Fast Message Passing for High Performance Computation in Workstation Clusters*

OU Xin-ming, SHEN Jun, ZHENG Wei-min

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

E-mail: oxm@263.net; zwm-dcs@tsinghua.edu.cn

http://www.tsinghua.edu.cn

Received January 28, 2000; accepted April 20, 2000

Abstract: The message passing system is crucial to the computational performance of workstation clusters. With the fast growth in the performance of networking device, the unnecessary overhead in traditional TCP/IP protocol has become the bottleneck of communication speed. To fully exploit the communication potential of the fast hardware, this paper introduces FMP, a new communication protocol specifically designed for Myrinet. FMP, which consists of a network part and a local part, is a concise protocol. The physical layer is assumed to be error free and thus the protocol becomes extremely simple and highly efficient. Compared with TCP/IP, the performance of FMP is much better. In this article the essential techniques used in the protocol are discussed and some results are presented.

Key words: parallel computing; workstation cluster; message passing interface; communication protocol; Myrinet

With the fast development of the networking equipment, workstation cluster has become a popular platform for parallel computing because of its high performance-price rate[1]. People hope that time-consuming programs that previously run on MPPs can now be executed on networks of workstations or PCs in approximately the same amount of time. However, this goal cannot be easily achieved. The most popular implementation of Message Passing Interface (MPI) that is used in workstation clusters takes TCP/IP as its underlying communication protocol. As we know, TCP/IP is designed for wide-range networks, e.g. the Internet. For the unreliability in long-distance communication and the heterogeneity of the hosts and networks between two communicating ends, it incorporates a great deal of concerns over error-detecting, error- correcting and flow control[2]. But in networks of workstations, hosts are often homogeneous and near to each other. The reliability of underlying hardware is so high that it is even impossible for an error to occur during the execution of a parallel program. For example, the Myrinet network adapters and switch have an error rate of 10^{-15}. This means that for an error to happen, one needs to wait for approximately 30,000,000 years. In such systems, the error detecting and correcting overhead in

* This project is supported by the National High Technology Development Program of China under Grant No. 863-306-ZD06-03-2 (国家863高科技发展计划基金). **OU Xin-ming** was born in 1975. He got his M.S. degree in 2000 from the Department of Computer Science and Technology of Tsinghua University. His research interests include parallel and distributed systems, high performance computing and advanced networks. **SHEN Jun** was born in 1969. He got his Ph.D. degree in 1999 from the Department of Computer Science and Technology of Tsinghua University. His research interests include parallel and distributed systems, high performance computing and advanced networks. **ZHENG Wei-min** was born in 1945. He is a professor in the Department of Computer Science and Technology of Tsinghua University. His research interests include computer architecture and parallel and distributed system computing.

TCP/IP is completely a waste of time. Moreover, TCP/IP is embedded in the kernel of Unix operating system. Access to the communication functions is through system calls, which entails time consuming context switch. This also affects the performance of message passing functions based on TCP/IP. With the rapid growth of the bandwidth in physical layer, the bottleneck of speed has migrated from hardware to software.

Our goal is to design a simplified communication protocol, which exploits the maximum communication potentials of the fast networking equipment, thus greatly reducing the execution time of parallel programs on networks of workstations.

1 System Overview

Our system architecture is illustrated in Fig. 1. Eight SUN Ultra2 workstations are interconnected with a Myrinet Switch. Each node is an SMP with two 200MHz UltraSPARC-I CPUs, 256MB memory and a Myrinet network adapter card on Sbus. The Myrinet network is a high-speed switch network developed by Myricom Inc[4]. It incorporates the techniques in both LAN and MPP and has a maximum transfer rate of 1.28Gbit/s. Each adapter card has a 32-bit CPU and two interfaces for sending and receiving. The cards are interconnected by Myrinet switch, forming a cluster.

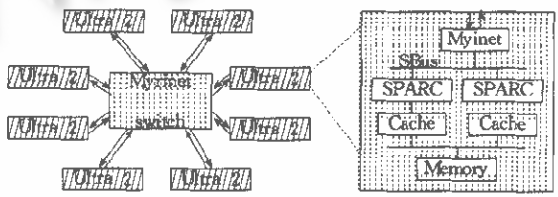


Fig. 1 System architecture

Our new protocol for this cluster system is called FMP (Fast Message Passing). In the following section the essential techniques used in it will be discussed in detail.

