International Journal of Parallel Programming, Vol. 28, No. 2, 2000

A Distributed Shared-Memory System on a Workstation Cluster Using Fast Serial Links

Hironori Nakajo,[1] Akihiro Ichikawa,[2] and Yukio Kaneda[2]

Received April 15, 1998; revised July 15, 1999

In this paper, a fast serial link, Serial Transparent Asynchronous First-in First-out Link (STAFF-Link), is introduced. Using such links, we construct a parallel processing system based on a workstation cluster. The workstation cluster implements a distributed sharedmemory mechanism for inter-process communication using a software controlled cache using a STAFF-Link router board. The board has a chained multicast capability with which we have implemented efficient invalidation protocol based on Eager Release Consistency (ERC) model in the DSM system. Performance results on several application programs from the SPLASH2 benchmark suites have been measured.

KEY WORDS: Distributed shared-memory; workstation cluster; STAFF-Link.

1. INTRODUCTION

Recent improvements of integrated circuit and network technologies allow anumber of workstations to be connected into a multiple processor system that can of achieve the high performance of parallel/vector computing systems.

There are two kinds of multiprocessors: the message-passing type and the shared-memory type. The communication can be either through explicit messages sent directly from one processor to another or through access to shared-memory. Message-passing systems present a programmer with a set

[1] Division of Computer Information and Communication Sciences, Tokyo University of Agriculture and Technology.
[2] Department of Computer and Systems Engineering, Kobe University.

of separate computers that communicate only by sending explicit messages. In shared-memory systems, memory is accessible by all processors and communication is accomplished through shared variables. A shared-memory machine can also be used as a message-passing machine with messages stored in shared-memory buffers.

Message-passing systems require explicit data movement via message exchange between processors. This makes it very difficult to write parallel programs with dynamic or irregular communication patterns. Shared-memory systems avoid this difficulty and have therefore achieved commercial success with tightly-coupled multiprocessor systems such as SUN SPARCcenter1000 or SGI Origin2000. Thus, even though message-passing libraries for parallel programming, e.g., PVM(1) or MPI(2) are now available for workstations clusters,(3) shared-memory communication support for multiple workstations is important.

This research has concentrated on problems for hardware and software implementation of a shared-memory programming system on workstation clusters.

The shared-memory system implementation on workstation clusters faces two major problems. First, there is no physically shared memory in a workstation cluster. Second, a local area network connecting the workstations usually does not have a high enough bandwidth. To address the first problem, virtual shared memory systems using message passing have been proposed, e.g., IVY,(4) TredMark,(5) Quarks,(6) and Tempest.(7) Such systems use caching techniques to minimize remote accesses and to conserve the low bandwidth of the LAN.

In a previous study,(8) we have implemented a distributed shared memory (DSM) system based on a 10-baseT local area network. The DSM system did not achieve expected performance on all applications because of the low bandwidth and high contention of the LAN. This experience showed that exible and sophisticated network facilities are required to implement an efficient shared-memory parallel computing environment. The work described here addresses the problem by implementing a workstation cluster using point-to-point fast serial links called a Serial Transparent Asynchronous First-in First-out link (STAFF-Link).

For the purpose of efficient routing between nodes in a cluster, we have implemented a STAFF-Link router board. The system has an efficient multicast capability called chained multicast in which asingle packet travels around specified nodes. The multicast is used for an efficient invalidation and implementing an Eager Release Consistency (ERC) model as described in a later section.

The remaining sections of this paper introduce the hardware configuration of a STAFF-Link, a router board and the parallel computing environment of a workstation cluster connected with STAFF-Link router boards. We introduce parallel programming environment of DSM with write invalidate protocol based on an ERC model in our cluster system. System performance has been measured using several application programs and the results are discussed.

2. THE STAFF-LINK

The STAFF-Link was designed to connect multiple I/O units and processing elements and used to configure the scalable I/O subsystem of JUMP-1(Japan University Massively Parallel computer).(9) The STAFF-Link interface design is general purpose which allowed us to use the fast serial links in constructing a high performance workstation cluster.

