MBCF: A Protected and Virtualized High-Speed User-Level Memory-Based Communication Facility

Takashi MATSUMOTO and Kei HIRAKI

Department of Information Science, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan.
e-mail: {tm, hiraki}@is.s.u-tokyo.ac.jp

Abstract

We introduce a novel high-speed user-level communication and synchronization scheme "Memory-Based Communication Facilities (MBCF)" for a general-purpose system with an off-the-shelf communication-hardware. This mechanism is protected and virtualized as completely as memory so that it can be used not only in parallel systems but also in distributed systems. The MBCF realizes the direct remote-memory-accesses in user-task-spaces and offers programmers and compilers a wide variety of functions and a large shared-memory space. In this paper we first explain outlines and features of the MBCF, and present the high-speed implementation techniques for the MBCF. Then this paper describes varieties of the MBCF functions and introduces two novel memory-based mechanism: the Memory-Based FIFO and the Memory-Based Signal. Next we show that the MBCF is more flexible and less expensive than the message-passing-style system-interfaces for communication or generalized active messages. Finally we show performance evaluations on the real MBCF implementation using 100BASE-TX Ethernet.

1 Introduction

For realizing a general-purpose computing environment on a large-scale multicomputers (MPP) and a Network of Workstations (NOW), performance of the system is often compromised by introduction of a multi-tasking system image, protection mechanisms and virtualization of system resources. Consequently, current parallel processing environment including NOWs and MPPs are mainly used as a back-end computer for numerical computations or as a distributed processing environment for almost independent single tasks. It is why various optimization technique in parallel processing and light-weight communication mechanisms require exclusive use of the system if the system is used as a parallel computer.

For solving this problem the key technology is the protected and virtualized high-speed user-level communication and synchronization. We proposed a novel high-speed user-level communication and synchronization scheme "**Memory-Based Communication Facilities (MBCF)**"[1, 2] for a general-purpose system with an off-the-shelf communication-hardware. The MBCF is a software-only solution for realizing protected and virtualized communication and synchronization, and it is developed for NOWs and distributed memory multiprocessors without hardware DSM mechanisms.

Conventional user-level communication interfaces (for examples, TCP/IP, UDP/IP and MPI) have message-passing-type ones, and their functions are limited in remote-write operations to specific message-buffer addresses in the kernel-space. To break out of this limitations, we adopt memory-based operations where arbitrary target addresses and wide variety of functions can be used. By adopting memory-based operations protections and virtualizations in communications and synchronizations can be replaced with those of memory accesses. This replacement makes high-speed implementations of the scheme feasible, since advanced architectural mechanisms of processors for memory-accesses can be exploited.

We assume that off-the-shelf Network Interface Cards (NICs) are equipped in the MBCF system. These cards have no functionalities for protection or security, transmit memory image of a packet to other nodes and receive packets from other nodes into a specified ring buffer in the system (kernel) space.

2 Memory-Based Communication Facilities (MBCF)

2.1 Outline of the MBCF

The MBCF emulates Memory-Based Processor (MBP[3, 4, 5, 6])'s functions using pure software routines which include user-level requesting codes (packet sending codes). In the MBP system, remote memory accesses are invoked by processors' memory operations. When a MBP detects processor's memory access whose target address belongs to a remote node, the MBP translates the access information into an inter-node communication style and makes a packet and transmits it to the target node. When the MBP in the target node receives the packet, it executes the remote access specified in the packet and returns a reply if necessary. On the other hand, in the MBCF system remote memory accesses are invoked by explicit system-calls for the MBCF functions. First a user-program prepares an MBCF packet in user-mode and executes the MBCF requesting system-call. Secondly the kernel-level routine of the MBCF-dedicated requesting system-call makes a inter-node communication-

style packet and transmits it using conventional NICs. Finally the MBCF-dedicated interrupt routine at the target node receives the packet and directly executes the remote access specified in the packet and returns a reply if necessary.

In comparison with the MBP systems, the MBCF systems suffer with some additional software overheads but an MBCF packet is not restricted to an access corresponding to processor's memory operation. Therefore the MBCF systems enjoy opportunities optimizing the number of communication packets and amount of communication data using this packet flexibilities. To put it concretely, large data can be handled in an MBCF operation and multiple MBCF operations can be merged into one communication packet. We call this merged packet a "combined packet" and this optimizing technique "combining". If the system has rather poor communication hardware comparing with processor power, these optimization opportunities are vital for efficient executions.

2.2 Protection and Security Mechanism

In the MBCF scheme, the MBCF-dedicated kernel-level interrupt routine makes a final access to the target space on the remote memory. If strong protection and security are needed in the system, powerful capability-check or authentication procedures can be added to the interrupt routine. However this addition increase the overhead of the MBCF interrupt routine. To prevent the overhead from increasing, we developed another smart method exploiting page-aliases and simple access-key.

