1998 International Computer Symposium
Workshop on Computer Architecture
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

Effective Prefetching and Replacement Policies in the Scalable Clustering-Based Multiprocessor System Design

Der-Lin Pean, Hsuan-Woei Huang, Jia-Rong Wu and Cheng Chen

Department of Computer Science and Information Engineering,
1001 Ta Hsueh Road, Hsinchu, Taiwan, 30050, Republic of China
Email:{dlpean, cchen}@csie.nctu.edu.tw

Abstract

Recently, advancement in VLSI and packaging technologies demonstrates attractiveness in building scalable parallel systems using clustering-based architecture by exploiting communication locality. In this paper, we propose effective prefetching and replacement policies for our clustering-based multiprocessor system to enhance its total performance substantially. Our inter-clustering prefetching mechanism not only prefetches data for the single processor locality but also for the communication locality of processors in the same cluster by using inter-clustering caches. It also enhances conventional prefetching schemes by prefetching more data with less traffic overhead. Replacement policy proposed decreases replacement overhead effectively in the intra-clustering bus-based system. Simulation results show that prefetching and replacement schemes presented improve clustering-based system performance up to 33% and 25% respectively for the SPLASH [1] benchmark suites.

1. Introduction

Shared memory multiprocessor systems have become one of the most important design trends in computer system architectures recently [1, 2]. Bus-based and directory-based multiprocessors have the limitation that they do not scale well to large number of processors. With advancements in VLSI and packaging technologies it has become cost effective to integrate multiprocessing elements into a board module. A modular and hierarchical approach to build large systems with good scalability will be one of the most effective future trends to develop high performance multiprocessor systems [2, 3]. Thus, how to boost the total performance of clustering-based multiprocessor system is often an interesting and important design topic so far [2].

In this paper, we propose the clustering-based multiprocessor architecture with good scalability. The system uses the SCI protocol as inter-clustering cache coherence protocol and the Berkeley protocol [12] as our intra-clustering cache coherence protocol. The inter-clustering caches are also developed to exploit locality of processors in the same clustering node. Previous research has shown that it is effective to hide memory access latencies by prefetching the data before they are really used [5]. However, we find that prefetching mechanism in clustering multiprocessor systems encounters much overhead to fetch data from several levels of interconnection networks. We propose inter-clustering prefetching scheme to gain communication locality of processors in the same clustering node and enhance conventional prefetching mechanisms with less traffic overhead. We also propose an effective replacement policy to reduce large replacement overhead of the intra-clustering cache coherence protocol. By avoiding the state transition from Dirty-Shared to Dirty-Exclusive state, the acquiring and acknowledgement messages for maintaining cache coherence can be greatly reduced. Hence, the total system performance benefits reach about 25% with our effective replacement policy and 33% with the inter-clustering prefetching scheme.

The rest of the paper is organized as follows. In Section 2, we give an overview of our clustering-based multiprocessor architecture. In section 3, we present our inter-

clustering prefetching scheme and effective replacement policy is also described in section 4. In section 5, our simulation environment is described first and then several performance evaluation gains about those schemes are given and explained in some detail. Finally, we will summarize the conclusions of the paper in section 6.

2. Overview of Our Clustering-based Multiprocessor Architecture

We start in the following to present the architecture we use as a base for the implementation and performance evaluations of our effective schemes. The architecture we consider here is a clustering-based distributed shared-memory multiprocessor system, as depicted in Fig. 1, which is a CC-NUMA clustering architecture consists of many clustering nodes interconnected by the K-ary, n-Cube network. Each clustering node contains a local shared-memory area, an inter-clustering cache, a processor environment, and a local bus. The inter-clustering cache proposed contains data that is usually used in the inner clustering processors.