2.1. Transfer Speed and Distance of Local and Wide Area Networks

Processing elements of a parallel computing system are usually connected via a fast backplane bus or via cables. High-speed data communication using cables with many signal lines is difficult to scale due to noise, clocking problems and packaging difficulties. Furthermore, if the transfer width or the number of router ports are increased the required connector area increases proportionately.

Given the tradeoff between physical and spatial restrictions and transfer speeds, we have implemented a communication link which can be accessed as FIFO memory located between the ends of the link. This link is referred to STAFF-Link *(Serial Transparent Asynchronous First In First Out Link)*.

High-speed serial communication LSI chips are now widely available for wide area networks such as B-ISDN and ATM. The reliability of communication using these LSI circuits has improved, and, combined with improvements in printed circuit board assembly technology for mounting such LSIs, makes serial communication between workstations very promising.

It can be extrapolated that a bundle of multiple STAFF-Links can achieve transfer speeds from tens of Mbps to a few Gbps and can cover a distance from a few meters to a few hundred meters. This makes STAFF-Links suitable for workstation cluster interconnect.

2.2. STAFF-Link Organization

Serial communication can be divided into the following five phases:

1. Data write.

2. Parallel-serial conversion.
3. Data transfer.
4. Serial-parallel conversion.
5. Data read.

A conventional serial communication interface delay due to parallel/serial data conversion limits transfer speeds. When using a STAFFLink, phases 2–4 are handled by a high-speed serial communication LSI. Buffers provided by the circuit at both the send and the receive ends of the link allow the five phases above to be overlapped and thus raise the throughput.

Figure 1 shows the STAFF-Link organization. A communication block consists of a high-speed bidirectional serial communication LSI (TAXI chip[(10)]), two FIFO memories (send and receive), and an asynchronous communication controller (X ow control) with handshaking to prevent FIFO over ow. Currently a single STAFF-Link can handle transfer rates of up to 140 Mbps.

Connection of two communication blocks by Category 5 twisted-pair cables results in a virtual bidirectional FIFO at each end, and in turn effectively hides the physical communication distance between nodes to ensure a transparent communication path. The communication controller provides asynchronous communication control simultaneously with Xon/Xoff control. Specifically, when the receiving FIFO at the destination is more than

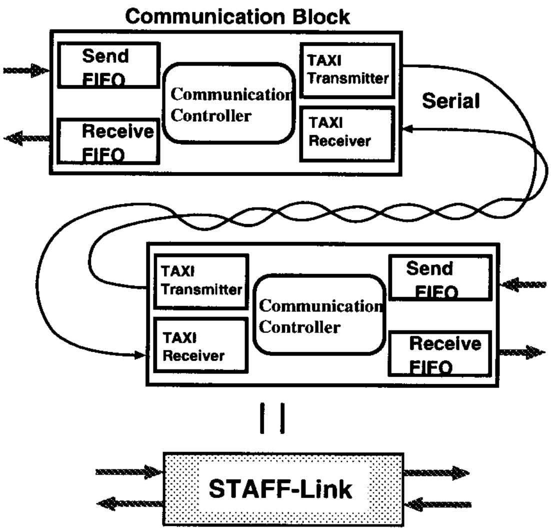

Fig. 1. Serial Transparent Asynchronous First-in First-out Link (STAFF-Link).

half full, the receiving side sends an Xoff message requesting an interruption of transmission. When the receiving FIFO is able to receive again, the Xon message is sent requesting resumption of transmission. This control is performed automatically by the two communication controllers. At the transmission side, data is written to the sending FIFO until it is full, and at the receiving side the data can be read from the FIFO until it gets empty. The use of multiple STAFF-Links allows a workstation cluster system to have a high communication bandwidth.

3. A DSM SYSTEM ON A WORKSTATION CLUSTER

3.1. Design Concept

The general organization of our DSM system is shown in Fig. 2. Instead of a single specific server node, each node participates in managing the shared-memory space. Thus the management load for the shared-space access is distributed over the entire system.

