

Memory-Based Communication Facilities and Asymmetric Distributed Shared Memory

Takashi MATSUMOTO
Department of Information Science
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113 Japan
tm@is.s.u-tokyo.ac.jp

Kei HIRAKI
Department of Information Science
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113 Japan
hiraki@is.s.u-tokyo.ac.jp

Abstract

In general-purpose parallel and distributed systems, performance of the protected and virtualized user-level communications and synchronizations is the most crucial issue to realize efficient execution environments. We proposed 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. The MBCF realizes the direct remote-memory-accesses in user-task-space and offers programmers and compilers a large shared-memory space. This paper describes outlines and characteristics of the MBCF, and evaluates basic performance of the MBCF/Ether which is the first sample implementation with 100BASE-TX interfaces. The evaluation tells that its peak bandwidth at half-duplex mode is 11.2Mbyte/sec and its round-trip latency is 49μsec. Finally, we introduce a brand-new remote cache scheme “Asymmetric Distributed Shared Memory (ADSM)”, which is suitable for the MBCF and optimizing compilers, and show the effects of optimization methods for the ADSM.

1. Introduction

The Memory-Based Communication Facilities (MBCF) [1, 2] is a software-only solution for realizing protected and virtualized high-speed user-level communication and synchronization. The MBCF is developed for Network of Workstations (NOW) and distributed memory multiprocessors without hardware Distributed Shared Memory (DSM) mechanisms.

So far, many hardware DSM mechanisms are proposed but no DSM mechanism has been proved to be sufficiently general for wide range of applications. Recent WSs and PCs are very powerful but inexpensive, and the performance of

their network interfaces improves rapidly. Therefore they can be used as nodes of parallel computers. There are no inexpensive hardware DSM mechanisms for making WS/PC clusters, consequently we need alternative methods for efficient user-level communication and synchronization without specialized hardware mechanisms.

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.

There are two factors which produce the major part of overheads on user-level communication and synchronization. The first one is a methodological aspect (including functionalities, protocols, packet format). Conventional user-level communication interfaces (for examples, TCP/IP, UDP/IP and MPI) are message-passing-type ones, and their functions are limited to remote-write operations into specific message-buffer addresses in the kernel-space. To break out of this limitations, we adopt memory-based operations where arbitrary target addresses and a 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. The other factor is a software engineering aspect (implementation methodology). In the conventional OSs, communications and synchronizations among nodes (machines) are regarded as usual I/O events like disk operations, and device-drivers for communications and synchronizations have the same data and control structures as device-drivers for other I/O devices. Consequently, the device-drivers suffer large overheads that are not necessary for functions of communication and synchronization.

To realize high-performance implementations, the MBCF-dedicated system-calls and the MBCF-dedicated interrupt routines have been developed and used, then there is no operation irrelevant to functions of the MBCF.

2. Memory-Based Communication Facilities

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, 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 an inter-node communication-style packet and transmit 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 the 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 execution.

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, protections and virtualizations 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 (Ltask) and a logical-address (Laddr) in the target logical-task and we write the combi-

nation 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)”.

2.4. 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. We list these techniques below.

- Direct accesses to target logical spaces
- On-memory-synchronizations
- Cache-conscious programming
- MBCF-dedicated requesting system-call
- MBCF-dedicated receiving interrupt routine
- Wide but fixed variety of MBCF functions

The above techniques does 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 the coexistence of multiple contexts
- Physical-address-tagged cache
- Light-weight context switching
- User-privileged memory-access capability in kernel mode

- Page aliasing capability of the MMU/TLB
- Registers dedicated for system-calls or interrupts

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.

2.5. Features of the MBCF

In this subsection we summarize and list the general features of the MBCF which is described in preceding subsections before we show an implementation and its performance using a specific NIC: an Ethernet card.