2 Crucial Techniques

2.1 Buffer management

FMP is an asynchronous protocol in which connection need not be set up before data transferring. Buffer management in such kinds of protocols is a very important issue. If we let all processes on a host share a single receive buffer, different processes on the host must be synchronized when accessing the buffer. The race among receiving processes will make the system extremely inefficient, especially in polling mode, where each process scans the buffer continuously for new messages. In FMP we use another policy. Each process has its own receive buffer and all processes share a single send buffer, as illustrated in Fig. 2.

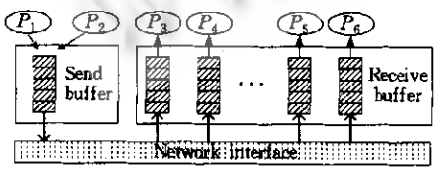


Fig. 2 Buffers in FMP

Buffers in FMP are queues. When a process wants to send a message, it just puts the message on the tail of the send queue. So the race among sending processes is not severe. This is why we use a single send buffer for all processes. The Myrinet control program (MCP) runs on the CPU of the network adapter card. It is responsible for transmitting the data in send buffer out and putting the incoming messages in the appropriate receive buffer. The process associated with each receive queue will check the buffer and remove messages from it. Now there is no need to synchronize different receiving processes because each one has its own receive queue. However, we must synchronize different sending processes because only one send queue exists. As stated above, this little overhead is trivial.

2.2 Polling or interrupt

When a process wants to receive a message that has not arrived yet, two choices exist. It can either scan the receive buffer continuously until its arrival or go to sleep and let the operating system to wake it up when the desired data come. The former is called polling mode and the latter is called interrupt mode, because when the process sleeps, the operating system will schedule other processes to run and the hardware must interrupt CPU to wake up the sleeping process.

The advantage of interrupt mode is obvious. It can increase concurrency and thus save CPU time. The disadvantage is not so easy to see. Both sleeping and waking up involve time-consuming context switch, which greatly affects the communication latency. The latency graph in section 3 proves it. Under most circumstances in parallel computing, each process will exclusively occupy a CPU. So generally speaking, the polling mode is more advantageous than interrupt mode because it is faster. Sometimes there may be more than one process sharing a CPU. For example, in PVM each machine must have a daemon process running on it. Since the daemon does not do much work, it usually shares CPUs with other computing processes. In such cases, interrupt mode is more suitable because CPU time can be saved for other processes when desired message has not arrived. To provide user with the flexibility of choosing one of these two modes, we implement two interfaces for each.

2.3 DMA or PIO

For a message to be sent, it must be copied to the buffer on the network adapter card. There exist two ways to do that. The first approach is programming I/O (PIO). The process puts the data directly into I/O port, byte by byte. Actually this is done as memory copy, because I/O addresses have been mapped to the process's virtual-memory space by the network driver[6]. The another approach is direct memory access (DMA), in which data are transmitted between memory and I/O automatically. DMA is much faster than PIO since CPU need not interfere with the transmission and time is spared for computation. But before each DMA operation, some preparation work must be done. In the case of small messages, the time used is even longer than data transmitting time. So DMA does not suit small messages. Figure 3 is a graph showing the transmitting properties of PIO and DMA.

Given these different properties, we choose different methods according to the size of messages—— PIO for short ones and DMA for large ones. If DMA is chosen, the sending process must first write the message into a kernel area, since kernel space cannot be swapped out, as required by DMA. Then the CPU on the Myrinet Card will start a DMA operation and transmit the message. To make the DMA operation and the network transmission parallel, we employ a pipelining technology. A message is cut into small pieces. The network card can start a network transmission as soon as a piece has been copied into it by DMA. Pipelining improves performance considerably, as shown in Fig. 4.

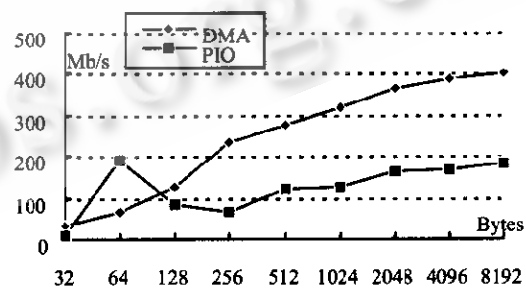


Fig. 3 Transmitting properties of PIO and DMA

2.4 Avoiding deadlock

Since FMP is an asynchronous and connectionless communication protocol, it is possible that one process sends a message to another process before the destination process is ready to receive it. Under such circumstances, the arriving message will occupy the receiving buffer until it is fetched. If the buffer is exhausted by too many unexpected messages, other processes will not be able to send data to it. This may lead to deadlock. For example, in

Fig. 5, process 0 on host 0 sends a large message to process 1 on host 1 and process 1 also sends a large message to process 0. The receiving buffers for both processes are exhausted before the messages are completely sent. However, neither process will receive the message from its buffer until it has finished sending its own one. Thus a deadlock occurs.