As described earlier, caching is required in workstation clusters implementing shared-memory. Thus our system supports a cache mechanism with a write invalidate based on a relaxed consistency model. The shared-memory space of our system consists of pages which are allocated in each local memory.

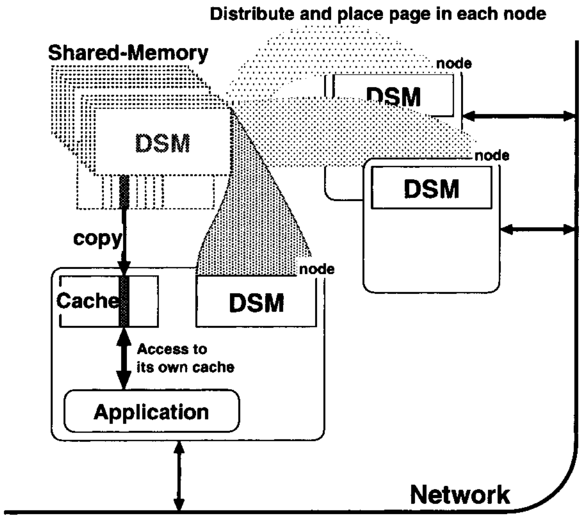

Fig. 2. The concept of our DSM system.

3.2. Consistency Control

Many consistency models have been proposed such as Release Consistency Model for better performance in shared-memory systems. In a release consistency model, there are some variants such as an Eager Release Consistency[11] model or a Lazy Release Consistency[12] model.

In a distributed shared-memory system, a vast shared space is divided into some fixed size of pages and allocated and managed in each node which configures an entire system. Home memory is defined as a collection of pages distributed and allocated in each node. Generally consistency is controlled in a unit of page in a software controlled shared-memory system.

If a size of a page is set to a small size, the number of pages in a shared space becomes large, and an amount of attributes of pages increases. Therefore, a size of page must be considered for implementation of a shared-memory system. As a result, some variables which are used in different processes or routines tend to be allocated in a same page.

In an Eager Release Consistency model, each collection of shared variables which are accessed by an application program during a specified period is managed for consistency control. Namely, a single synchronization variable is defined for each collection of shared variables, and coherence of shared variables are maintained while an only node which acquires a right of accesses to the synchronization variables accesses to a collection of the shared variables.

In a same way, a Lazy Release Consistency model adopts a right of accesses to a synchronization variable.

An Eager Release Consistency model conducts consistency control of shared variables in releasing a right of accesses to a synchronization variable. On the other hand, in a Lazy Release Consistency model, consistency control is postponed until a right of accesses to a synchronization variable is acquired in a next session of accessing to shared variables. Therefore, amount of communication in a Lazy Release Consistency model is smaller than in a Eager Release Consistency model, however, consistency control is complicated and information for consistency control must be kept in a long time. From the before, we have implemented a distributed shared-memory system base on an Eager Release Consistency model in our system.

3.3. Software Organization in a Workstation

The software organization in each node is shown in Fig. 3. A **manager process** maintains the **DSM** using workstation memory pages. A directory is maintained and updated according to accesses or invalidation messages

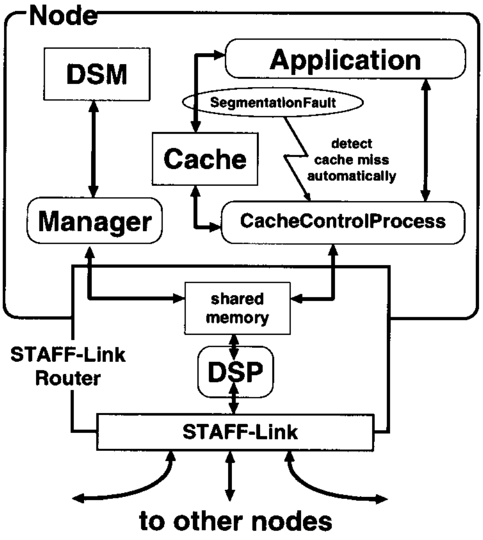

Fig. 3. Structure of one node.

from other nodes. The cache memory uses part of main memory through which an application program accesses a shared-memory space.