In parallel processings, when some errors occur in an activity, further execution of related activities is probably meaningless. Owing to this characteristic a simple protection mechanism which separates the task from other unrelated tasks is enough for parallel processings. In order to prevent the bad influence of the errors from spreading to other tasks, the MBCF uses logical address spaces with memory management mechanism. Only the memory areas mapped to the target task can be accessed through the MBCF. To protect the memory from attacks of other tasks, we adopted unique access-key which represents the right to access the target memory-space.

On the other hand, in distributed processings (e.g. client-server model) the server activity must be protected from errors and attacks of client activities. In this case a strict protection mechanism which distinguish the working area of a client from the others' areas is required. We solved this issue using unique access-key and page-aliasing. Owing to the lack of space we describe only basic outline of the scheme here. When the server is requested MBCF communication from an untrusted client, in the same node the server creates an agent (another activity which has an independent memory-space) which deputizes for it on the communication with the client. Then the server allocates the working memory-area for communication with the client and the same area is also mapped for the agent. In other words, the area is intra-node shared-memory between the server and the agent using page-alias mechanisms. After these preparations on the server node, the agent informs the client of the agent's access-key (not the server's access-key), and the client communicate with the server using the agent's memory-space. Even if the client intends to destroy the server, it can only damage the agent's space and cannot stop the execution of the server activity.

On the MBCF scheme and the MBP scheme, protection and virtualization in communications and synchronizations are replaced with those of memory accesses. This replacement makes high-speed implementations of the mechanisms feasible. Especially on the MBCF scheme, since the TLBs and the MMUs of the node processors are exploited for translations of remote accesses, no additional hardware mechanisms are required.

2.3 Virtual Global Address of the MBCF System

In the MBCF scheme, communications and synchronizations are performed through virtual inter-node memory locations. An address of some location is specified by the combination of a logical-task-ID and a logical-address in the target logical-task and we write the combination as "(Ltask:Laddr)". The task is an abstraction of a processor's activity and has its own memory-space, and it belongs to a node in the MBCF system. In the MBCF system, a task is specified by the combination of a physical-node-ID and a physical-task-ID in the physical-node, we write the combination as "(Pnode:Ptask)". In user-level application programs, only Ltask is used to specify a task. It is the reason why the additional virtualization enables the MBCF system to migrate tasks among nodes. The OS for the MBCF system maintains one translation table for each task, and the table represents the correspondences between Ltasks and (Pnode:Ptask)s. When some tasks are migrated from their original nodes to other nodes, the OS updates the tables which has entries on the migrated tasks.

Ltask notations for MBCF applications are local and virtual identifiers for individual tasks, then (Pnode,Ptask) notations are used in MBCF inter-node packets. Therefore, in the MBCF packet, the notation of a global address is "(Pnode:Ptask:Laddr)".

3 High-Speed Implementation Techniques of the MBCF

To make implementations of the MBCF as high-performance as possible, we apply many techniques on software engineering and exploit advanced architectural features of latest commercial processors. In this section we list and explain these techniques in a lump.

- **Direct accesses to target logical spaces**
  The most distinctive feature of the MBCF is that a requester (sender) specifies the logical address of the target location to which some specified operation is applied. Since there is no intermediate queue structure, cost of queue operations can be omitted. If queue structures are intrinsically needed for some application, the Memory-Based FIFO which will be explained in the next section can be efficiently utilized in the MBCF scheme.
- **On-memory-synchronizations**
  It is the basic scheme of synchronizations in the MBCF to use synchronization variables on memory of the target task. They are updated in the MBCF interrupt routine without any other side effects for synchronizations. The target task can check synchronizations with only direct accesses to the local variables. In the best cases where MBCF operations for the synchronizations are completed before the accesses for checking the variables, on-memory-synchronizations of the MBCF can reduce the overheads of conventional system-calls for inter-node synchronizations. If a blocking-type synchronization is required for avoiding wasteful spin-

wait, the Memory-Based Signal or MBCF_WRITE with scheduling option can be used. They will be explained in the next section.

- **Cache-conscious programming**
  On the latest processors, cache-miss penalties are very large comparing with the average instruction cycles. The programs of the MBCF-dedicated system-call and the MBCF-dedicated interrupt routine are developed to avoid cache-misses as much as possible.
- **MBCF-dedicated requesting system-call**
  System-calls of the conventional OSs like UNIX have the large overheads, because there are many copy-operations which are added only for ease of programming and unnecessary operations which are only for portabilities or system-level maintenances (e.g. signal check or accounting). The MBCF-dedicated system-call does not include such wasteful operations at all.
- **MBCF-dedicated receiving interrupt routine**
  The situation of the MBCF-dedicated interrupt routine is similar to that of the system-call. The interrupt routine neither include unnecessary operations. It utilizes only dedicated-subroutines which are optimized for the MBCF operations and most of subroutines are inlined.
- **Wide but fixed variety of MBCF functions**
  In the MBCF system, user-customizations of the MBCF functions are prohibited. This restriction enable the MBCF interrupt routine to operate the target address directly within the kernel mode. Complicated and heavy functions are realized with combinations of the system-provided commands.

