Ninf and PM : Communication Libraries for Global Computing and High-performance Cluster Computing

Mitsuhisa Sato [a], Hiroshi Tezuka [a], Atsushi Hori [a], Yutaka Ishikawa [a], Satoshi Sekiguchi [b], Hidemoto Nakada [b], Satoshi Matsuoka [c] and Umpei Nagashima [d]

[a] *Real World Computing Partnership,Mitsui-Blg.16F,Tsukuba, Ibaraki 305, Japan*

[b] *Electrotechnical Laboratory, 1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan*

[c] *Tokyo Institute of Technology, Ookayama 2-12-2, Meguro-ku, Tokyo 152, Japan*

[d] *Ochanomizu University, Otsuka 2-1-1, Bukyo-ku, Tokyo 112, Japan*

Abstract

This paper presents two advanced communication libraries, Ninf and PM. Ninf is an ongoing global network-wide computing infrastructure project which allows users to access computational resources including hardware, software and scientific data distributed across a wide area network. Computational resources are shared as Ninf remote libraries executable at a remote Ninf server. Users can build an application by calling the libraries with the Ninf Remote Procedure Call. In order to facilitate location transparency and network-wide parallelism, Ninf metaserver maintains global resource information regarding computational server and databases, allocating and scheduling coarse-grained computation for global load balancing. PM is a high performance communication library for workstation clusters connected with Myrinet gigabit LAN card, which has a dedicated processor and on-board memory to handle a communication protocol. In order to obtain high performance communication and support a multi-user environment, we co-designed PM, an operating system realized by a daemon process, and the run-time routine for a programming language. Several unique features, e.g., network context switching and *modified ACK/NACK* flow control algorithm have been developed for PM. PM for the Suns has a speed of 20 μseconds round trip for a user-level 8 bytes message and 38.6 Mbytes/second bandwidth for an 8 Kbytes message.

Keywords: Global Computing, Remote procedure call, Cluster computing, Gigabit LAN, User-level communication protocol, High-performance computing.

Preprint submitted to Elsevier Science *11 November 1997*

1 Introduction

In recent years, two important trends, global computing and cluster computing, have been gathering many interests in high performance network computing.

Remarkable growth of computer network technology has spurned a variety of information services accessible through the Internet. While most existing global network services such as e-mail, file archives, and the WWW, are limited to merely sharing data resources, the global network could be far better utilized, embodying the potential to provide a computational environment to share computational resources including CPUs and disk storage. The coming of gigabit information superhighway will further enhance the vision of world-wide global computing resources, being able to tap into powers of enormous numbers of computers with idle computation cycles. For the global network computing, libraries and systems such as Legion [4], NetSolve[3], Remote Computation System (RCS) [2], have been proposed.

On the other hand, it has become possible to cluster high-performance commodity workstations and PCs to build cost-effective parallel computing platforms using high-speed interconnect technologies such as ATM, Fibre Channel and Myrinet[8]. These interconnects, "killer switch", have performance and scalability of the MPP interconnect and the flexibility of a local area network. It is viable to build parallel systems that are fast, inexpensive, and highly available, by simply connecting these switches together with the last high performance commodity components.

As an infrastructure for world-wide global computing in scientific computation, we are currently pursuing the *Ninf (Network based Information Library for high performance computing)* project [13,11] [1]. The goal of Ninf is to provide a platform for global scientific computing with computational resources distributed in a world-wide global network. The basic Ninf system supports client-server based computing. The computational resources are available as remote libraries at a remote computation host which can be called through the global network from a programmer's client program written in existing languages such as Fortran, C, or C++ using *Ninf remote procedure call (RPC)*. The programmer can build a global computing systems by using the Ninf remote libraries as its components, without being aware of complexities and hassles of network programming.

[1] The URL of Ninf project is `http://ninf.etl.go.jp/`. The nickname "Ninf" comes from a similar pronounce word "Nymph", which we expect it, living in a network world, to make the dream of quickly getting information resource come true.