A **Cache Control process** is linked to an application program and executed as a thread. When the process detects a miss access in the application program, it requests a desired page from the **Manager process** and also locks the requested page. Moreover, the **Cache Control process** performs invalidations according to requests from other nodes in order to keep cache memory coherent.

3.4. Detecting Cache Misses

In a tightly coupled bus-based multiprocessor system, data accesses from processing elements are always snooped by hardware. In a workstation cluster environment, the operating system cannot keep watch over all accesses and a different mechanism is required. Our system detects miss accesses to DSM cache via UNIX *Segmentation fault* (SIGSEGV), a mechanism commonly used in software-controlled cache systems. As a result, there is no need to check the state of DSM cache on each access and therefore a cache hit access does not incur any overhead.

3.5. Efficient Invalidation and Synchronization by Chained Multicast

A write access to a page in DSM causes copies of the page in other nodes to be invalidated in order to keep coherence. A bus network allows a broadcast of an invalidation message at the same time. But bus-based systems lack scalability due to a limited bandwidth. In contrast, a point-to-point network cannot support efficient broadcast even though it has sufficient bandwidth to support a larger number of workstations.

It has also been reported that a number of copies of a shared page is limited to two or three, on average, in most application programs for invalidation-based cache coherence protocols.(13) Thus invalidation using a broadcast message does not offer a great advantage over multiple "almost-simultaneous" requests.

Therefore, our system has adopted a chained multicast as an invalidation mechanism. In this case a single message travels to just the required nodes which are specified in a header of the message. The chained multicast invalidation message includes the address of the node which issues the invalidation request and the message is finally sent to the original node as an acknowledge message. Moreover we have implemented an Eager Release Consistency model with a chained multicast mechanism for efficient synchronization.

The basic concept of chained multicast mechanism was introduced by Nakajo *et al.*(14) A similar mechanism is used in myrinet.(15)

3.6. Hardware Organization of STAFF-Link Router Board

Software-controlled routing in a workstation cluster may cause a significant overhead and may degrade system performance by consuming CPU cycles. A hardware router, independent oftheworkstation operating system, is therefore desirable. For this reason we have designed and implemented a STAFF-Link router board shown in Fig. 4.

The I/O network router board has the following characteristics:

- Board: SBus double-height
- Routing controller: DSP (TMS320C40)
- Program memory: 512 KWord (2 MB)
- Data memory: 512 KWord (2 MB)

Data transfer performance using the router board is shown in Table I which shows detailed communication time. The results helped us find that a bottleneck exists in accessing main memory of the workstation. This limits the data transfer speed of a STAFF-Link to a maximum of 64 Mbps.

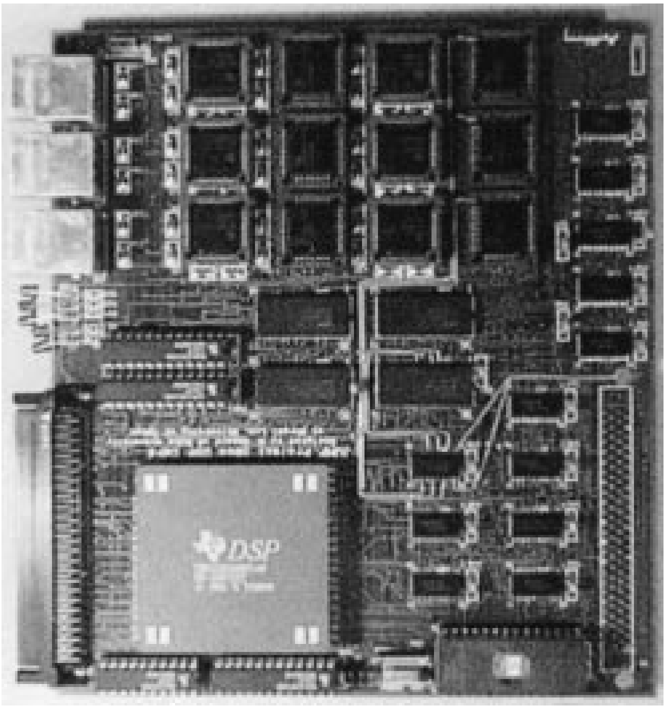

Fig. 4. STAFF-Link router board.