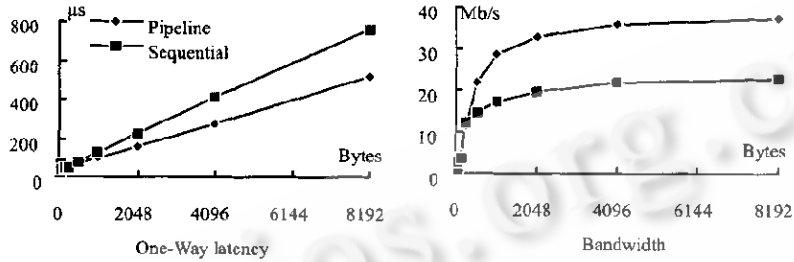


Fig. 4 Comparison of pipelining mode and sequential mode

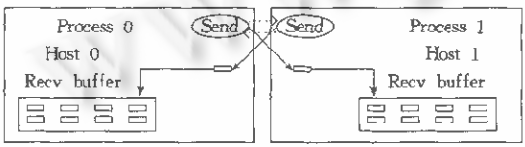


Fig. 5 Deadlock situation

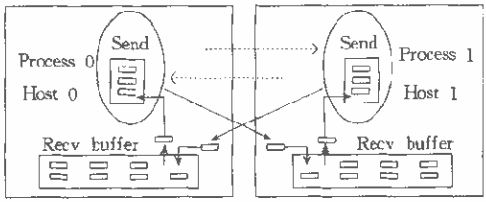


Fig. 6 Avoiding deadlock

To avoid such kind of deadlock, we move some messages from the receiving buffer into a temporary area in the process's address space when communication is blocked. Compared with the size of the buffer at network adapter card, the memory space is so large that it can even be thought as infinite. Thus we can prevent the deadlock incurred by buffer exhaustion (Fig. 6).

2.5 Local communication

In a network of SMP workstations, communicating processes may happen to reside on the same host. In such conditions, message passing between two processes is carried out using the local communication part of FMP protocol, LM_FMP (the network part of FMP is called NC_FMP). Local communication in FMP is through the memory shared among different processes. Sending and receiving of messages are implemented as various memory copy functions. Each process on a host has a channel number assigned to it. And each channel is associated with a message-header queue. The header of a local message destined to a channel is put into the tail of the corresponding header-queue. If the message is small enough, the whole message is embedded in the header. Otherwise a pointer in the header indicates the place where the body of the message is stored (Fig. 7).

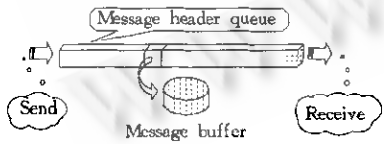


Fig. 7 Structure of LM-FMP

How the body of large messages should be stored is a question worth discussing. A simple solution is to allocate a continuous memory block from the shared memory. This makes it easy to copy data into or from the buffer. But the memory allocation algorithm in this policy is difficult to devise and tends to create lots of small fragments that are hard to utilize.

In LM_FMP, we use another buffer management approach. The body of a large message is segmented into fixed-length units and stored in a linked list. Each unit is a list element. Since the length of each unit is the same, the

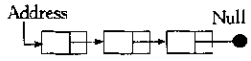
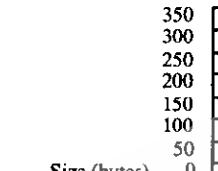


Fig. 8 Body of a large message

buffer allocation algorithm becomes much easier and no external fragments will be created. We need only to write a set of new memory-copy functions to accommodate this data structure of discontinuous memory space (Fig. 8).

3 Performance of FMP

We tested the performance of FMP protocol and compared it with TCP/IP. First, we measured the point-to-point communication latency and bandwidth of FMP by calling FMP sending and receiving functions directly in user's program. Then, we tested the overall performance of MPI based on FMP by a series of benchmarks.



Size (bytes)	1	4	8	16	32	64	128	256	512	102
NC_FMP(p)	12	13	14	16	19	18	21	35	49	72
NC_FMP(i)	40	40	41	42	47	45	49	63	77	101
TCP/IP	220	221	220	222	225	235	258	268	281	322

3.1 Point-to-point performance of FMP

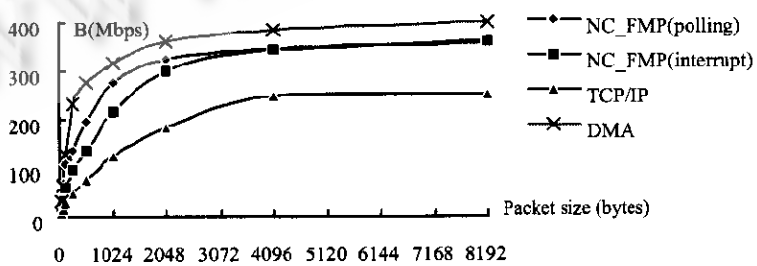
3.1.1 Latency

Note that "*p*" means polling mode and"*i*" means interrupt mode.