PM[15] is a high performance communication library for workstation clusters connected with Myrinet gigabit LAN card. We have co-designed the PM library for a run-time system for a programming language called MPC++[7,6], an operating system called SCore to support multi-user parallel processing environments. SCore is implemented as a daemon process called SCore-D on top of a Unix operating system and manages a set of user processes, called a *parallel process*, on workstations. SCore-D employs gang scheduling to change the context of the parallel process, including its network state. This allows multiple users to access the workstation cluster in a time and space sharing (TSSS) fashion[5]. The PM also provides a reliable FIFO order message passing facility which the MPC++ run-time routine assumes.

The Myrinet network interface has a dedicated processor, which allows users to design own communication protocol. To achieve a low latency and high bandwidth communication, user memory mapped network drivers are widely employed, such as Active Messages (AM)[1], Fast Messages (FM)[12], and U-Net[16]. In these drivers, the user process directly accesses the network hardware to eliminate kernel traps and data copies. However, most communication facilities can only be used by a single process at a time because the process exclusively uses the network hardware resource. The PM supports a communication channel, which represents the network status and includes a message buffer on the on-board memory, to support multiple communication paths required for context switching of the SCore gang scheduling facility. *Modified ACK/NACK* flow control algorithm and *Immediate Sending* technique are developed to provide high throughput and low latency communication and to guarantee the FIFO order message delivery.

In the next section, we describes the overview of Ninf, its components, and the underlying technologies that support such global computing. Section 3 presents the design and implementation of PM library. Section 4 concludes the paper.

2 Ninf: A Network based Information Library for Global World-Wide Computing Infrastructure

2.1 Ninf Overview

The basic Ninf system employs a client-server model. The server machine and a client may be connected via a local area network or over an Internet. Machines may be heterogeneous: data in communication is translated into the common network data format.

A Ninf server process runs on the Ninf computation server host. The Ninf remote libraries are implemented as executable programs which contain network *stub* routine as its main routine, and managed by the server process. We call such executable programs *Ninf executables (programs).* When the library is called by a client program, the Ninf server searches the Ninf executables associated with its name, and executes the found executable, setting up an appropriate communication with the client. The stub routine handles the communication to the Ninf server and its client, including argument marshaling. The underlying executable can be written in any existing scientific languages such as Fortran, C, etc., as long as it can be executed in the host.

The benefits of Ninf are as follows:

- A client can execute the most time-consuming part of his program in multiple, remote high-performance computers, such as vector supercomputers and MPPs, without any requirement for special hardware or operating systems. If such supercomputers are reachable via a high speed network, the application will naturally run considerably faster. It also provides uniform access to a variety of supercomputers.
- The Ninf programming interface is designed to be extremely easy-to-use and familiar-looking for programmers of existing languages such as FORTRAN, C and C++. The user can call the remote libraries without any knowledge of the network programming, and easily convert his existing applications that already use popular numerical libraries such as LAPACK.
- The Ninf RPC can also be asynchronous and automatic: for parallel applications, a group of *Ninf metaservers* maintains the information of Ninf servers in the network in a distributed manner, and automatically allocates remote library calls dynamically on appropriate servers for load balancing. To allocate multiple calls to achieve network-wide parallelism, Ninf provides a transaction system, where the data-dependencies among the Ninf calls are automatically detected and scheduled by the metaserver. The Ninf metaserver could be regarded as a network agent which locates an appropriate server depending on client request and status of network resources.
- The Ninf network database server provides query on accurate constant database needed in scientific computation, such as important constant numbers in physics and chemistry. By doing so, the user is freed from the burdens and mistakes of inputting accurate constant data from printed charts. Furthermore, we plan to develop a framework where data values of reports on experiments by researchers could be referenced directly over a network in the midst of user's computation.

While Ninf can serve to speed-up the applications by exploiting the power of remote supercomputer servers and network parallelism, it is a convenient tool for obtaining the answer of a particular numerical function and values quickly. We have designed a WWW interface, called *NinfCalc* as a front-end so that

Ninf libraries registered in computation servers could be cataloged, browsed, and tried out through Web browsers.