Table I. Communication Time of a STAFF-Link Router Board

Process	Time or through put
Access start up on communication buffer	45 μS
Write to communication buffer	128 Mbps
Communication buffer $\rightarrow$ FIFO	200 Mbps
Communication start up	320 nS
Send FIFO $\rightarrow$ receive FIFO	140 Mbps
FIFO $\rightarrow$ communication buffer	200 Mbps
Read from communication buffer	64 Mbps
Routing (per 1 node)	1 μS

Table II. The Workstation Node Configuration

Workstation	Sun SPARCstation-20(SS20)
CPU	SuperSPARC(60 MHz) × 2
OS	Solaris 2.5.1(SunOS 5.5.1)
Memory	64 MB

4. PERFORMANCE EVALUATION OF THE DSM SYSTEM

An experiment workstation cluster system consists of four Sun Microsystems SPARCstation20 (SS20) workstations. A specification of the workstation used in the experiments is described in Table II.

4.1. Cache Miss Penalty

Figures 5 and 6 show read miss and write miss penalty of shared-memory accessing respectively. From Figs. 5 and 6, sending and receiving a page sized 4 KB spends about 3 ms. It spends about 700 μs to evacuating current cache in order for generating update information.

4.2. Performance Evaluation with Parallel Processing Applications

4.2.1. Matrix Multiplication

We have measured processing time of matrix multiplication as a first parallel processing application.

Figure 7 shows a result of matrix multiplication sized 128 × 128. The vertical and horizontal lines in the figure are processing time and the

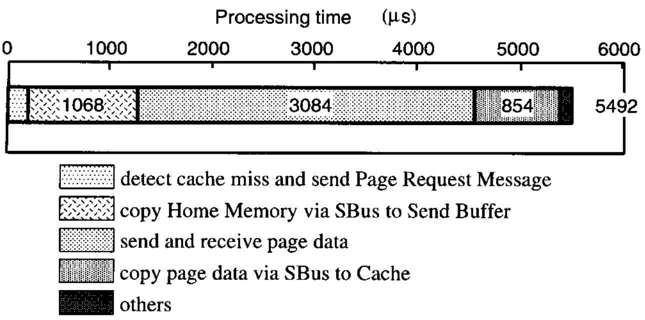

Fig. 5. Miss penalty of Read Miss.

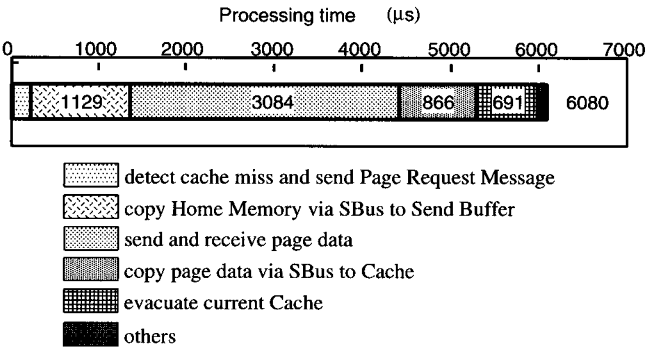

Fig. 6. Miss penalty of Write Miss.

number of nodes used for processing respectively. Dotted line shows a processing time with the single SPARCstation20.

Computation, Release, Barrier and Initialization mean computation time, processing time for releasing a right for accessing to shared variables, waiting time for barrier synchronization and time for initialization of data respectively.