The above techniques do not assume any special-mechanisms of processors and they are applicable to all computers. The following mechanisms are useful for high-speed implementation of the MBCF, and they are implemented in most of latest advanced processors (e.g. SuperSPARC[7], UltraSPARC[8]).

- **TLB corresponding to coexistence of multiple contexts**
  Each entry of the Translation Look-aside Buffer (TLB) in a recent processor has a field for context identifier (context-ID) and the OS for the processor can switch contexts without clearing entries of the TLB. In the MBCF interrupt routine, each MBCF packet requires a few page-entries to be used for direct memory-accesses to the target space. With this type of TLB, the MBCF interrupt routine replaces at most only a few entries of the TLB. (If the MBCF operations access the same locations frequently, it is likely that the page entries corresponding to the locations are resident in the TLB) The cost of the replacement itself is small and the influence of the MBCF operations is also very small after the MBCF interrupt.
- **Physical-address-tagged cache**
  Recent processors have internal caches with physical-address-tags and the OS for the processors can switch contexts without purging entries of the caches.
- **Light-weight context switching**
  This item is strongly related to the preceding two items. There are no costly operations for switching contexts or address-spaces of advanced processors. Therefore, an access to a different context (space) only costs almost the same as much as an access within the current space.
- **User-privileged memory-access capability in kernel mode**
  Some smart processors can perform user-privileged memory-accesses in the kernel mode. If a processor has the facility, without range-checks the the kernel programs can access user-spaces through user-specified pointers.
- **Page aliasing capability**
  Since the mechanisms for page aliasing among multiple memory-spaces are used to realize copy-on-write facilities or shared-memory facilities, most of processors have the page-aliasing facilities. As described before, in secure operations a MBCF-requester uses the dummy space to which only the limited and selected pages within the original space are allocated.
- **Registers dedicated for system-calls or interrupts**
  In some advanced processors (e.g. UltraSPARC) there are spare registers for interrupts or exceptions. Part of ordinary registers are superseded by the spare registers at the beginning of the interrupt routine or the exception routine, and are restored at the end of the routine. If we develop the routines only with the extra registers, overheads of the context switching are dramatically reduced. Moreover, if there are many sets of the spare registers corresponding to the types of events, important pointers and parameters can be resident in the registers.

If some architectural mechanisms described above are absent from a processor, there are some performance degradations but the MBCF can be implemented with software emulations of the mechanisms.

4 MBCF Functions

In this section we list commands, options and supporting commands of the MBCF. A command specifies a main function at the target location in the target node. Some options are attached to the command and give some additional meanings to it. Several supporting commands are implemented to complement the MBCF system, and they are used for preparations of the MBCF communication, modifications of the task-translation table, and user operations of the MBCF queue structures.

There is no limitation on the variety of the MBCF functions though heavy commands are unsuitable to the system from the viewpoint of fairness of task-schedulings. The operations in an MBCF interrupt routine are regarded as part of its target task. The operations of an receiving interrupt in the conventional system are also asynchronous and disturb the execution of the interrupted task. If the cost of the MBCF receiving routine is comparable to that of the conventional one, degrees of fairness in the MBCF system is considered to be similar to those of the conventional system.

4.1 Variations of the Commands

There is a wide variety of the MBCF commands. We list them in the pages that follow.

4.1.1 MBCF_WRITE

MBCF_WRITE is a usual remote-write operation and a basic and simple command of the MBCF. The data-size of one command is specified arbitrarily but the maximum data-size is restricted to be under the capacity of one packet of the

network. This restriction on the data-size is valid for other MBCF commands.

4.1.2 MBCF_READ

MBCF_READ is a usual remote-read command. This command is followed by a reply packet which writes the read-data to the pre-specified location in the requester node.

4.1.3 Memory-based atomic commands

The variations of the atomic commands are as follows.

- **MBCF_SWAP**
- **MBCF_FETCH_ADD**
- **MBCF_COMP_SWAP**

When a packet with one of these commands arrives to the target node, the MBCF interrupt-routine atomically execute specified operations to local data at the target location with the data in the packet and returns a packet with the local data if necessary. In memory-based (MBCF and MBP) scheme, since all atomic operations are basically executed in the target node without any interlocks (like bus locks), multiple atomic operations to the same location can be simultaneously invoked and executed in the pipeline-like manner.

4.1.4 Page-based multicasting commands

The variations of the multicasting commands are as follows.