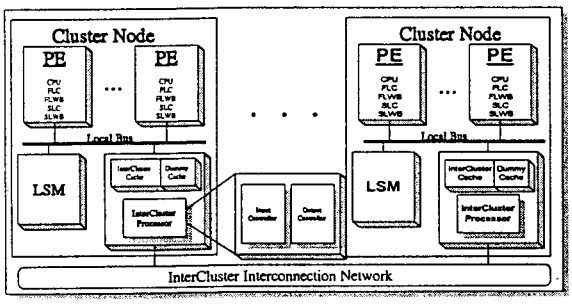

Fig. 1. The overall architecture of our clustering multiprocessor system

Each processor environment includes a two-level cache hierarchy and associated with write buffers, as displayed in Fig. 2. The cache hierarchy is interfaced to the local portion of the shared memory and the inter-clustering cache by a local bus according to Fig.1. In order to support release memory consistency models we implemented a lockup-free second level cache [10]. In this architecture, we adopt the SCI cache coherence protocol [7] to implement our linked-based cache coherence protocols. The clustering system architecture presented above exploits some advantages: Resources are shared, packaging technologies are exploited, and processors within a cluster can share data more effectively.

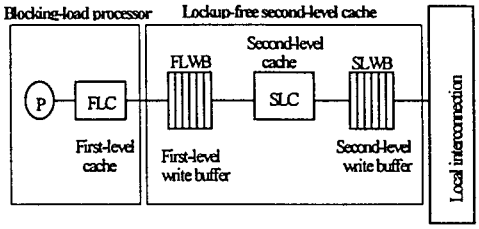

Fig. 2. The processor environment

The first-level cache (FLC) is a write-through on chip cache whereas the second-level cache (SLC) is a copy-back cache. Both caches are direct-mapped with the same line size in both caches and full inclusion property is supported; if a block is present in the FLC it is also present in the SLC. All coherence actions associated with the system-level cache coherence protocol are handled by the SLC, inter-clustering cache and memory controller.

In order to reduce the substantial network traffic overhead incurs by deep memory hierarchy of the clustering multiprocessor system, we propose an effective replacement policy to reduce intra-clustering network traffic and an inter-clustering prefetching mechanism to reduce inter-clustering network traffic. With the inter-clustering prefetching scheme, not only the read miss rate but also the network traffic of the intra-clustering processors can be greatly reduced. In the following sections, we will describe these mechanisms in some detail.

3. Effective Prefetching Scheme

It is necessary to effectively explore communication locality [11, 12] in clustering multiprocessor systems for the purpose of upgrading the system performance. However, while there is no locality property in the data set, it is impossible to reuse these data since they may often be replaced during a short period [2, 13, 14]. Data prefetching is an effective way to hide memory access latencies [8, 9, 10]. With using of prefetching techniques in clustering multiprocessor systems, we may overcome such long data access latencies. It is efficient to reduce long access laten-

cies by using simple hardware-based sequential prefetching schemes such as fixed and adaptive prefetching schemes [9] in those non-clustering multiprocessor systems. However, it is much complex to use prefetching mechanism when considering clustering multiprocessor systems that need to access data through intra- and inter-clustering interconnection networks. The traffic overhead becomes very serious when conventional prefetching techniques are implemented in the clustering multiprocessor systems. Therefore, it is necessary to develop new effective prefetching schemes in our clustering multiprocessor system architectures.

Therefore, the conventional hardware prefetching mechanism enhances the performance of non-clustering multiprocessor systems. However, due to large traffic overhead of the clustering multiprocessor system, hardware prefetching mechanism must be improved to further increase the performance of the clustering multiprocessor system. We propose our enhanced method in the following.

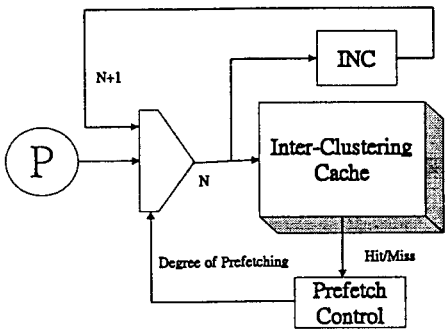

Fig. 3. The inter-clustering prefetching mechanism

3.1. Inter-clustering Prefetching Technique

There are more levels of memory hierarchy in clustering than in non-clustering multiprocessor systems. Thus, a normal prefetching access must traverse more memory levels to read or write data while miss occurs. In addition to original memory accesses, there are multiple times of such accesses while prefetching is used. On the other hand, the traffic in every level of clustering multiprocessor systems also increases several times as the number of prefetching data is increased. If the prefetching mechanism is built in the higher level of memory hierarchy, the prefetching accesses need to be delay at every level of memory hierarchy because the traffic is increased in every memory hierarchical level.