- protected and virtualized communication and synchronization
- using only commodity hardware
- forming a logical and global shared-memory space
- nonblocking command requests
- fundamental synchronizations based on polling memory-locations
- guaranteeing fifoness of point-to-point communications
- guaranteeing arrivals of transmitted packets
- realizing the MBP’s functions (remote-memory operations)
 - remote-memory-accesses
 - atomic operations which are executed in the target node
 - multi-casting operations using hierarchical multi-casting and acknowledge combining[3]
 - memory-based fifo[3]
 - memory-based signal[3]
 - memory-based primitives specialized to a specific system
- applying high-speed system-software implementation techniques
- exploiting hardware mechanisms of advanced processors if available
- optional capabilities for synchronizations
 - status reports of execution results of the MBCF commands
 - counter maintenance for elastic memory barrier
 - scheduling target tasks into the run-queue of the OS

2.6. 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.

2.7. Qualitative Comparison with the Active Message

The Active Message[9] (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[10] (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 securities 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 protections and virtualizations, 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.

3. Basic Performance of the MBCF/Ether

The Ethernet is the most popular method of local area networks (LANs) and is also promising as candidate for the high-speed communications in workstation clusters and personal-computer clusters. Therefore, we adopted the Ethernet (100BASE-TX[12] and 10BASE-T[11]) as the communication method of the first sample implementation of the MBCF. We call it the "MBCF/Ether". The Ethernet system cannot guarantee the arrival of transmitted packets. The MBCF/Ether has a considerably complicated protocol which guarantees packet-arrival and fineness of point-to-point communications, but the buffering mechanisms of the MBCF/Ether can avoid performance degradations in usual unsaturated communications. Moreover Cache-conscious programming also prevents the overhead of the complicated protocol to become large.

In this section we measure performance (peak bandwidth and latency) using the real system with the MBCF/Ether.

3.1. Environment for the performance evaluation

We use the following workstation cluster to measure basic performance of the MBCF/Ether.

- Node of the NOW
 - Axil 320 model8.1.1
(Sun SPARCstation20 compatible,
85MHz SuperSPARC x 1)
 - Fast Ethernet SBus Adapter 2.0
- Network
 - Non-switching hub connection of (100BASE-TX and 10BASE-T)
 - Switching hubs are also available but were not used in the measurements
- Operating system of the NOW
 - SSS-CORE/NOW Ver.1.0[13, 14]
 - * General-purpose operating system
 - * Scalable system

- * Using time-sharing system and partitioning system together
- * Fair and efficient scheduling scheme for multiple parallel tasks[13, 15]
- * High optimizability for user-programs
- * Test bed of the MBCF and the ADSM[2, 16]
- * Development from scratch to attain high-speed implementations



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

3.2. Peak bandwidth of the MBCF/Ether

Table 1 shows peak bandwidths of the MBCF/100BASE-TX and the MBCF/10BASE-T. We measure the bandwidth using the MBCF_WRITE (remote-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/Ether 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. Ideal values of the peak bandwidths are 12.5Mbyte/s for the 100BASE-TX and 1.25Mbyte/s for the 10BASE-T, and these ideal values correspond to all transmitted data including all headers and all additional signals for protocols. Therefore, the results in the table 1 tell that the performance limits of the NICs and the Ethernet systems are bottle necks of the MBCF/Ether's peak bandwidth.

Table 1. Peak bandwidths of the MBCF/100BASE-TX and the MBCF/10BASE-T

| 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 |
| 10BASE-T (Mbyte/s) | 0.04 | 0.17 | 0.48 | 0.89 | 1.13 | 1.17 |

To compare the MBCF/Ether with ordinary communication methods in the conventional OS(SunOS4.1.4), we measure peak bandwidths of the TCP/IP and the UDP/IP on the same hardware environment. In these measurements we use socket libraries of SunOS4.1.4. Coping with measurements of the fine-grained communications, we add the TCP_NODELAY option to the TCP/IP sockets. Because of guaranteeing fineness and packet-arrival, the same protocol as the MBCF/Ether is added to the UDP/IP transmissions with user-level routines. We call the TCP/IP communication of the 100BASE-TX "TCP100/SunOS". The UDP/IP communication of the 100BASE-TX are called "UDP100/SunOS".