From the user's perspective, Ninf offers yet another way to share resources over a global network. There are already various network infrastructures and tools already in use, and one might ask would another infrastructure be needed; for example, one could claim the extreme that, one could use existing file transfer services such as ftp to remotely obtain numerical libraries, compile on his local machine, and execute the program there. Aside from the issue of having a supercomputer locally in hand, there are many other advantages to Ninf in this case — security and proprietary issues naturally are obvious topics; there are other practical issues of expending the efforts of fetching, compiling, and maintaining code in heterogeneous computing environments. Ninf allows reduction of maintenance costs by concentrating the efforts of high-quality, well-maintained libraries on compute servers and not on each machine. Thus, even for slower networks where there are no speed advantages to be gained, Ninf is still beneficial in this regard.

2.2 The Programming Interface

`Ninf_call()` is the sole client interface to the Ninf compute and database servers. In order to illustrate the programming interface with an example, let us consider a simple matrix multiply routine in C programs with the following interface:

```
double A[N][N],B[N][N],C[N][N];       /* declaration */
....
dmmul(N,A,B,C);  /* calls matrix multiply, C = A * B */
```

When the `dmmul` routine is available on a Ninf server, the client program call the remote library using `Ninf_call`, in the following manner:

```
Ninf_call("dmmul",N,A,B,C); /* call remote Ninf library */
```

Here, `dmmul` is the name of library registered as a Ninf executable on a server, and `A`,`B`,`C`,`N` are the same arguments. As we see here, the client user only needs to specify the name of the function as if he were making a local function call; `Ninf_call()` automatically determines the function arity and the type of each argument, appropriately marshals the arguments, makes the remote call to the server, obtains the results, places the results in the appropriate argument, and returns to the client. In this way, the Ninf RPC is designed to give the users an illusion that arguments are shared between the client and the server. Note that the physical location of the Ninf server is specified in an environment variable, or a setup file.

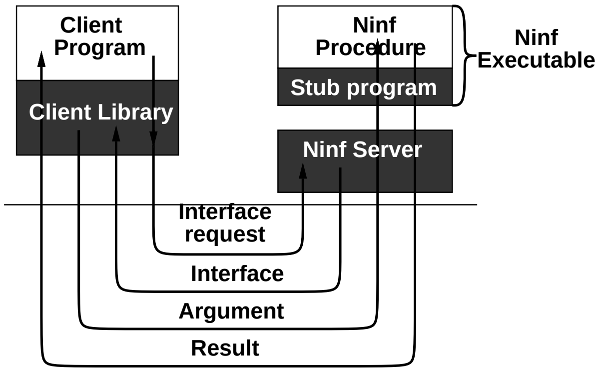


Fig. 1. Ninf RPC

To realize such simplicity in the client programming interface, we designed Ninf RPC so that client function call obtains all the interface information regarding the called library function at runtime from the server. As shown in Figure 1, the client function call requests the interface information of the calling function to the Ninf server, which in turn returns the registered Ninf executable interface information to the client. Using these informations, Ninf RPC automatically performs argument marshaling, and generates the sequence of sending and receiving data from/to the Ninf server. The client library then interprets and marshals the arguments on the stack according to the supplied information. For variable-sized array arguments, the IDL must specify an expression that includes the input scalar arguments whereby the size of the arrays can be computed. In the above example, the client function call sends the input arrays, `A` and `B`, whose size is computed by the parameter `N`. The Ninf server invokes the Ninf executable of `dmmul` library, and forwards the input data to it.

This design is in contrast to traditional RPCs, where stub generation is done on the client side at compile time. As a result of dynamic interface acquisition, Ninf RPC does not require such compile-time activities at all, relieving the users from any code maintenance. Nakada[10] describes the structure of the interface information and the protocol in detail. Although this will cost an extra network round trip time, we judged that typical scientific applications are both compute and data intensive such that the overhead should be small enough.