From the result of Fig. 7, the system shows good performance in the increasing number of nodes because grains of computation of matrix multiplication are quite large, thus the accessing time to shared-memory is short.

4.2.2. Processing Time of FFT

As the next application, we have adopted an FFT program from SPLASH2[16] benchmark suites.

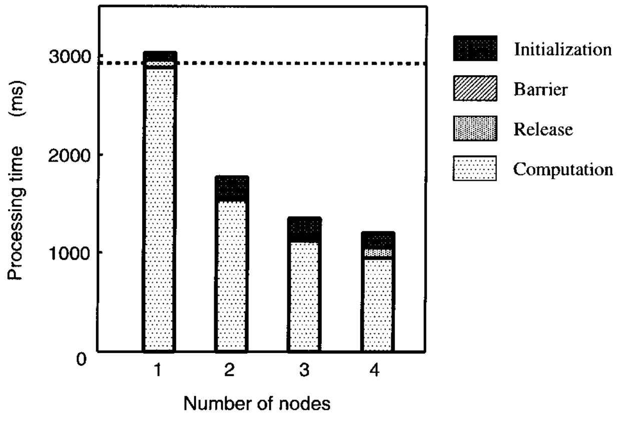

Fig. 7. Processing time of matrix multiplication sized 128×128.

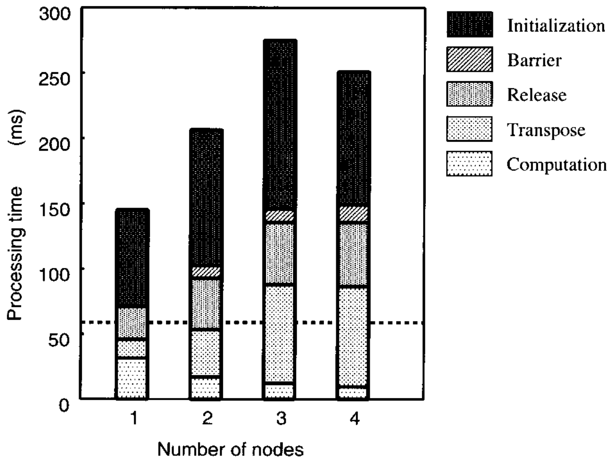

Fig. 8. Processing time of FFT with 1024 data points.

Figures 8 and 9 show the results of FFT with 1024 and 65536 data points respectively.

As in the matrix multiplication, Computation, Release, Barrier and Initialization mean computation time, processing time for releasing a right for accessing to shared variables, waiting time for barrier synchronization and time for initialization of data respectively. Transpose means a time for writing results to shared-memory.

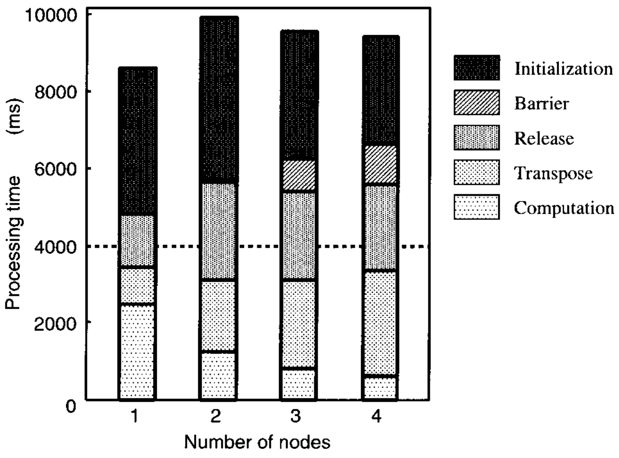

Fig. 9. Processing time of FFT with 65536 data points.

The results of Figs. 8 and 9 shows that the system performs well with focus on only computation time. However, total performance is worse than with a single SPARCstation20, because initialization of data spends about 40–50% of whole processing time. Moreover, an FFT program consists of fine grained processing and needs many synchronization points during computation, thus the costs of writing results and releasing a right of accessing to synchronization becomes larger in the increasing number of nodes.