Figure2 shows the peak bandwidths of the MBCF100(MBCF/100BASE-TX), the TCP100/SunOS and the UDP100/SunOS. The x-axis of the figures represents the

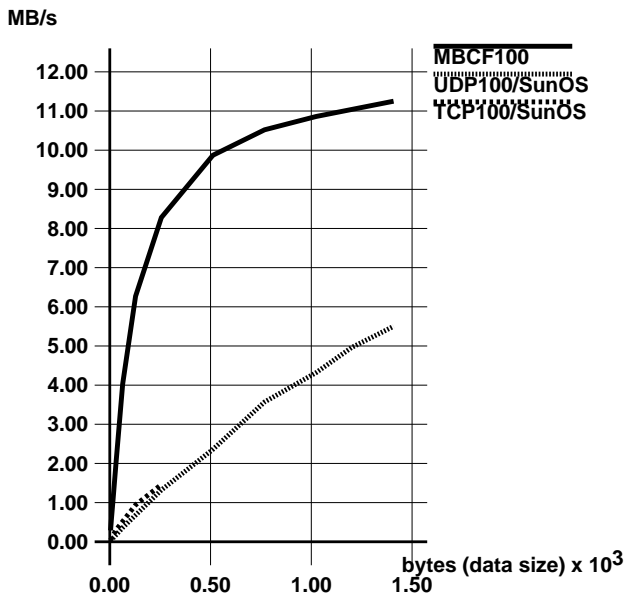


Figure 2. Comparison of peak bandwidth

data size of one ether-packet, and the y-axis is the data transfer rate. For the TCP100 a drastic performance drop is observed at 512byte packet-size. It is due to the TCP-specific protocols (congestion avoidance, slow-start and so forth) and these protocols are poor at eager and repeated transfers. The start slope of the UDP100/SunOS (or the TCP100/SunOS) is much softer than that of the MBCF100. Although the

curves of the MBCF100 is almost saturated at data-sizes over 1024byte owing to the limitation of the hardware, the UDP100/SunOS cannot reach the half of maximum performance of the MBCF100 at the point of the biggest data-size.

3.3. Round-trip latency of the MBCF/Ether

By referencing the clock-counter LSI we measure latencies using the MBCF_WRITE command which is accompanied with status report options. Just before requesting the command we read the start time, and we check the end time just after recognizing the status return by polling. The results of the measurements include the overheads of referencing the clock-timer. In order to compare latencies between the MBCF/100BASE-TX and usual communication methods on the conventional OS, we show Figure3 using the TCP100/SunOS and the UDP100/SunOS which are described in the previous subsection. The x-axis represents data size of one packet, and the y-axis is the latency. Latencies of both the TCP100/SunOS and the

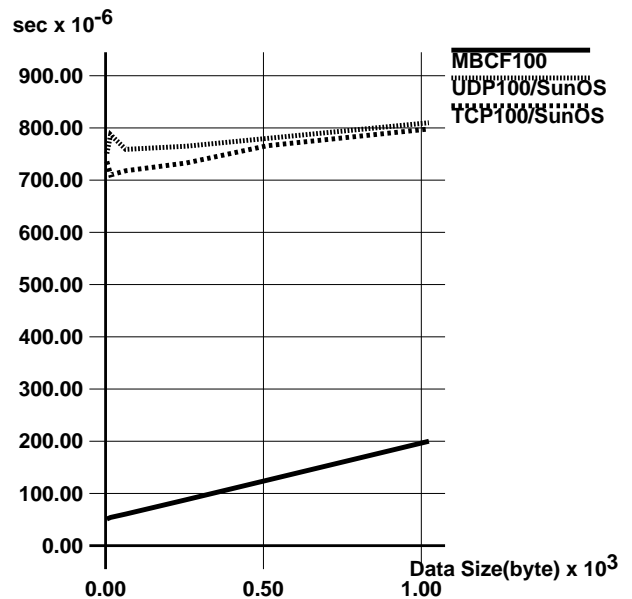


Figure 3. Comparison of round-trip latencies

UDP100/SunOS are more than 700μsec and also are more than 10 times as much as that of the MBCF/100BASE-TX at data-sizes under 64byte. For the curves of the

TCP100/SunOS and the UDP100/SunOS we can hardly find linearities according to data-sizes, because they are hidden in fluctuations of the measurements with non-deterministic factors in the conventional OS.