The Ninf RPC may also be invoked asynchronously to exploit network-wide parallelism. It is possible to issue the request to a Ninf server, continue with the other computation, and poll for the request later. Multiple RPC requests to different servers are also possible. For this reason, the asynchronous Ninf RPC is an important feature for parallel programming.

Since the Ninf client programming interface is designed to be as language independent as possible, the Ninf client may written in a variety of programming languages. It is usually easy to design a client interface to Ninf, so long as the language supports standard foreign function interface to C programs. We have already designed and implemented the client `Ninf_call` functions for C, FORTRAN, Java, and Lisp. The client and the remote library can be written in a totally different language.

2.3 Ninf IDL (Interface Description Language)

A *library provider* , who provides the numerical library and computational resource to the network at large, prepares the Ninf executables by the following steps: (1) write the necessary interface description of each library function in *Ninf IDL (Interface Description Language)* , (2) run the Ninf IDL compiler, which emits the necessary header files and stub code, (3) compile the library with the compiler for the programming language the library is written in, and, (4) link with the Ninf RPC libraries, finally, (5) *register* them with the Ninf server running on his host. After these steps, anyone in the network can use the libraries by the Ninf RPC in a transparent. Some existing libraries, such as LAPACK, have already been 'Ninfied' in this manner.

For example, the interface description for the matrix multiply given in the previous section is:

```
Define dmmul(long mode_in int n, mode_in double A[n][n],
        mode_in double B[n][n], mode_out double C[n][n])
"... description ..."
Required "libxxx.o" /* library including this routine. */
Calls "C" dmmul(n,A,B,C);  /* Use C calling convention. */
```

where the *access specifiers* , `mode_in` and `mode_out`, specify whether the argument is read or written. To specify the size of each argument, the other `in_mode` arguments can be used to form a size expression. In this example, the value of `n` is referenced to calculate the size of the array arguments `A`, `B`, `C`. Since the Ninf system is designed for numerical applications, the supported data type in Ninf IDL is tailored for such a purpose; for example, the data types are limited to scalars and their multi-dimensional arrays. The interface description is compiled by the *Ninf interface generator* to generate a stub program for each library function described in its interface information. The interface generator also automatically outputs a makefile with which the Ninf executables can be created by linking the stub programs and library functions.

2.4 Ninf Metaserver

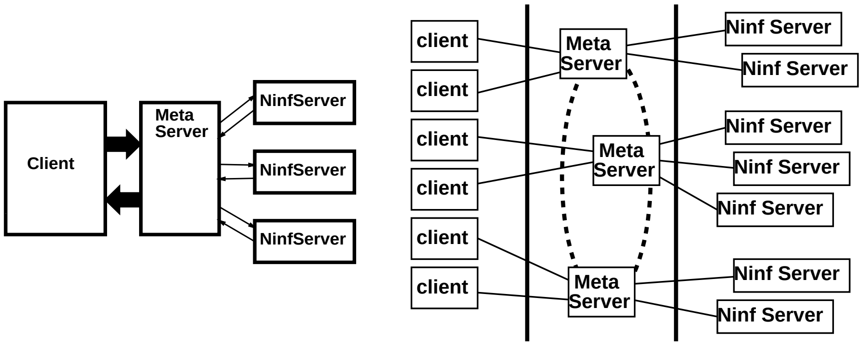


Fig. 2. Ninf Metasever and Metaserver Network

The *Ninf metaserver* [9] is an agent, a set of which gathers network information regarding the Ninf servers, and also helps the client to choose an appropriate Ninf server, either automatically or semi-automatically. Figure 2.4 shows a configuration with metaservers. The entire Ninf infrastructure consists of sets of servers and metaservers distributed in a network. Metaservers exchange information with each others periodically For example, if a new Ninf server is added into the metaserver, the location of the new server is propagated among the metaserver network. Also, when a metaserver does not have a particular information available (e.g., availability of a certain library), a query is propagated through the metaserver network and the result is gathered by the issuing metaserver, who in turn delegates the info or makes automatic decisions for the client.