As Fig. 9 shows, processing times are almost same with the number of nodes because the processing of initialization is distributed to nodes with increasing number of nodes and initialization time gets smaller.

4.2.3. Processing Time of LU Decomposition

In this subsection, results of LU decomposition are shown. The results of LU decomposition sized 128×128 and 512×512 are respectively shown in Figs. 10 and 11.

From the results of Figs. 10 and 11, computation time is getting smaller in the increasing number of nodes.

However the system cannot perform sufficient speed up with the increasing number of nodes because, as in FFT, an LU decomposition program has many synchronization points and there needs releasing a right of accessing to shared variables in each synchronization point.

After a process releases a right of accessing to shared variables, state of cache is changed from **READ/WRITE** to **READONLY**. Thus when a

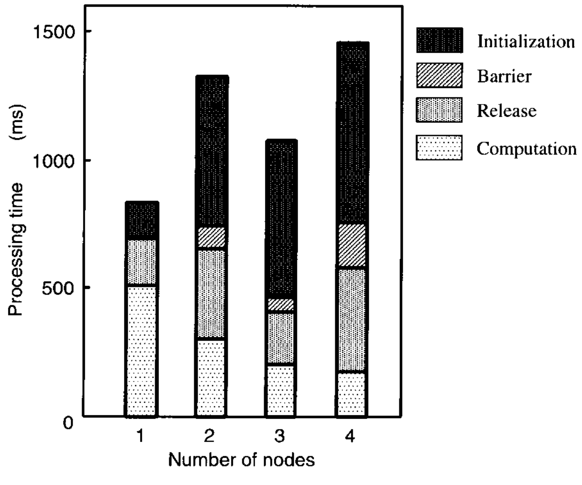

Fig. 10. Processing time of LU decomposition with 128×128 Matrix.

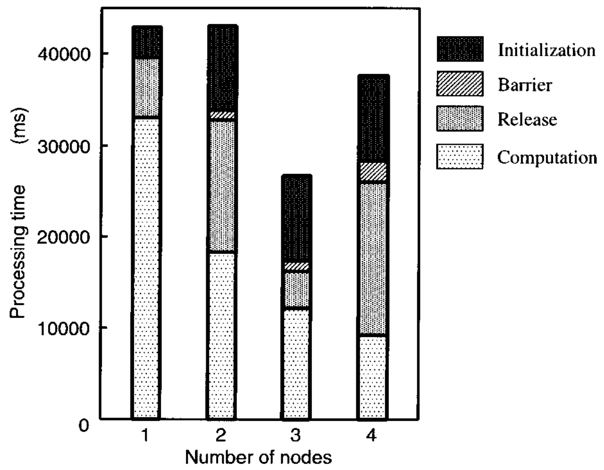

Fig. 11. Processing time of LU decomposition with 512×512 matrix.

process writes to the cache, consistency control is performed based on **Clean Write** which isa write access to cache in a state of clean. Consequently the number of **Clean Write** increases and the cost for consistency control is significantly large in LU decomposition.

In both results of Figs. 10 and 11, a releasing time with 3 nodes is shorter than other cases.

We have investigated the number of Update Request Message and Update Message in processing LU decomposition with a matrix sized 512×512 as shown in Table III.

An Update Request Message is a request to **manager process** which sends an Update Message in order for cache to be updated for implementing an Eager Release Consistency model.

From Table III, since the number of Update Messages is significantly smaller than the one of Update Request Messages, the releasing time with three nodes is short. There are less nodes which holds targeted pages of Update Request Messages, because distribution of shared data is well performed in the case of 3 nodes.

Table III. Number of Messages

Message	1	2	3	4
Update request message	6633	12466	12979	13492
Update message	0	6088	2593	16502

5. CONCLUSIONS

This paper described a workstation cluster system with a distributed shared memory. It uses a fast serial link called STAFF-Link to build an interconnection network.

In our system, chained multicast has been adopted for an efficient multicast in a limited bandwidth network. Chained multicast is used not only for efficient invalidation but also efficient synchronization based on an Eager Release Consistency model.