- **MBCF_UPDATE**
- **MBCF_INVALIDATE**

Each memory-page which is allocated for the multicasting commands has a multicasting pattern. The delivery pattern consists of inter-node links which form a tree-route. When a multicasting command arrives at a node, the MBCF interrupt routine updates or invalidate the specified page and the multicasting packets are copied and transmitted to the outside pages which are specified in the multicast pattern. We call the multicasting technique with this delivery pattern a "hierarchical multicasting"[3, 4].

In the case that a multicasting command is used for protocols of the DSM system, write operations in copy-pages (i.e. pages except for tree-roots) are forwarded to the home-page (i.e. tree-roots of multicast patterns) at first and then the multicasting packets with the command are invoked from the home-page.

In the case that the status report option or the acknowledge option for memory barriers is accompanied with a multicasting command, the "acknowledge combining" technique [3, 4] is used for collecting reports or acknowledges. Acknowledges (or reports) from leaf nodes (leaf pages) are returned to the immediate parent node of the multicasting pattern but the original requesting node, and the parent node waits for all arrivals of acknowledges from all immediate children, then the parent merges the acknowledges and transmit a combined acknowledge to the parent of the parent node. This combining technique cancels the hot-spots of concentrating and collecting acknowledges at the original requesting node (or home node).

4.1.5 Memory-Based FIFO (MBCF_FIFO)

MBCF_FIFO is a command for storing data in fifos. In the MBCF scheme, users can create any number of fifos at any locations, MBCF_FIFO command packets do not directly specify the location for storing data but include the target location of the structures which define the fifos. In each fifo structure, pointers of data buffers and status flags are stored and maintained. The MBCF interrupt routine reads the pointers and the flags first, and stores data in the buffer corresponding to the specified fifo structure, and updates the structure. These operations are performed locally and atomically. If the buffer for the specified structure is full, the data in the MBCF_FIFO packet are cancelled and the status report option can inform the requesting task of this cancellation of the command.

Owing to the possibilities of the cancellations, this command cannot guarantee to keep the orders of the MBCF_FIFO commands from the same task intact. If the fifoness (i.e. the characteristic of keeping orders intact) among fifo-data-entry commands from the same requester is needed, users use acknowledgments using the status report option or MBCF_FIFOe commands in the next item.

For keeping atomicity of the operations, users are needed to use some light-weight system-call for reading data in the buffers of the memory-based fifos. This system-call is a member of the MBCF supporting commands. If protection is required for the fifos, memory-pages for fifo structures are set as read-only.

4.1.6 MBCF_FIFOe

MBCF_FIFOe consists of two commands and these commands are applied to the eager usage of memory-based fifos with keeping the fifoness among the commands from a same requester. MBCF_FIFOe commands exploit the same structures as the MBCF_FIFO command. The differences between MBCF_FIFOe and MBCF_FIFO are only in the method handling the buffer full. **MBCF_FIFOe_NORMAL** is a usual data-entry command of the MBCF_FIFOe. Once a cancellation of the MBCF_FIFOe_NORMAL command occurs, the consecutive MBCF_FIFOe_NORMAL commands will also be cancelled even though there is a room to receive data. A **MBCF_FIFOe_RETRY** command whose target is the stopped fifo resumes the fifo if the buffer has a room to receive data.

The MBCF_FIFOe scheme assumes that requesters maintain the order of sending packets and keep copies of the packets in the node. When status-reports returned from the target tell the requester occurrence of cancellations, the requester resends the cancelled packets in proper order using the copies in the node. Only one status-report in the series of returned packets for cancelled packets carries information that fifo-state has transited to the cancellation state. The requester resend the special MBCF_FIFOe_RETRY command only for the transition report.

If communication system is reliable and sender nodes have no copies for transmitted packets, this MBCF_FIFOe-method can be implemented by copying contents of the requesting packets into cancellation reports. It shows that this flow-control scheme is an extension of the "return-to-sender" method, which is widely used[9, 10, 11], to keep orders of point-to-point packets intact.

4.1.7 Memory-Based SIGNAL (MBCF_SIGNAL)

MBCF_SIGNAL command is accompanied with a remote invocation of the user-specified program in the privilege of the target task. The invocation mechanism of the MBCF_SIGNAL is similar to that of UNIX's signals, and invoked programs are executed only in the scheduling periods of the target task.

MBCF_SIGNAL command corresponds to a memory-based signal structure in the target task, and the structure has three types of subcommands (an invocation-type, a parameter-store-type and an invocation-condition), an

invocation-program-counter and pointers for a data-buffer. These parameter in the signal structures form a wide variety of MBCF_SIGNAL functions

For protection and security, the target task set up the structure of the MBCF_SIGNAL and is able to restrict capabilities of the MBCF_SIGNAL command. Since the information in the signal structures is important and critical, memory-pages for signal structures should be set as read-only.