The Ninf metaserver keeps track of the following information for each Ninf server: (1) the location (address and port number), (2) the list of functions registered in the server, (3) the distance to the server with respect to the bandwidth of communication, (4) computational ability of the machine (clock speed, Flop/s, etc...), (5) the status of the server including the load average.

The current metaserver is implemented in Java, in order to exploit its portability, ease of network programming, and multi-threading, and also given that speed is not an absolute factor.

The Ninf metaserver helps to alleviate this situation by gathering the meta-information of the status and the resource made available by the servers, and also by making Ninf services location transparent. For example, if many clients share the same server, the computation time is greatly influenced by the computation requested by other clients. Such dynamic information is constantly probed and delegated to other metaservers in a distributed way.

In order to support parallel programming with the meteservers, the client issues multiple requests asynchronously via a Ninf call `Ninf_call_async()`. Each outstanding request is dispatched to the best server by the metaserver. The client may continue its local computation, and wait for result to come back with `Ninf_call_wait` function. When the client executes `Ninf_call` asynchronously, he is able to issue a request to a metaserver, who in turn chooses the appropriate server and delegates the request. Also, instead of calling the server directly, the user may call the metaserver directly to find the "best" server to execute the requested library, resulting in good load balancing for multiple severs over the entire network.

Another programming interface for parallel programming , *Ninf parallel transaction*. The parallel transaction is a program region surrounded by `Ninf_check_in()` and `Ninf_check_out()`. When `Ninf_check_in()` is executed, the Ninf calls do not issue actual requests spontaneously, but the information of the calls are is stored and a dataflow graph is built to represent the data dependency of the remote computations. When the execution reaches `Ninf_check_out()`, the dataflow graph is sent to the Ninf metaserver. The metaserver schedules the computations in the dataflow graph to execute them efficiently with multiple Ninf servers, according to the information it has.

3 PM: An Operating System Coordinated High Performance Communication Library

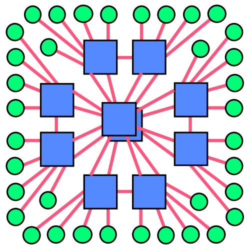
3.1 Workstation Cluster and Myrinet Gigabit LAN Card

PM is a high performance communication library for Myrinet, which is a gigabit LAN commercially produced by Myricom Inc. Figure 3 shows the configuration of our workstation cluster of 36 Sun Sparc Station 20s connected by Myrinet.

The Myrinet LAN interface card has a 32-bit microprocessor, an SRAM, a network interface, and a SBus or PCI DMA controller. The on-board processor executes programs stored in the SRAM to control the network interface and the DMA controller. Thus, we can develop new own protocols on Myrinet by changing the program in the card.

Because all messages transmitted to and received from the link must be stored in the SRAM, a separate data transfer between the SRAM and main memory is needed. This on-board SRAM is located in the SBus or PCI address space. By mapping this SRAM address area into the user address space, users can directly access data in the SRAM. This results in low latency message transfer using the polling technique.

Myrinet is distinguished from other LAN hardwares such as Ethernet and ATM by 1) its guarantee of message delivery using hardware flow control and



A circle and a square represent a node and a switch, respectively.

Fig. 3. Network topology

2) the preservation of message order transferred via the same path.

3.2 Co-design with Programming Languages and Operating Systems

Our MPC++ programming model is based on multiple threads on a distributed memory parallel machine in which thread execution continues unless it waits for synchronization or exits. The MPC++ version 2 template library[6], called the Multiple Threads Template Library (MTTL), supports local/remote thread invocation, synchronization structures, and global pointers. An interrupt-driven communication mechanism is not required in the MPC++ run-time system which deals with communication and user-level threads. The MPC++ run-time system assumes a reliable FIFO order message delivery mechanism. Since the MPC++ programming model is based on a distributed memory parallel machine, low latency and high throughput communication is required to communicate with remote threads.