From the results of experiments, the system performs well in computation in the increasing number of nodes, however, there exists overhead of page transferring or consistency control. Compiler supports will be needed for better performance in the future.

ACKNOWLEDGMENTS

We would like to thank Prof. Hidehiko Tanaka of University of Tokyo and Prof. Shinji Tomita of Kyoto University for their supports in implementing a STAFF-Link and the router board. This work was supported in part by the Japan Ministry of Education, Science and Culture under Grant in Aid for Scientific Research 042235103 entitled "Research on Massively Parallel Hardware Architecture" and 06508001 entitled "Development and Testbed of Massively Parallel Computer Prototype."

REFERENCES

1. V. S. Sunderam, PVM: A Framework for Parallel Distributed Computing, *Concurrency: Practice and Experience* **2**(4):315–339 (1990).
2. Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, *Int'l J. Supercomputer Applications and High Performance Computing* **8**(3/4):159–416 (1994).
3. Craig C. Douglas, Timothy G. Mattson, and Martin H. Schultz, Parallel Programming Systems for Workstation Clusters, Technical Report TR-975, Yale University Department of Computer Science Research (1993).
4. K. Li and P. Hudak, Memory Coherency in Shared Virtual Memory Systems, *ACM Trans. Comput. Syst.* **7**(4):321–359 (1989).
5. Pete Keleher, Sandhya Dwarkadas, Alan L. Cox, and Willy Zwaenepoel, TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Rice COMP TR93214 (1993).
6. Dilip Khandekar, Quarks: Portable Distributed SharedMemory on UNIX, *quarks/doc/tech-report.ps* including *ftp://jaguar.cs.utah.edu/pub/dsm/Quarks.tar.Z* (1995).
7. Mark D. Hill, James R. Larus, and David A. Wook, Tempest: A Substrate for Portable Parallel Programs, *Proc. of COMPCON'95*, pp. 327–332 (1995).
8. H. Nakajo, K. Kuramae, Y. Kaneda, and S. Maekawa, The Implementation and Evaluation of Software Distributed SharedMemory (DSM) for Workstation Clusters (in Japanese), *Trans. IPS Japan* **36**(7):1719–1728 (1995).

9. H. Nakajo, S. Ohtani, T. Matsumoto, M. Kohata, K. Hiraki, and Y. Kaneda, An I/O Network Architecture of the Distributed Shared-Memory Massively Parallel Computer JUMP-1, *Proc. of 11th Int'l. Conf. on Supercomputing (ICS97)* (1997) (to appear).
10. Advanced Micro Devices, Inc. Am7968/Am7969-175 TAXI-175 Transmitter/Receiver Data Sheet and Technical Manual (1992).
11. J. B. Carter, J. K. Bennet, and W. Zwaenepoel, Implementation and Performance of Munin, *Proc. 13th ACM Symp. Operat. Syst. Principles*, pp. 152–164 (1991).
12. Pete Keleher, Alan L. Cox, and Willy Zwaenepoel, Lazy Release Consistency for Software Distributed Shared Memory, *ACM SIGARCH Computer Architecture News* **20**(2):13–21 (1992).
13. Daniel E. Lenoski and Wolf-Dietrich Weber, *Scalable Shared-Memory Multiprocessing*, Morgan Kaufmann Publishers (1995).
14. Hironori Nakajo, Takeshi Yoshinaga, Koichi Wada, and Yukio Kaneda, Ring-Connected Parallel Computer KORP-Coherence Protocol for Distributed Shared-Memory, *Proc. Int'l. Conf. Parallel and Distrib. Syst. ICPADS'92*, pp. 504–511 (1992).
15. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su, Myrinet: A Gigabitper Second Local Area Network, *IEEE Micro* **15**(1):29–36 (1995).
16. Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, SPLASH-2 Programs: Characterization and Methodological Considerations, *Proc. 22nd Int'l. Symp. Computer Architecture*, pp. 24–36 (1995).