From the latency graph we can see that the latency of FMP is smaller than TCP/IP by one order of magnitude. That is because the reduced communication protocol eliminates most of the overhead in legacy protocol stack, which to a large extent determines the latency. We can also notice that the latency of polling mode is only one-third of that of interrupt mode. This testifies that context switch plays an important role in communication overhead.

3.1.2 Bandwidth

The bandwidth graph shows that bandwidth in FMP is approaching the rate of DMA. This explains why the bandwidth of FMP is still much lower than the 1.28Gbps rate of the Myrinet. In transmission of long messages, DMA becomes the bottleneck. And if we adopt other bus architectures, for instance, if we use PCI bus instead of SBUS, we can expect to achieve much better results.



3.2 Benchmark test of MPI based on FMP

Table 1 NAS benchmark test result

Prog	MPI/FMP	MPI/P4	Improvement (%)
bt. A. 9	525.84	567.85	7
bt. A. 16	273.70	329.75	17
sp. A. 9	291.92	387.61	25
sp. A. 16	151.90	296.26	49
ft. A. 8	31.30	38.39	18
ft. A. 16	17.70	23.26	24

To illustrate the overall performance of FMP protocol, we use NAS Benchmark to compare the execution time of MPI/FMP and that of MPI/P4——the popular TCP/IP based implementation of MPI used in workstation clusters. Table 1 lists a part of the results. The program name of NAS Benchmark is 'prog. CLASS. nproc', where prog represents the kind of problems to be solved by the parallel program, CLASS indicates the scale of the problem and nproc is the number of processors used. The execution time is in seconds. All results show improvements of MPI/FMP. Limited by the paper space, we only list the execution time of three kinds of programs with at least 8 processors.

4 Conclusions

In this article, we present FMP——a communication protocol specifically designed for workstation clusters, which eliminates the unnecessary software overhead of TCP/IP. FMP is a complete protocol in that functions at all levels are implemented, from the physical layer (Myrinet control program, network driver) to application layer (MPI and PVM). Our goal is to extensively exploit the performance potential of advanced communication equipment. The philosophy of FMP is to make the protocol as simple as possible, assuming the reliability of the underlying hardware. From our performance test, this approach is feasible and successful in improving both the point-to-point features and the overall execution speed of parallel applications.

References:

[1] Dong, C. L., Zheng, W. M., *et al*. A scalable parallel workstation cluster system. In: IEEE Computer Society ed. Proceedings of the APDC'97. Shanghai: IEEE Computer Society Press, 1997. 307~313.

[2] Kay, J., Pasquale, J. The importance of non-data-touching overheads in TCP/IP. In: ACM SIGCOMM Computer Communication Review, 1993,23(4):259~260.

[3] Boden, N. J., Cohen, D., Felderman, R. E., *et al*. Myrinet: a gigabit-per-second local area network. IEEE Micro, 1995, 15(1):29~36.

[4] Dubnicki, C., Bilas, A., Li, K., *et al*. Design and implementation of virtual memory-mapped communication on Myrinet. In: IEEE Computer Society ed. Proceedings of the 1997 International Parallel Processing Symposium. Shanghai: IEEE Computer Society Press, 1997. 388~396.

适用于工作站机群系统高性能计算的快速消息传递协议

欧新明, 申 俊, 郑纬民

(清华大学 计算机科学与技术系,北京 100084)

摘要: 消息传递系统对于工作站机群系统的计算性能是至关重要的.随着网络设备性能的飞速提高,传统的TCP/IP协议中不必要的开销已经成为通信速度的瓶颈.为了能够充分利用高速硬件的通信潜能,介绍了一种专门为Myrinet设计的新的通信协议——FMP.该协议包含网络部分和本地部分,是一个精简的通信协议.物理层假定是不会出错的,因此该协议十分简单而高效.与TCP/IP相比,该协议的性能有了很大的提高.介绍了该协议中的关键技术,并展示了部分结果.

关键词: 并行计算;工作站机群;消息传递接口;通信协议;Myrinet

中图法分类号: TP393 **文献标识码**: A