Next we measure the round-trip latencies using a logic analyzer to acquire accurate values. In this measurement we use the MBCF_WRITE_STAT with fixed 4byte data. The results fluctuate from 48.4 μ sec to 52.0 μ sec according to the cache conditions. The value measured most frequently is about 49.0 μ sec.

3.4. Quantitative Comparison with User-level Communication Mechanisms of the MPPs

In this subsection we quantitatively 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.

Table2 shows the quantitative comparisons on peak bandwidth and round-trip latency. All the machines in the table except the SS20 clusters (SSAM[10] and MBCF) have original high-speed communication hardware 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[10], and figures on SP-1/SP-2 (with ‡ mark) are from the paper[17].

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 protections and virtualizations, 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 table2 show that methodology and implementation techniques of the MBCF/100BASE-TX are excellent and remarkable.

4. Asymmetric Distributed Shared Memory

4.1. Outline of the ADSM

To execute shared-memory-based parallel programs efficiently in a system without hardware-remote-cache mechanisms, some software cache scheme must be performed by the OS and/or user codes. In the MBCF system, considering code optimizations for inter-node communications, the full user-level cache scheme (User-level DSM:UDSM), where the MBCF interfaces are directly used in user-level codes to maintain software-remote-caches, is better than OS-based software DSMs. In other words, the UDSM scheme is more suitable to exploit flexibilities of the MBCF for the optimizations of communication and execution than OS-based DSMs. However, in the UDSM case the user-level execution-code must explicitly maintain, check and modify software-controlled-cache tags. Up to now, neither processors are fast enough to neglect the overhead of handling software cache-tags nor optimizing compilers are sophisticated enough to hide and/or reduce it. Inter-node communications occur only at shared-write situations and in usual applications the number of shared-writes is much less than shared-reads. Considering these characteristics we introduce a brand-new remote cache scheme “Asymmetric Distributed Shared Memory (ADSM)” [14, 2].

In conventional page-based (i.e. OS-based) DSMs, not only read-cache-misses but also shared-writes are supported by the TLB/MMU mechanisms of node processors using write-protection traps and page-fault traps. Though the ADSM is one of page-based cache schemes, only read-cache-misses are supported by the TLB/MMU mechanisms. For each shared-write in the ADSM scheme, a proper sequence of instructions which maintains the cache consistency of the system is inserted into the user-level execution-code by the optimizing compiler. In the MBCF system, the user-level code-sequences include the MBCF-dedicated system-calls and invalidate (or update) remote caches while modifying the local cache-states. Since the instructions for the consistency maintenances at shared-writes are explicitly inserted in the application codes, there is large room for various code optimizations. Strategy of handling shared-reads (read-cache-misses) and that of handling shared-writes are different. Therefore we call this scheme the “asymmetric” DSM.

The combination of the MBCF and the ADSM can realize an efficient distributed shared memory environment on

Table 2. Basic performance of user-level communication mechanisms

| Machine + soft | Peak bandwidth (Mbytes/s) | Round-trip latency(μ s) |
|--|---------------------------|------------------------------|
| 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 |

Network of Workstations or distributed-memory multiprocessors without DSM-dedicated hardware.

4.2. Optimizations for the ADSM

In this subsection we list the code-optimization techniques which are suitable for the ADSM scheme (and the UDSM scheme).