The operating system on our workstation cluster is designed as a daemon process called SCore-D which communicates with other SCore-D daemon processes via Myrinet to implement the gang scheduling. That is, the user and the daemon processes access the Myrinet hardware resources simultaneously. To allow multiple processes (SCore-D and user processes) to use the communication facility simultaneously, such a connection will be realized by a channel. A message sent from a channel of the sender node is transferred to the same channel on the receiver node. Each channel will be occupied by a process. A channel must have its own message buffer so that congestion in one channel does not affect other channels.

Because it is not possible to create as many channels as needed with the limited resources of a LAN card's on-board memory, a channel must be used by multiple processes in a time-sharing fashion. To realize this, a mechanism of channel context switching is required. A channel context consists of all memory associated with the channel, including the message buffer and messages floating on the network.

We adopted a polling method to decrease the communication latency. In PM, the LAN card on-board processor transfers messages and the receiver thread of the user process polls the arriving messages. Polling immediately notifies the receiver thread about message arrivals and makes it possible to communicate with low latency. Protocol implementations which use system calls and interrupts and which are used in ordinary LANs such as Ethernet cannot satisfy the low latency requirement. Although AM on the sun workstation with the ATM (called SSAM)[14] use a special trap instruction to solve this problem, modifications to the operating system kernel are required.

Because the Myrinet network interface can only access the data in the on-board SRAM, transferring data between the host memory of each node involves four steps: 1) transferring the data by DMA from main memory to the on-board SRAM, 2) sending the data from the on-board SRAM to the network, 3) receiving the data from the network to the on-board SRAM, and 4) transferring the data from the on-board SRAM to main memory.

The sequential execution of DMA and message transmission cannot utilize the bandwidth of the DMA controller and the network interface. Although double buffering allows the DMA transfer of the next message to be executed while the current message is being sent in order to increase the throughput, it cannot decrease the latency of each message transfer.

We developed a technique called *Immediate Sending* that starts sending data from the SRAM to the network immediately after the DMA transfer from main memory to the SRAM begins, as in pipeline processing. *Immediate Sending* can both increase the throughput and decrease the latency.

On the receiver node, it is not possible to receive a message and issue a DMA transfer simultaneously, because a CRC error is detected after the whole message is received. PM uses the double buffering technique on the receiver node to increase throughput.

If the host program does not receive incoming messages, the message buffer in the on-board memory may overflow. We designed a new flow control based on the ACK/NACK protocol.

The flow control method for a workstation cluster should be scalable even with a large number of nodes. A window-based flow control algorithm is not scalable, because the receiver node must manage receive buffers dedicated to each sender node, and a larger number of nodes makes the effective buffer size smaller. Although the ordinary "ACK/NACK and Re-transmit" method or "Return to Sender [12]" which sends back messages which cannot be received do not have such a dividing buffer problem, they also do not preserve the message order.

A scalable flow control algorithm, which preserves the order of messages, has been designed. This algorithm is called *Modified ACK/NACK* in the sense that it uses the ACK/NACK and Re-transmit method, and the sender/receiver state is introduced to preserve message order. In this algorithm, message order is preserved because succeeding messages are not received until a message which could not be initially received is received normally. This algorithm requires the sender message buffer to store messages. A message buffer is freed only when the sender receives the ACK message for that message. This buffer does not occupy a large space. The Myrinet hardware, which guarantees the delivery of messages, allows us to use this simple flow control algorithm.

There are two advantages of the *Modified ACK/NACK* flow control. One is that the sender node knows whether or not a message has arrived at the receiver node. We use this feature to implement channel context switching. The context of the PM channel consists of data structures and buffers in the host main memory and on-board SRAM corresponding to the channel. To switch the channel context, these memory areas are saved to the host memory and the next channel context is restored. The context switching of channels must not cause duplicate messages, missing messages or the mixing of messages between contexts. To guarantee this, the context must be switched only when no out-going messages or in-coming acknowledges are present. Using our flow control, receiving all ACK/NACK's for messages already sent confirms that no messages remain on the network.