Thus, we implement the prefetching mechanism in the inter-clustering cache rather than the SLC of our clustering multiprocessor system as shown in Fig. 3. The inter-clustering prefetching scheme extracts some performance gains from decreasing of network contention in the higher levels of the memory hierarchy. It also increases the locality of intra-clustering data because it brings in the data that may be used soon in this clustering node.

On the other hand, the access counts in the conventional hardware prefetching scheme bring in much traffic overhead as described above. Another approach to improve it is to send only one request at every read miss request. As shown in Fig.4, while reading for block number n misses, it issues only the read miss request for block number n. When other SLCs and inter-clustering caches receive the request for block number n, they search for not only block n, but also block n+1. After searching for these data blocks, they reply for these blocks.

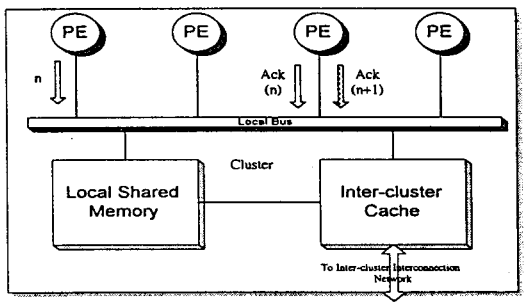

Fig. 4. The traffic requests of the inter-clustering prefetching scheme

This inter-clustering prefetching technique causes less local bus requests because it issues less read miss requests and improves our clustering system. Due to the limited size of the cache, replacement also incurs serious traffic overhead in the clustering multiprocessor system. We will propose another effective replacement method to reduce such serious traffic problems further.

4. Effective Block Replacement Scheme

Because the number of processors in the same clustering node would not be large, the snoopy bus protocol is suitable for the intra-clustering cache coherence protocol [13, 14, 15]. Thus, we implement the Berkeley protocol [12] as our intra-clustering cache coherence protocol. We will first describe the protocol briefly in the following.

The Berkeley protocol uses the following states: Invalid, Clean-Shared (possibly shared and not modified), Dirty-Shared (possibly shared and modified), and Dirty-Exclusive (no other copies in caches and modified) as shown in Fig. 5. A block in either state Dirty-Shared or Dirty-Exclusive must be written back to main memory if it is selected for replacement. A block in state Dirty-Exclusive can be in only one cache. A block can be in state Dirty-Shared in only one cache, but it might also be present in state Clean-Shared in other caches. It uses the idea of ownership. If a block is not owned by any cache, memory is the owner. The consistency solution is given in the following:

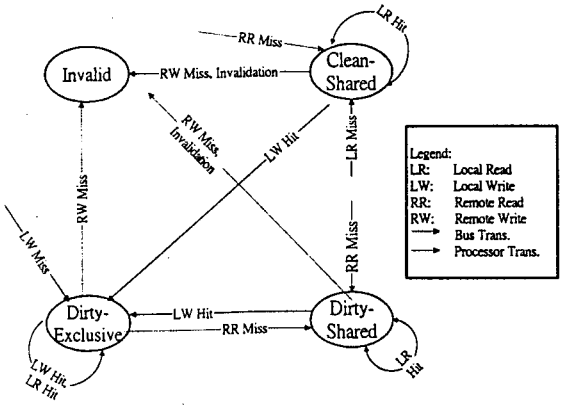

Fig. 5. The transition diagram of the Berkeley protocol

(1) Read miss. If the block is in state Dirty-Shared or Dirty-Exclusive, the cache with that copy must supply the block contents directly to the other cache and set its local state to Dirty-Shared. If the block is in any other state or not cached, it is loaded from main memory. In any case, the state of the block in the requesting cache is set to Clean-Shared . Note that the block always comes directly from its owner.

(2) Write hit. If the block is already in the Dirty-Exclusive state, the write proceeds with no delay. If the block is in Clean-Shared or Dirty-Shared state, an invalidation signal must be sent on the bus before the write is allowed to proceed. All other caches invalidate their copies upon matching the block address, and the local state is changed to Dirty-Exclusive in the originating cache.

(3) Write miss. Like a read miss, the block comes directly from the owner. All other caches with copies change the state to Invalid and the block in the requesting cache is loaded in state Dirty-Exclusive.