The invocation-type subcommand specifies the type of invocation programs, i.e. one of User-Level Scheduler (ULS), Destination-Specified Program (DSP) and Source-Specified Program (SSP). ULS is the special program for scheduling activities (threads) in the target task. At most one ULS is enrolled to the OS per task by the target task itself. DSP is an ordinary user-level subroutine of the target task. The entry-instruction-pointer of the DSP is included in the signal structure. SSP is also an ordinary user-level subroutine of the target task. The entry-pointer of the SSP is specified by the requester and included in the MBCF_SIGNAL packet.

MBCF_SIGNAL commands can not only invoke programs but also deliver data like MBCF_WRITE or MBCF_FIFO. The parameter-store-type subcommand controls actions of storing data into buffers or fifos whose pointers are also maintained in the signal structures. In some configurations of the signal structures information (Ltask, Pnode and Ptask of the requester) about received MBCF_SIGNAL packets is also stored to the buffer or the fifo.

The last subcommand specifies conditions of user-program invocations. The invocation-condition subcommand controls the exclusion property of the invocation programs corresponding to a signal structure. If a preceding invocation-program is still running at arrival of the new MBCF_SIGNAL command to the same structure, on the command there are three options: 1. refusing to receive the command, 2. storing data and the information to the fifo (buffer) but not invoking programs, 3. storing data and invoking programs. The invocation-condition subcommand selects one of the three options.

4.2 Variations of the Options

Three types of options can be specified to each MBCF command. We list the options below.

- **Replying status report** This option decides whether a status reply of the command are returned to the requester or not. If it is returned, it is stored to the location pre-specified by the requester. The requester use the stored status to know the completion of the MBCF command or the occurrence of some failure at the target task. To reduce traffic of status reports, users can specify **REPLY_ON_FAILURES** in which an MBCF command replys a failure status only when some failure occurs in the target task during the MBCF procedure.

- **Elastic memory barrier** This option enable users to perform elastic memory barrier (EMB)[12] synchronizations with MBCF commands. The EMB is an extended memory barrier synchronization. If the requester specify the use of EMB and the EMB color of the MBCF command, the request-counter of the color is incremented, and the reply-counter of the color is incremented after arrival of the reply packet. Users compare these counters to know the preceding MBCF commands in the color are all finished.

- **Task scheduling** This option is for the blocking-type synchronizations. If this option is set and the target task is not in ready-state or active-state, the MBCF interrupt routine wakes the task up. With requesters' specifying this option, the target task may block its execution without setting any hooks for the re-scheduling.

4.3 Supporting Commands for Initializations

We list two supporting commands of the MBCF which are for initial remote-messages. Other supporting commands (system-calls for the MBCF) are described in the explanation of the main MBCF commands related to them.

In the usual distributed processings, a client task knows neither the access-key of the server nor locations of the working-area to be accessed. Therefore, the initial message to the server cannot take the MBCF style, and some special messaging method is needed. Conventional communication methods (e.g. TCP/IP) can be used only for the initial communication, but a special MBCF_WRITE which is called **WRITE_TASKQUE** is prepared in the system owing to strengthen the securities of the MBCF. The tasks which want to start MBCF communications should use the WRITE_TASKQUE command first. If some violation of memory access is occurred in the MBCF processings after the establishment of communication channels, the responsibility of the violation rests with the task which issued the WRITE_TASKQUE (earlier if both issued). Furthermore, the task which receives WRITE_TASKQUE packet from some requester acquires the strong privilege with which it can cancel the execution of the requester task.

Each task can create at most one special fifo-queue whose name is "TASKQUE" in the kernel-space. When a WRITE_TASKQUE command arrives, the contents of the packet is stored in the TASKQUE of the target task if the task has created it. The target task can read the data from the TASKQUE by the **READ_TASKQUE** light-weight systemcall. This system-call is also a supporting command of the MBCF.

5 Qualitative Comparison with the Message-Passing-Type Communication Mechanisms

In most conventional systems, message-passing-type interfaces are popular not only as user-programming-interfaces but also as system-interfaces (in other words, kernel-user-interfaces or system-call-interfaces). The MBCF is primarily a system-interface but can be directly used as a programming interface. By using additional user-level codes, any message-passing-type interface can be realized in the MBCF scheme. Conversely, all functions of the MBCF are carried out with any message-passing-type system-interface and some user-level additional codes. The selection of programming-interfaces is only an issue of taste of programmers or language-designers. Therefore, the problem is which type should be supported as the system-interfaces of the parallel and distributed systems.

In this subsection we abbreviate Message-Passing-type System-Interfaces as "MPSI". The MPSI is less flexible than the MBCF. We explain the difference on the flexibility-issue in the MBCF words,

- The MPSI limits target logical addresses to only one (or a few) implicit message-buffer address.
- The MPSI also limits functions to only one-type:"MBCF_WRITE" (simple remote write).
- In the MPSI system the correspondences among "send"s and "receive"s are essential. Hence the execution order of them is inflexible. In the MBCF scheme a simple function can be encapsulated into an atomic MBCF-command and the placement of these commands is more elastic than "send"s and "receive"s of the MPSI.

