Portability in Implementing Distributed Shared Memory System on the Workstation Cluster Environment

Nanri, Takeshi
Computer Center, Kyushu University

Sato, Hiroyuki
Department of Computer Science and Communication Engineering, Kyushu University

Shimasaki, Masaaki
Kyoto University

https://doi.org/10.15017/1522389

出版情報：九州大学大学院システム情報科学紀要. 2 (2), pp.185-190, 1997-09-26. 九州大学大学院システム情報科学研究科
バージョン：
権利関係：

九州大学大学院
システム情報科学研究科報告
第2巻 第2号 平成9年9月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.2, No.2, September 1997

Portability in Implementing Distributed Shared Memory System on the Workstation Cluster Environment

Takeshi NANRI*, Hiroyuki SATO**, Masaaki SHIMASAKI***

(Received June 23, 1997)

Abstract: Cluster of workstations or personal computers connected with high speed network has become one of major architectures of distributed memory parallel computers. However, software on the cluster environment is still not improved in performance. In particular, there has not been proposed a standard distributed shared memory mechanism on the cluster environment, The distributed shared memory can be a solution of programming style on distributed memory parallel system including clusters because we know from experiences that shared memory model makes programming easy. It must be implemented with care for the performance problem. However, there is no hardware support for shared memory system. Another, but serious problem is the portability. In this paper, we discuss the design and implementation of portable distributed shared memory system. Our shared memory system is based on PVM in consideration of portability. Our contributions in this paper is the design and implementation of portable shared memory system on the cluster environment using faithful implementation of active messages fully in software, together with an enhancement of PVM to support active messages.

Keywords: Portability, Cluster environment, Distributed shared memory, Active message

1. Introduction

Cluster of workstations or personal computers connected with high speed network has become one of major architectures of distributed memory parallel computers. Its advantage over parallel computers provided with special hardware is the cost effectiveness. On the other hand, on performance, there has not been put stress. However, as the performance of general purpose network such as Ethernet or FDDI has been improved rapidly, the parallel performance of clusters has also been improved. Today, a cluster can be considered as an alternative of distributed memory parallel computers with special hardware. From this viewpoint, there have been carried out projects such as NOW of UCB[8], RWC of RWCP[9] and SWCP of U. Penn.

On the side of software on the cluster environment, programs have been written almost in the message passing style in which programming is not easy. As the programming style of the cluster environment, however, the distributed shared memory can be a solution on distributed memory parallel system because we know from experiences that shared memory model makes programming easy. However, it must be implemented with care because in its implementation on the cluster environment, we cannot assume any hardware support, but the high performance is strongly required to remote memory access. Another, but serious problem is the portability. Distributed shared memory must be implemented on a standard language or standard message passing libraries.

In this paper, we examine the design issues of portable distributed shared memory system. In its implementation, we must consider the simulation of primitive level functions related with memory access such as DMA and hardware interrupt. We discuss our implementation from the above viewpoint together with that of portability. Specifically, we show our design and implementation of portable shared memory system on the cluster environment using faithful implementation of active messages fully in software, together with an enhancement of PVM to support active messages.

The rest of this paper is organized as: Section 2 surveys the related work. Section 3 discusses our design of portable shared memory system. Section 4 outlines our design and implementation of distributed shared memory system using our faithful implementation of active messages fully in software, together with an enhancement of PVM to support active messages. Section 5 shows the performance of our implementation. Section 6 gives a brief summary.

* Computer Center
** Department of Computer Science and Communication Engineering
*** Kyoto University

2. Related Work

Today, we can say that the cluster environment is a major class of distributed memory parallel architectures. Projects on the cluster environment include NOW of UCB[8], RWC of RWCP[9], and SWCP of U. Penn, together with the well-tuned message passing libraries such as active message of NOW and the PM library of RWCP. All of these projects aim at proposing alternatives of "real" parallel machines by building parallel system by connecting high performance workstations with high speed network, which is a significant difference from conventional clusters. Our project shares the problem with those projects. However, we also put stress on portability, and we do not use high performance network such as Myrinet[2], but restricting the network interface within 10BaseT or 100BaseT and connecting switches.

TreadMark[1] is a project of distributed shared memory system. Memory allocation and access are implemented as library calls. The access method to the global memory space is separated from that to the local memory space. In TreadMark, memory consistency is also given a consideration, and lazy release consistency is implemented. However, it is pointed out that in the implementation, even the access to the global memory resident on the local processor raises signals.