Another advantage is that all messages are always sent to the destination without being blocked whether or not the receiver message buffer overflows. If a user process sends a huge amount of messages over a channel and blocks the network, another channel never has a chance to send a message. The SCore-D operating system using a channel cannot proceed in such a case. The *Modified ACK/NACK* flow control precludes such a case from occurring.

The drawback of our flow control algorithm is in the possibility to increasing the network load when re-transmission occurs, because the sender continues to transmit messages until a NACK arrives at the sender node.

3.4 Performance

We have measured the basic performance of PM using two Sun Sparc station model 20/71 connected with the Myrinet switch and the Myrinet LANai 4.0 hardware. Figure 4 shows the round trip time of PM. In this evaluation, we varied the message size from 8 to 8192 bytes, and measured 1) With Immediate

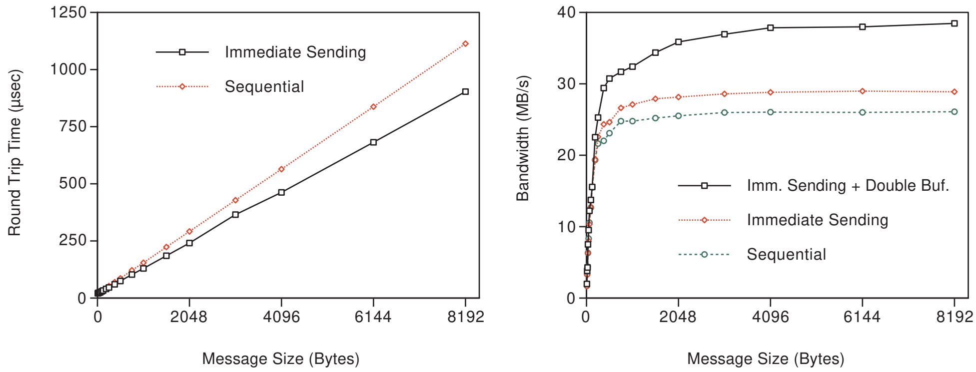


Fig. 4. Round Trip Time and Throughput

Sending, and 2) Without Immediate Sending. The PM round trip time for an 8 byte message is about 20 microseconds. Figure 4 demonstrates that the Immediate Sending technique decreases the latency for large messages.

Figure 4 shows the throughput of PM. In this evaluation, we varied the message size from 8 to 8192 bytes, and measured 1) With Immediate Sending and receive double buffering, 2) With Immediate Sending only, and 3) Without Immediate Sending and receive double buffering. The figure demonstrates that the Immediate Sending technique increases the throughput of large messages, and receive double buffering increases the throughput for all message sizes. The maximum throughput of PM is about 36.8 Mbytes per second with 8192 byte messages.

Condition	Save	Restore
No message	0.13	0.11
Send buffer full	1.88	1.40
Receive buffer full	3.39	1.95
Send and receive buffer full	5.15	3.22

Table 1
Context switch time (miliseconds)

Table 1 shows the time to save and restore a channel context. We measured the four cases: 1) no message in the buffer, 2) send buffer is full: 511 messages (12 Kbytes) are in the send buffer, 3) receive buffer is full: 2730 messages (32 Kbytes) are in the receive buffer, and 4) both send and receive buffers are full.

It takes longer to save and restore the channel context when messages are in the buffers. This is because PM has a relatively large message buffer, and the overhead to access the on-board SRAM via the SBus is large. Although the amount of data to be transferred when saving and restoring the context is the same, saving the context takes longer. This is because reading data from the SBus space takes longer than writing data to it.

4 Concluding Remarks

In this paper, we have presented two important communication libraries for advanced network computing: Ninf for global computing on wide-area network, and PM for high performance cluster computing on Myrinet connecting commodity workstations and PCs.