These differences on flexibilities often cause the big difference on performance. For example, because the implicit message-buffer of the MPSI is implemented in kernel space, the data should be copied into another buffer in user-space when user want to use them. From the inflexibility on the target-address specifications, the number of data-copies in the MPSI is essentially greater than the MBCF.

To be flexible, the MBCF accesses directly to the user-spaces from the interrupt routine (kernel-mode). However, recent high-end processors (like SuperSPARC[7], UltraSPARC[8]) has the following mechanisms which can realize user-level memory-accesses in the kernel-mode without penalties.

- the processor in the kernel-mode can perform memory-accesses with user-privilege without paying penalties.
- Many pages from various task-spaces can exist at the same time in processor's TLB.
- Changing the current context is inexpensive and costs only one instruction and a few clock-cycles.

Therefore, without paying any penalties the MBCF get much more flexibility than the MPSI. In other words, the MBCF interface is much better than the MPSI.

6 Qualitative Comparison with the Active Message

The Active Message[13] (AM) is originally invented to perform high-speed executions of dataflow-type programs directly on the bare hardware of parallel computers, and there is no mechanism for protection, security or virtualization. The primary feature of the AM is that a user-level receive-routine is selected for each message and the receive-routine for the message is specified in the message itself (the entry pointer of the receive-routine is included in the message).

The SparcStation Active Message[14] (SSAM) is an extension of the AM for workstation clusters with general-purpose OSs. In the SSAM scheme a message (packet) is prepared explicitly by users and transmitted through the SSAM-dedicated system-call. This packet-sending procedure of the SSAM is almost same as the MBCF but there is no access-key for security nor virtualization for task-migrations. The receiving procedure of the SSAM is much different from the MBCF. Because the receiving interrupt-routine is executed in kernel-mode and operates HW registers of the NIC, the user-level receiving-routine which is specified in the SSAM message must be invoked indirectly through some kernel-level interrupt-routine. A kernel-level temporal receiving-routine is invoked at every interrupt of the NIC, and the routine receives the packet to the target-task's buffer in the kernel-user-shared-space. After receiving the packet in the buffer, the mechanism like the signal of the UNIX is used to invoke the specified user-level receiving-routine, and the user-level routine performs specialized functions using the data in the temporal buffer. On the other hand, while guaranteeing protection and virtualization, the MBCF-receiving routine is directly invoked in kernel-mode when interrupts of packet arrivals occur. Since user-customizations of functions are prohibited in the MBCF system, the receiving-routine can be kept safe and fair, and upper limits of operation costs are known before execution by kernel.

Copies to the temporal buffers of the SSAM are additional overheads comparing with the MBCF. Moreover, in the SSAM system there is a possibility that the invocation of the receiving-routine and the reply of the message are delayed since the routine is invoked and executed only when the target task is scheduled in the core. In the cases of simple remote-memory-accesses, it's most likely that the cost for invocations of user-level receiving-routines is another overhead. On these three points the SSAM is qualitatively inferior to the MBCF. On the contrary the flexibility that receiving-functions are able to be customized perfectly is the strong point of the SSAM. In the MBCF system, however, there is no limitation of the MBCF-command varieties and critical functions can be added in the lineup. Therefore, the perfect customizability of receiving functions is less significant than the number of data-copies is.

7 Preliminary performance evaluation

7.1 Implementation environment of the MBCF

We have developed the MBCF system using the Fast Ethernet (100BASE-TX)[15] and its supporting operating system. The Ethernet system cannot guarantee the arrival of transmitted packets. The MBCF/100BASE-TX has a considerably complicated protocol which guarantees packet-arrival and fifoness of point-to-point communications, but the buffering mechanisms of the MBCF/100BASE-TX can avoid performance degradations in usual unsaturated communications. Moreover cache-conscious programming also prevents the overhead of the complicated protocol to become large. We use the following hardware environment.

- Node of the NOW
  - Axil 320 model 8.1.1 (Sun SPARCstation20 compatible, 85MHz SuperSPARC x 1)
  - Fast Ethernet SBus Adapter 2.0
- Network
  - Hub (Non-switching or Switching) connection of 100BASE-TX and/or 10BASE-T
  - Non-switching hubs are used in the measurements unless otherwise specified.
- Operating system of the NOW
  - SSS–CORE/NOW Ver.1.0[16]
    - General-purpose scalable operating system
    - Using time-sharing system and partitioning system together
    - Fair and efficient scheduling scheme for multiple parallel tasks[16, 17]
    - High optimizability for user-programs
    - Test bed of the MBCF and the ADSM[2, 18]
    - Development from scratch to attain high-speed implementations

Table 1: Peak bandwidthes of the MBCF/100BASE-TX

data size (byte)	4	16	64	256	1024	1408
100BASE-TX (Mbyte/s)	0.29	1.06	4.03	8.28	10.86	11.24

Figure 1: NOW with the SSS-CORE/NOW Ver.1.0 where full set of the MBCF is implemented

7.2 Peak Bandwidth of the MBCF/100BASE-TX

Table 1 shows peak bandwidthes of the MBCF/100BASE-TX. We measure the bandwidth using the MBCF_WRITE commands with various data-size. Figures of the table are net quantities which correspond to only payload data without packet header of the MBCF/100BASE-TX or additional data for Ethernet protocol. The measurement method is that a requester repeatedly sends MBCF_WRITE commands to a fixed target without checking any acknowledges except for an acknowledge per 16 transmissions. The acknowledgments every 16 transmissions are used to avoid saturation of the Ethernet.