Porting of Split-C[4] is reduced to the implementation of active messages on the target environment. Culler reports an implementation of active message on Myrinet[10]. In the UCB implementation of Split-C on PVM, however, the active message is not faithfully implemented. Our previous work[5] is an implementation of Split-C on PVM. However, the implementation of active messages is not safe.

HPF is another major distributed shared memory programming language. Sato et.al.[7] report the implementation of HPF using Split-C as its intermediate language.

3. Design of Portable Distributed Shared Memory System

Our problem is to design and implement portable distributed shared memory system fully in software.

3.1 Functions to be Implemented

In implementing the distributed shared memory system on the cluster environment, the fact that we cannot expect any hardware support nor any operating system support is a severe handicap. As for the hardware support, the function which handles the request of memory access and the transfer of data is strongly required. In other words, we need a kind of DMA method to a remote processor for the performance improvement. As for the operating system support, the function which resolves the address of the data posessed by a remote processor is indispensable.

In real distributed memory parallel machines, those functions are fully provided, or at least they are given a consideration. However, in the cluster environment, there is no support of such functions. Therefore, we must implement them fully in software.

Major troubles in implementing them are as follows:

1. we do not have any DMA to remote memory. This means that we must implement the memory access method using software interrupt and user-level event queue.
   In early architectures, the network interface raises an interrupt to CPU when network packets are transferred to memory. However, it has become clear that the cost of such interrupts can be the obstacle in the performance improvement. Architectures of these days implement DMA-like interface between network interface and memory. Because the cluster environment does not have DMA between network and memory, the cost is severely high.
   Though we cannot fully solve this problem, we can lower communication cost by eliminating unnecessary transfer of data. Active message can be a candidate for a solution.
2. we cannot make any assumption on the static address resolution because data on the distributed memory machines is usually transferred using ports. This means that we must implement a certain dynamic mechanism for the address resolution.
   We solve this question by imposing the condition: we consider the SPMD programming style on the homogeneous cluster environment, and we fork the same binary file on each node processor, by which the memory layout is known at compile time, and thus the address resolution problem becomes easy.

3.2 Portability

The next major problem is portability. Today, portability is one of major concerns in programming. To keep portability, we write programs in

- a standard programming language which is widely used, and
- a standard message passing library functions.

We choose Split-C as the target of the programming language which supports distributed shared memory. We implement its distributed shared memory system on PVM. The protocol stack of our implementation is, therefore, figured in **Figure** 1.

In our implementation, active messages on PVM is proposed as the common memory access methods. On our active messages for the cluster environment, we can build a variety of shared memory programming languages other than Split-C. In the subsequent section, we discuss the enhancement of PVM to support active messages. In fact, we need more powerful functions of signals to simulate the real hardware interrupt functions by PVM.

MPI vs. PVM

PVM and MPI are two major portable message passing libraries. In choosing one of them, we consider their expressive power in implementing the distributed shared memory system.

As discussed in 3.1, we must design our distributed shared memory system by virtually simulating hardware functions for parallel machines. From this viewpoint, PVM has advantage over MPI in that PVM has the inter-processor signal handling functions by which we simulate the hardware interrupt of each processing element of parallel machines, and implement a parallel *virtual* machine.

In MPI-2[11], the demand-driven data transfer functions are implemented as `MPI_WIN_PUT`, `MPI_WIN_GET`, and `MPI_WIN_ACCUMULATE`, though those functions are still clumsy. Windows which are defined to be the pair of memory block and its owner process for the remote memory access must be declared first. The remote data is accessed via the offset from the base address of the memory block of the window. Moreover, access phase and open phase are defined. RMA(remote memory access) can be allowed only when the origin is in the access phase, and the target is in the open phase. These phases must be controlled by a programer.

The principles of RMA of MPI-2 are different from that of shared memory in which remote memory is accessed in the same way as the local memory without phase control nor domain restriction. Active messages do not have the above phase problem because in any phase, they interrupt the target process with the handler specified by the message.

4. Implementation

Our design problems of distributed shared memory are summarized as the address resolution of the remote data and the safe and faithful implementation of active messages.