- When shared-writes are performed to the contiguous locations and there is no synchronization point among them, those consistency-management codes can be coalesced[18, 16]. The **coalescing optimization** reduces the runtime overheads of local consistency-maintenances and the number of communications.
- When there are many fine-grain communication packets whose destinations are the same task of the same node, those can be combined into a large packet at the compiling time and/or runtime. The **combining optimization** reduces the number of communications.
- When multiple shared-writes by a node modify the same location between two contiguous synchronization points, they can be neglected except for the last shared-write. The **last-write optimization** reduces the runtime overheads, the number of communications and the amount of transferred data.
- By changing the instructions for consistency maintenance, the compiler can specify various consistency protocols according to the characteristics of target shared-variables in an application program. The **protocol-switching optimization** reduces the number of communications and the amount of transferred data.

4.3. Preliminary evaluations of the optimizations

We have implemented a prototype of the compiler[19] and the runtime system[19] of the ADSM on the SSS-CORE/NOW ver.1.0. The execution environment is the

same as that of the performance evaluation of the MBCF. Because the SSS-CORE/NOW ver.1.0 has not supported user-level page-fault handlers yet, we cannot detect page fault via traps. In order to detect that the processor attempts to access the shared page which is not allocated or invalid, we insert the code checking the corresponding page's validity before each shared access. The messages which request memory-copies or cache invalidations are serviced through the memory-based signal mechanism of the MBCF. As for the cache consistency protocol we adopt the SAURC protocol[14] which is a variation of the LRC protocols[20] and emulates the AURC protocol[21] with explicit communication codes.

We evaluate the performance on LU-Contig and Radix of SPLASH-2 benchmark suit[22] using 4 nodes. The problem size of LU-Contig is a 512×512 matrix with 16×16 blocks and that of Radix is 1M sorting keys. Table4.3 shows the results of LU-Contig and Table4 represents the results of Radix.

For each table, we evaluate three optimization methods for the ADSM:

NO No optimizations,

MB Dynamic combining,

AL Static intra-procedural coalescing, and

IA Static inter-procedural coalescing (which includes AL).

The "Opt" column in the tables expresses the combination of the optimization methods which are applied to the corresponding measurement. The "#CM" column shows the consistency managing time and also shows the number of instructions for consistency management.

Since LU-Contig is a simple application that the processor accesses the contiguous locations, only with intra-procedural analysis the compiler can find many opportunities to coalesce a sequence of instructions for consistency management. On the other hand, static intra-procedural coalescing on Radix can perform only little speed-up. Though

Table 3. Effects of optimization methods on LU-Contig (n=512,b=16)

| Opt | Exec Time (sec) | Number Of Packets | Data Traffic (MByte) | #CM |
|---------|-----------------|-------------------|----------------------|----------|
| NO | 417.54 | 33554528 | 85.12 | 11184770 |
| MB | 56.08 | 189035 | 212.07 | 11184770 |
| AL | 69.90 | 719561 | 84.50 | 753632 |
| MB & AL | 18.37 | 88694 | 99.0 | 744928 |
| IA | 9.10 | 3552 | 6.38 | 2830 |
| MB & IA | 9.03 | 3550 | 6.38 | 2830 |

Table 4. Effects of optimization methods on Radix (#key = 1M)

| Opt | Exec Time (sec) | Number of Packets | Data Traffic (MByte) | #CM |
|---------|-----------------|-------------------|----------------------|---------|
| NO | 34.05 | 3180349 | 42.54 | 9506802 |
| MB | 9.06 | 44357 | 50.89 | 9506802 |
| AL | 30.01 | 3167439 | 42.53 | 9437300 |
| MB & AL | 8.99 | 44248 | 50.73 | 9437300 |
| IA | 3.19 | 21054 | 18.11 | 16480 |
| MB & IA | 2.44 | 9995 | 11.79 | 16480 |

there are large fluctuations on the amount of optimization effects, all optimizations reduce the execution time of the applications and the best execution-time is over ten times faster than no optimization case.