7.3 Latency of the MBCF/100BASE-TX

We show one-way latencies of the MBCF/100BASE-TX in Table 2. We measured latencies in the tables using three commands: MBCF_WRITE, MBCF_FIFO and MBCF_SIGNAL. Originally we measured the round-trip latencies and calculated one-way latencies from them. MBCF_FIFO latencies include cost of one light-weight system-call for reading fifo data. MBCF_SIGNAL latencies include cost of one invocation of user-level subroutine and one light-weight system-call for reading fifo data.

Table 2: One-way latency of the MBCF/100BASE-TX

data size (byte) command	4	16	64	256	1024
MBCF_WRITE (μs)	24.5	27.5	34	60.5	172
MBCF_FIFO (μs)	32	32	40.5	73	210.5
MBCF_SIGNAL (μs)	49	52.5	60.5	93	227.5

7.4 Quantitative Comparison with User-level Communication Mechanisms of the MPPs

In this subsection we compare the MBCF/100BASE-TX with several user-level communication mechanisms of the commercial Massively Parallel Processors (MPPs). Those mechanisms of the MPPs have a greater or a less degree of limitations, for examples some mechanisms force an application to use them exclusively, others force the OS to allocate tasks only in gang-scheduling manner. Besides all mechanisms except for Active Message type ones are message-passing style and there is no variety of functions but message-sending. Therefore, in a qualitative comparison the MBCF/100BASE-TX is more protected, virtualized and varied than the mechanisms of the MMPs, and superior to them.

Table 3 shows the quantitative comparisons on peak bandwidth and round-trip latency. All the machines in the table except the SS20 clusters (SSAM[14] and MBCF) have original high-speed communication hardwares which are much faster than the 100BASE-TX system. The SSAM system use a 156Mbps ATM NIC which is also faster than the NIC of the MBCF/100BASE-TX.

The figures in the table except those in the MBCF row are quoted from following papers. Figures without any marks are from the paper[14], and figures on SP–1/SP–2 (with ‡ mark) are from the paper[19].

The implementation of the SSAM in the table have no mechanism to guarantee packet arrival or FIFO property. The practical SSAM would suffer larger overheads than the SSAM in the table. The SP–2 has two entries in the table: “MPL/p” is a method such that an application exclusively uses the SP–2’s high-speed communication hardware, “MPL/udp” is a method where the communication library uses the UDP interface of the SP–2’s OS. The former is not worth calling a “virtualized” interface, and the latter should be compared with the MBCF.

Considering two points:

- As for the level of protection and virtualization, the MBCF is the highest of all, and
- As for the performance of the raw communication hardware, the MBCF is the lowest of all,

figures in the table 3 show that methodology and implementation techniques of the MBCF/100BASE-TX are excellent and remarkable.

8 Concluding Remarks

We have proposed a user-level high-speed communication and synchronization scheme: Memory-Based Communication Facilities (MBCF). In the MBCF scheme protection and virtualization in communications and synchronizations are replaced with those of memory accesses. This replacement makes high-speed implementations of the scheme feasible, since advanced architectural mechanisms of processors for memory-accesses can be exploited. Moreover, memory-based operations have large flexibilities which may increase the performance, and qualitatively the MBCF is superior

Table 3: Basic performance of user-level communication mechanisms

Machine + communication software	Peak bandwidth (Mbytes/s)	Round-trip latency(μs)
SP–1 + MPL/p	8.3 / 8.7‡	56 / 75‡
Paragon + NX	7.3	44
CM–5 + Active Message	10.0	12
SP–2 + MPL/udp	10.8‡	554.0‡
SP–2 + MPL/p	35.5‡	78.0‡
SS–20 cluster + SSAM (156Mbps ATM)	7.5	52
SS–20 cluster + MBCF (100BASE-TX)	**11.2**	**49**

not only to message-passing-type system-interfaces but also to generalized active messages.

We also proposed novel two communication and synchronization mechanism: the Memory-Based FIFO and the Memory-Based Signal, The Memory-Based FIFO is more general and virtualized than conventional fifo-queue-type communication interfaces, and has the generalities without paying any penalties. The Memory-Based Signal is an internode and memory-based extension of UNIX signals and it is promising primitive for making high-speed Remote Procedure Call and/or Remote Method Invocation.