We design and implement a distributed shared memory system on the cluster environment by porting active messages[3] of Splt-C[4] on PVM. What we must implement is the faithful simulation of the memory access sequence to a remote node in real parallel machines, which includes the interface of network interface with memory unit and CPU unit and the address resolution of the memory to access.

In this paper, we divide the porting problem into two stages: implementation of active messages on PVM and address resolution mechanism on the cluster environment.

4.1 Problems in Unix

Design of the shared address space of Split-C depends on the implementation of functions of 3.1 which are supported by hardware in real parallel machines. As our implementation of these functions, we use software signals as the substitute of hardware interrupt, and we make the SPMD assumption: every object on each processor has the same image.

However, these decisions can cause other kind of problems. First, we must simulate the hardware interrupt related with memory access using software signals of Unix. In Unix, software signals are not guranteed to be raised. At least, sending a signal while handling a signal and having a waiting signal results in the loss of the signal last sent, and there is no method of knowing whether a signal is lost or not as in **Figure** 2. In PVM, `pvm_sendsig`, signal handling function is provided. This function has the same semantics as the software signals. Because simulating hardware interrupts needs safe 'signal' handling, PVM lacks the expressive power in this sense.

Next, SPMD assumption makes the address resolution easy, but the real address is determined at the fork of a process, which means that we cannot resolve addresses fully at compile time, and therefore that we need some runtime mechanism of address resolution.

4.2 Active Message

In our design, we aim at faithfully implementing active messages fully in software.

Split-C	↔	Language Layer	
Active Message			
PVM	↔	Message Passing Layer	Vender-Specific
UDP, (Domain Socket)	↔	Transport Layer	Hardware-Support
IP			
Cluster			"Real" Parallel Machine

Fig.1 Protocol Stack

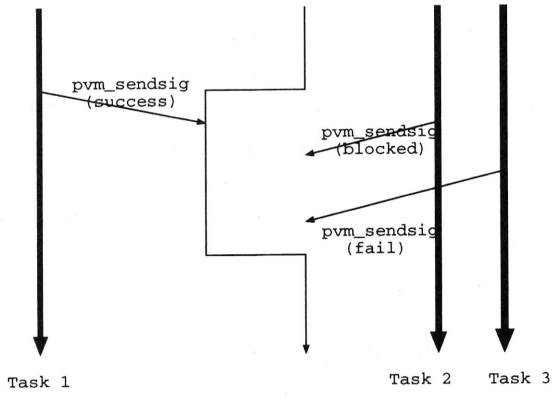

Fig.2 Case that a Unix Signal may be Lost.

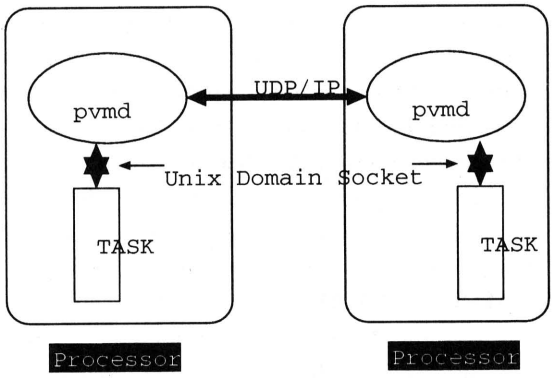

Fig.3 Communication in PVM

4.2.1 Active Messages on PVM

von Eiken et.al. proposed active messages[3]. Communication using active messages is characterized as the reduced access phases, and as the bufferless handler raising. In the conventional data transfer, three-phase protocol(request, ack, actual data transfer) is used. Instead, an active message sends a request together with a(n address of) handler for message handling, and the phase of *ack* can be eliminated. The message is handled immediately, and therefore message buffer is not needed. In this sense, an active message is suitable for handling small size messages because of this reduced access phase.

In this subsection, we examine PVM from the above viewpoints.

We use PVM signal handling functions in implementing handler invocation of active messages. In PVM, this is implemented as the daemon-daemon communication and daemon-task communication as in **Figure** 3.

First, we discuss the phase reduction in active messages. The principle in phase reduction in active messages is that by using active message mechanism, we can avoid the unnecessary acknoledgment between communication request and the data send/receive action. In our implementation, our version of `pvm_sig` is sent with the signal number and the data to be sent/received on UDP. Unlike TCP, the implicit acknoledgment is not sent in UDP. Therefore, if we ignore the Unix domain socket protocol used in communication between intra-processor tasks, we can conclude that we can avoid three-phase protocols in our implementation.