We have evaluated the performance of PM library using the CG (Conjugate Gradient) method benchmark of NAS Parallel Benchmark on our Sun workstation cluster shown in Figure 3. The result shows that our workstation cluster achieves comparable performance to the Cray T3D of the same number of processors. The PM allows users to exploit the parallel system of modern high performance processors and to share the system under the SCore operating system.

As the Ninf project is an on-going effort, we do expect various changes and extensions to occur as we gain more experience from real applications, and as the system becomes more mature through widespread use. Currently, we are running a suite of Ninf servers on Cray J-90, Sun UltraSparcs, and a 32-processor DEC Alpha workstation cluster at Electorotechnical Laboratory of Tsukuba, Japan as our platforms. On these servers, we have ported existing mathematical libraries by defined Ninf IDLs and registering with the Ninf server. The libraries include linear algebra packages such as LAPACK, Cray's LibSci, and other special mathematical libraries. A scientific constant database for physics and chemistry is also being designed and being built.

References

[1] http://now.cs.berkeley.edu/AM/lam_release.html.

[2] P. Arbenz, W. Gander, and M. Oettli. The Remote Computational System. *Lecture Note in Computer Science, High-Performance Computation and Network*, 1996.

[3] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. Technical report, University of Tennessee, 1996.

[4] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr. Legion: The Next Logical Step Toward a Nationalwide Virtual Computer. Technical report, University of Virginia, 1994.

[5] A. Hori, H. Tezuka, Y. Ishikawa, N. Soda, H. Konaka, and M. Maeda. Implementation of Gang-Scheduling on Workstation Cluster. In D. G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop on Job Scheduling Strategies for*

Parallel Processing, volume 1162 of *Lecture Notes in Computer Science*, pages 76–83. Springer-Verlag, April 1996.

[6] Y. Ishikawa. Multi Thread Template Library – MPC++ Version 2.0 Level 0 Document –. Technical Report TR–96012, RWC, September 1996. *Visit http://www.rwcp.or.jp/lab/mpslab/mpc++/mpc++.html.*

[7] Y. Ishikawa, A. Hori, H. Tezuka, M. Matsuda, H. Konaka, M. Maeda, T. Tomokiyo, and J. Nolte. MPC++. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming Using C++*, pages 429–464. MIT Press, 1996.

[8] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and Wen-King Su. "Myrinet – A Gigabit-per-Second Local-Area Network". *IEEE MICRO*, 15(1):29–36, February 1995.

[9] H. Nakada, Kusano, S. Matsuoka, M. Sato, and S. Sekiguchi. A Metaserver Architecture for Ninf: Networked Information Library for High Performance Computing. In *SIGNote of Information Prorcessing Society*, number 96-HPC-60, 1996. in Japanese.

[10] H. Nakada, M. Sato, and Sekiguchi. Ninf RPC Protocol. Technical Report TR 95-28, Electrotechnical Laboratory, 1995. in Japanese.

[11] M. Sato, H. Nakada S. Sekiguchi, , S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A Network based Information Library for Global World-Wide Computing Infrastructure. *Lecture Note in Computer Science, High-Performance Computing and Networking*, pages 491–502, 1997.

[12] Scott Pakin, Mario Lauria and Andrew Chein. "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet". In *Proceedings of Supercomputing '95, San Diego, California*, 1995.

[13] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. — Ninf — : Network based Information Library for Globally High Performance Computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA)*, Feb. 1996.

[14] T. von Eicken, V. Avula, A. Basu and V. Buch. "Low-Latency Communication over ATM Networks using Active Messages". In *Proceedings of Hot Interconnects II, 1994 Palo Alto*, August 1994.

[15] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An Operating System Coordinated High Performance Communication Library. *Lecture Note in Computer Science, High-Performance Computing and Networking*, pages 708–717, 1997.

[16] Thorston von Eicken, Anindya Basu, and Werner Vogels. U-Net: A User Level Network Interface for Parallel and Distributed Computing. In *Fifteenth ACM Sumposium on Operating Systems Principles*, pages 40–53, 1995.