We have developed the MBCF using a commodity communication hardware: the Fast Ethernet (100BASE-TX). Though the MBCF is protected and virtualized as completely as memory, it shows about the same performance as or better than user-level communication mechanisms installed in MPP systems which have dedicated communication hardwares.

Acknowledgment

This work is partly supported by Advanced Information Technology Program (AITP) of Information-technology Promotion Agency (IPA) Japan and by Real World Computing Partnership (RWCP) Japan.

References

[1] T. Matsumoto and K. Hiraki: Memory-Based Communication Facilities of the General-Purpose Massively Parallel Operating System: SSS–CORE (in Japanese). *Proc. of 53rd Annual Convention of IPS Japan*, Vol.1, 5B–3, pp.37–38 (September 1996).

[2] T. Matsumoto, T. Komaarashi, S. Uzuhara, and K. Hiraki: The Asymmetric Distributed Shared Memory Using Memory-Based Communication Facilities (in Japanese). *Proc. of Computer System Symp.*, IPS Japan, pp.37–44 (November 1996).

[3] T. Matsumoto and K. Hiraki: A Shared-Memory Architecture for Massively Parallel Computer Systems (in Japanese). *IEICE Japan SIG Reports*, Vol.92 No.173, CPSY 92–26, pp.47–55 (August 1992).

[4] T. Matsumoto and K. Hiraki: Distributed Shared-Memory Architecture Using Memory-Based Processors (in Japanese). *Proc. of Joint Symp. on Parallel Processing '93*, IPSJ/IEICE/JSSST, pp.245–252 (May 1993).

[5] T. Matsumoto and K. Hiraki: Dynamic Switching of Coherent Cache Protocols and its Effects on Doacross Loops. outlines of the MBP in section 5, *Proc. of the 1993 ACM Int. Conf. on Supercomputing*, pp.328–337 (July 1993).

[6] T. Matsumoto, K. Nishimura, T. Kudoh, K. Hiraki, H. Amano, and H. Tanaka: Distributed Shared Memory Architecture for JUMP–1:A General-Purpose MPP Prototype (Invited Paper). *Proc. of Second Int. Symp. on Parallel Architectures, Algorithms and Networks (I-SPAN'96)*, IEEE Computer Society Press, pp.131–137 (June 1996).

[7] Texas Instruments: *SuperSPARC User's Guide*. SPKU005, Texas Instruments (October 1992).

[8] Sun Microelectronics: *UltraSPARC-I User's Manual*. STP1030-UG, Sun Microelectronics (January 1996).

[9] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith: The Tera computer system. *Proc. 1990 Int. Conf. on Supercomputing* (June 1990).

[10] Cray Research, Inc.: Cray T3D System Architecture Overview. (March 1993).

[11] S. Pakin, M. Lauria, and A. Chien: High Performance Messaging on Workstations: Illinois Fast Message (FM) for Myrinet. *Proc. Supercomputing'95* (December 1995).

[12] T. Matsumoto and K. Hiraki: Elastic Memory Consistency Models (in Japanese). *Proc. of 49th Annual Convention of IPS Japan*, Vol.6, 5K–3, pp.5–6 (September 1994).

[13] T. von Eicken, D. E. Culler et al.: Active Messages: A Mechanism for Integrated Communication and Computation. *Proc. 19th Int. Symp. on Computer Architecture*, pp.256–266 (May 1992).

[14] T. von Eicken, A. Basu, and V. Buch: Low-Latency Communication Over ATM Networks Using Active Messages. *IEEE Micro*, pp.46–53 (February 1995).

[15] IEEE: IEEE Std 802.3u-1995 CSMA/CD Access Method, Type 100BASE-T. IEEE, New York (October 1995).

[16] T. Matsumoto and K. Hiraki: Resource management schemes of the general-purpose massively-parallel operating system: SSS–CORE (in Japanese). *Proc. 11th Conf. of Japan Society for Software Science and Technology*, pp.13–16 (October 1994).

[17] Y. Nobukuni, T. Matsumoto, and K. Hiraki: Resource Management Methods for General Purpose Massively Parallel OS *SSS–CORE*. *Proc. of International Symposium on High Performance Computing*, Springer-Verlag LNCS 1336, pp.255–266 (November 1997).

[18] J. Niwa, T. Inagaki, T. Matsumoto, K. Hiraki: Efficient Implementation of Software Release Consistency on Asymmetric Distributed Shared Memory. *Proc. of Int. Symp. on Parallel Architectures, Algorithms and Networks (I-SPAN'97)*, IEEE Computer Society Press, pp.198–201 (December 1997).

[19] M. Snir et al.: The Communication software and parallel environment of IBM SP2 *IBM Systems Journal*, Vol. 34, No. 2, (1995).