Second, we discuss the handler invocation in active messages. The handler invocation is implemented using PVM signal invocation. At this point, the reliability of handler invocation must be discussed, because PVM signal invocation depends on the Unix signal invocation. We solve this problem by enhancing the daemon around signals.

4.2.2 Enhancement of PVM

We add two entries of libraly functions of PVM to send signal safely as **Table** 1.

For the implementation of active messages, we need an enhancement of PVM on the safe signal handling, and on the argument passing to the signal handling. Here, we discuss the former enhancement which needs the essential rewriting of PVM daemon.

In `pvmd`, we have made the pending queue of signal request for each PVM task. It is guaranteed that no two signals are not raised at the same time by a `pvmd`. Changes around `dm_sendsig`(the entry for signal handling in `pvmd`) are figured in **Figure** 4

Just before the invocation of `dm_sendsig`, signal handling functions in the daemon, a flag is tested whether a signal is raised or not. If a flag is down, `dm_sendsig` is invoked as usual. If a signal is raised, however, the request of `dm_sendsig` is put into the

Table 1 Signal Handling Functions of PVM(* our enhancement)

Function	Description
`pvm_sendsig(tid, signum)`	Raise Signal(conventional semantics)
`pvm_safesigsend(tid, signum, mid)*`	Raise signal safely, and send data to the handler. Signal is guaranteed to be raised.
`pvm_safesigsendrecv(tid, signum, mid, bufid)*`	Raise signal safely, and send data to the handler, and receive the data from the handler. Signal is guaranteed to be raised.

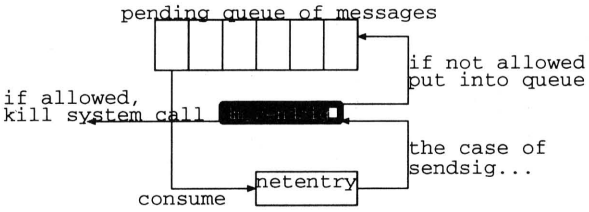

Fig.4 Changes around signal handling routines

message queue, and waits in the queue for the completion of the signal handler.

4.3 Global Address Space

On distributed memory machines, if a program is written in SPMD style, each processing element is assumed to have the same memory image. This essentially simplifies the implementation of virtual shared memory system, because data in a remote processor is placed at the identical address corresponding to that in a local processor. The loader and the operating system of the distributed memory machines are responsible for keeping this correspondence.

In the cluster environment, however, there is no guarantee that each node-executable file have the same address space. As a solution, we have separated the address of the global memory address in each processing element into a base address and its offset. The base address is obtained from the initialization routine by `sbrk(0)`. Global memory is accessed uniformly via the offset which is guaranteed to be the same in a given executable image in SPMD style. To obtain a piece of data from a remote processor, the node sends a request for the data whose address is given as the offset only. When the node has received the request, it returns the data of the address which is the base address plus the offset of the request.

4.4 Memory Access Sequence

With thus enhanced PVM and the address resolution mechanism, we can implement the access to

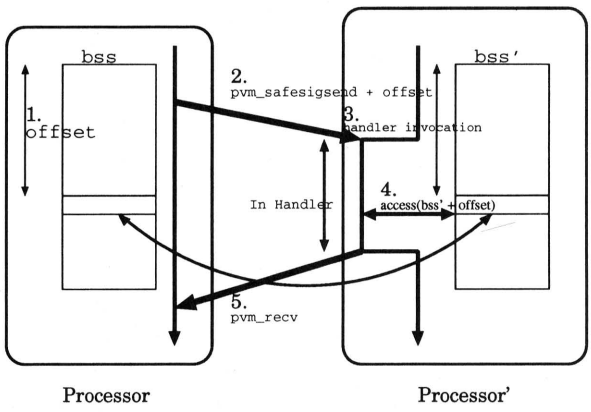

Fig.5 Sequence of Remote Memory Access

remote memory as follows:

1. issue a request of memory access with its address offset calculated as (corresponding address of the local processor) - (bss) to the pvm layer,
2. send the request to the task of the remote processor using `pvm_safesigsendrecv`,
3. receive the request, and invoke the signal handler,
4. calculate the address as (the offset sent in the request) + (bss), and access the data of the address in the signal handler,
5. return the value of the data.