5. Concluding Remarks

We have proposed a user-level high-speed communication and synchronization scheme: Memory-Based Communication Facilities (MBCF). Though the MBCF is protected and virtualized as completely as memory, it is implemented with off-the-shelf communication hardware and it shows about the same performance to dedicated communication as the hardware installed in an MPP system.

In the MBCF scheme protections and virtualizations 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.

We have developed the MBCF using the Fast Ethernet (100BASE-TX), and its peak bandwidth at half-duplex mode is 11.2Mbyte/sec and its round-trip latency is 49 μ sec. Its performance is several times better than usual communication methods in the conventional OS.

We proposed a brand-new remote cache scheme "Asymmetric Distributed Shared Memory (ADSM)", which is suitable for the MBCF and optimizing compilers, and made a prototype compiler with several optimization methods for communications. By the real executions of the compiled

codes, we found that optimized codes can be executed over 10 times faster than non-optimized ones.

In order to improve the performance of the MBCF/Ether, we are now porting the SSS-CORE operating system, which includes the full set of the MBCF/Ether system, to latest workstations with UltraSPARC CPU and a faster network interface (Gigabit Ethernet or Fibre Channel).

Acknowledgment

This work is supported by Advanced Information Technology Program (AITP) of Information-technology Promotion Agency (IPA), Japan. Authors thank Mr. Junpei Niwa and Mr. Tatsushi Inagaki for developments of the optimizing compiler and performance measurements of the compiled codes. Authors also thank Mr. Shigeru Uzuhara for his collaboration on developments of the SSS-CORE operating system.

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] 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).
- [10] T. von Eicken, A. Basu, and V. Buch: Low-Latency Communication Over ATM Networks Using Active Messages. *IEEE Micro*, pp.46-53 (February 1995).
- [11] ISO/IEC: ISO/IEC 8802-3: 1996 (ANSI/IEEE Std 802.3, 1996 Edition) CSMA/CD. IEEE, New York (July 1996).
- [12] IEEE: IEEE Std 802.3u-1995 CSMA/CD Access Method, Type 10BASE-T. IEEE, New York (October 1995).
- [13] 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).
- [14] T. Matsumoto et al.: A General-Purpose Massively-Parallel Operating System: SSS-CORE — Implementation Methods for Network of Workstations — (in Japanese). *IPS Japan SIG Reports*, Vol.96 No.79, OS-73-20, pp.115-120 (August 1996).
- [15] 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).
- [16] J. Niwa, T. Inagaki, T. Matsumoto, and 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).
- [17] M. Snir et al.: The Communication software and parallel environment of IBM SP2 *IBM Systems Journal*, Vol. 34, No. 2, (1995).
- [18] J. Niwa, T. Inagaki, T. Matsumoto, and K. Hiraki: Compiling Techniques on Asymmetric Distributed Shared Memory (in Japanese). *IPS Japan SIG Reports*, Vol.97 No.75, HPC-67-21, pp.121-126 (August 1997).
- [19] J. Niwa, T. Inagaki, T. Matsumoto, and K. Hiraki: Performance Evaluation of Compiling Techniques on Asymmetric Distributed Shared Memory (in Japanese). *IPS Japan SIG Reports*, Vol.97 No.102, ARC-126-16, pp.91-96 (August 1997).
- [20] P. Keleher, A. L. Cox, and W. Zwaenepoel: Lazy Consistency for Software Distributed Shared Memory. *Proc. the 19th Int. Symp. on Computer Architecture*, pp.13-21 (May 1992).
- [21] L. Iftode, C. Dubnicki, E. W. Felton, and K. Li: Improving Release-Consistent Shared Virtual Memory using Automatic Update. *Proc. the 2nd Int. Symp. on High-Performance Computer Architecture*, pp.14-25 (February 1996).
- [22] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta: The SPLASH-2 Programs: Characterization and Methodological Considerations. *Proc. the 22nd Int. Symp. on Computer Architecture*, pp.24-36 (June 1995).