture, address translation is done in host kernel. This simplifies the design and implementation of the communication protocol.

The semi-user-level communication architecture has been implemented on a cluster superserver called Dawning 3000. This 270-CPU, 403-Gflop/s system is in production use today. On the Dawning 3000, the semi-user-level communication architecture achieves a latency is 2.7μs within one node and 18.3μs between two nodes. The bandwidth is 391MB/s within one node and 146MB/s between two nodes. Compared to a fully user-level communication scheme, we estimate that the semi-user-level communication scheme increases the communication latency by about 22%, and has little performance penalty for bandwidth.

References

[1] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin. User-space communication: A quantitative study. *In Proceedings of SC'98*, Nov 1998.

[2] I. T. Association. Infiniband architecture specification release 1.0. *Available online at http://www.infinibandta.org*, Oct 2000.

[3] R. Bhoedjang and et al. Design issues for user-level network interface protocols on myrinet. *IEEE Computer*, 31(11):53–60, Nov 1998.

[4] R. Bhoedjang, T.Ruhl, and H. Bal. Efficient multicast on myrinet using linklevel flow control. *In Int. Conf. On Parallel Processing, Minneapolis, MN*, Aug 1998.

[5] M. Blumrich, C. Dubnicki, E. Felten, and K. Li. Virtual memory mapped network interfaces. *IEEE Micro*, Feb 1995.

[6] V. Eiken, A. Basu, V. Buch, and W. Vogels. U-net: A user-level network interface for parallel and distributed computing. *In Proceeding of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[7] W. Gropp and E. Lusk. The second-generation adi for the mpich implementation of mpi. *Available online at http://www-unix.mcs.anl.gov/mpi/mpich*.

[8] W. Hu, W. Shi, and Z. Tang. Jiajia: An svm system based on a new cache coherence protocol. *In Proceedings of the High-Performance Computing and Networking Europe 1999 (HPCN'99)*, pages 463–472, April 1999.

[9] L. Prylli and B.Tourancheau. Protocol design for high performance networking: A myrinet experience. *Tech. Rpt. 97-22, LIP-ENS Lyon*, Jul 1997.

[10] S.Paking, Karamcheti, and A.Chien. Fast message (fm): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Concurrency*, 5(2), April - June 1997.

[11] T. Takahashi, S. Sumimoto, A. Hori, H. Harada, and Y. Ishikawa. Pm2: A high performance communication middleware for heterogeneous network environments. *In Proceedings of Supercomputing 2000*, 2000.

[12] T. Warschko, W. Tichy, and C. H. Herter. Efficient parallel computing on workstation clusters. *In Proceedings of Supercomputing'95*, 1995.

[13] R. Weber and et al. A survey of messaging software issues and systems for myrinet-based clusters. *In Proceedings of PDCP'99, Special Issue on High Performance Computing on Clusters*, May 1999.

[14] J. Wilkes. An interface for sender-based communication. *Tech. Rep. HPL-OSR-92-13, Hewlett-Packard Research Laboratory*, Nov 1992.

[15] Z. Xu, N. Sun, D. Meng, and W. Li. Cluster and grid superservers: The dawning experiences in china. *In Proceedings of the IEEE Cluster 2001 International Conference, to appear*.