Figure 5 illustrates the sequence. As the consequence, we can conclude that we faithfully implement active messages, and that we implement the address resolution mechanism with SPMD assumption.

5. Experimental Result

We have experimentally implemented the distributed shared memory on the cluster environment of **Table** 2 fully in software. **Table** 2 shows the latency and throughput of our distributed shared memory system. We find that the bottleneck of the performance is on the latency, rather than the throughput.

It is to be noted that the overhead of signal in-

Table 2 Latency and Throughput of the Remote Memory Access

Environment:

Processor	Pentium 166MHz
Network	10BaseT with XYLAN Switching Hub
OS	BSDI BSD/OS 2.1

Performance:

(get)

# bytes	latency	thruput
8	3.34	—
1024	5.94	0.172
8192	18.79	0.436
65536	114.54	0.572
644640	1050.93	0.613
	(msec.)	(MB/sec.)

(put)

# bytes	latency	thruput
8	3.26	—
1024	5.58	0.183
8192	17.24	0.475
65536	114.45	0.573
644640	1529.97	0.428
	(msec.)	(MB/sec.)

vocation is not yet a problem of the latency in our system. Let us consider the transfer of 8 bytes. In BSDI BSD/OS, the overhead of signal invocation is measured as 29 μsec. If we estimate the ideal communication cost as the throughput performance, it is approximately 16 μmsec(8 / ((0.613 + 0.428)/2)). Therefore, we can conclude that most of the latency is the overhead incurred by PVM and UDP/IP handling. To improve the latency, we must first improve the performance of PVM and UDP/IP.

There is another approach by which we can improve the performance. We can restructure the communication pattern by overlapping the communication and computation, by collecting the small size communication(by using bulk data transfer)[7], and by utilizing data locality (by using software cache)[6].

6. Concluding Remarks

In this paper, we discussed the design problems of distributed shared memory system on the cluster environment. Faithful implementation of active messages was discussed, and an enhancement of PVM to support active messages was also discussed. Finally, the performance of our implementation was discussed and some approaches for the performance improvement were investigated.

The authors have been partially supported by The Japan Society for The Promotion of Science Research for the Future Program (JSPS-RFTF96P00505: Software for Distributed and Parallel Supercomputing) and by The Ministry of Education, Science and Culture(No. 08458071).

References

1) Amza, C., Cox, A.L., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W., Zwaenepoel, W.: "TreadMarks: Shared memory computing on networks of workstations," IEEE Computer, 1996, pp. 18–28.
2) Boden, N.J., Cohe, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Wen-King Su: "Myrinet – A Gigabit-per-Second Local-Area Network," IEEE Micro, Vol. 15, No. 1, 1995, pp. 29–36.
3) von Eicken, T., Culler, E., Goldstein, S., Schauser K.: "Active Messages: a Mechanism for Integrated Communication and Computation," Proc. 1992 Int. Sympo. Computer Architecture, 1992, pp. 256–266.
4) Krishnamurthy, A., Culler, E., Dusseau, A., Goldstein, S., Lumetta, S., von Eicken, T., Yelick, K.: "Parallel Programming in Split-C," Proc. Supercomputing'93, 1993, pp. 262–273.
5) Nanri, T., Sato, H., Shimasaki, M.: "Implementing a Portable SPMD Shared Memory Model Parallel Language in a Distributed Computing Environment," Proc. Int. Symp. Parallel and Distributed SuperComputing, 1995, pp. 243–252.
6) Nanri, T., Sato, H., Shimasaki, M.: "Using Cache Optimizing Compilier for Managing Software Cache on Distributed Shared Memory System," Proc. HPC Asia 97, 1997, pp. 312–318.
7) Sato, H., Nanri, T., Shimasaki, M.: "Using Asynchronous and Bulk Communication to Construct an Optimizing Compiler for Distributed-Memory Machines with Consideration Given to Communication Costs," Proc. 1995 ACM ICS, 1995, pp. 185–189.
8) `http://now.cs.berkeley.edu`
9) `http://www.rwcp.or.jp`
10) `http://www.cs.berkeley.edu/AM/lam_release.html`
11) `http://ftp.cs.wisc.edu/pub/lederman/mpi2`