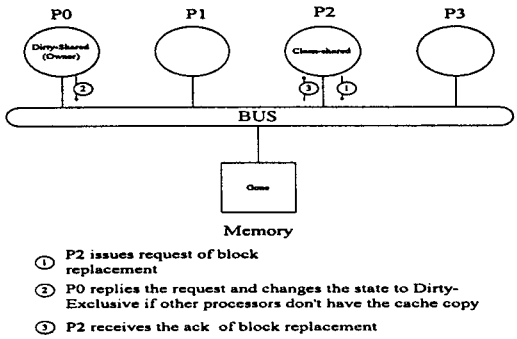

Fig. 6. Block replacement of Clean-Shared state in the Berkeley protocol

For the replacement of a cache line that is in the Clean-Shared state, some actions must be done in advance. As shown in Fig. 6, the states of the specific block in P0、P2 and memory are Dirty-Shared, Clean-Shared and Gone respectively. The cache in P2 first issues a request of block replacement. Once P0 receives the request and other processors do not have the cache copy, the cache of P0 changes the state to Dirty-Exclusive. Finally, P2 receives acknowledgements of block replacement and replaces the cache block. If the cache block is in the Dirty-Exclusive state, the cache block in the dirty-Exclusive state can be written directly. If some block of Clean-Shared state is replaced by other blocks in the Berkeley protocol, the cache controller

has to issue the request of block replacement. Because we use Berkeley protocol as our intra-clustering protocol, the replacement actions of Clean-Shared block are the same as those we have discussed above.

However, if the total number of write requests from Dirty-Exclusive state which comes from Dirty-Shared state is less than the number of block replacement from Clean-Shared state, we don't need to change the state from Dirty-Shared to Dirty-Exclusive. Table I displays the total number of write requests from different states. A Dirty-Exclusive state is divided into a pure Dirty-Exclusive and a impure Dirty-Exclusive state. The state is set to either impure Dirty-Exclusive state if the Dirty-Exclusive state comes from Dirty-Shared state or pure Dirty-Exclusive state otherwise. As illustrated in table I, the number of changes from Dirty-Shared states to Dirty-Exclusive states are much more than the number of write requests from the Dirty-Exclusive states. Hence, effective replacement mechanism avoids the overhead of changing Dirty-Shared to Dirty-Exclusive states.

Benchmark	Write from pure Dirty-Exclusive	Write from impure Dirty-Exclu-sive	Write from Dirty-Shared	Delete from Clean-Shared
Mp3d	2.8(M)	0	13760	676
FFT	5.6(M)	624	13358	118906
Ocean	14.2(M)	42432	603722	107581
PTHOR	643344	108	12900	209274
Water	2.5(M)	0	3093	3093

Table I Total number of write requests from different states

5. System Simulation Environment and Performance Evaluations

To study and observe the performance of a multiprocessor system, we have developed a simulation and performance evaluation environment called SEESMA [4] before. And then, we extended it to support clustering-based multiprocessor systems. With various options setting, we can use it to evaluate some design issues in non-clustering and clustering multiprocessor systems and provide a good tool for research and education purposes.

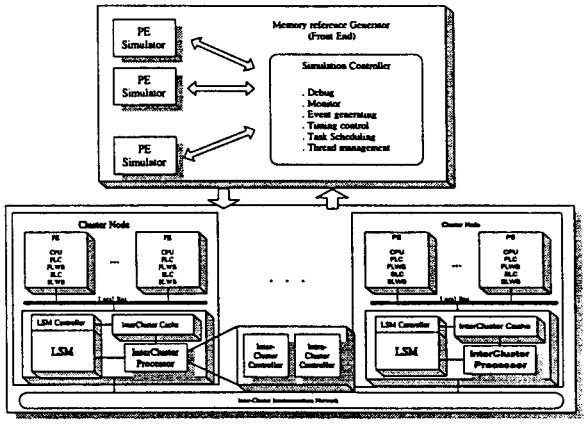

Fig. 7. The whole structure of our SEECMA

Our simulation environment is a program-driven simulator and constructed based on the MINT [11] package named SEECMA (A Simulation and Evaluation for Cluster-based Multiprocessor Architecture) [5]. The whole environment consists of two parts: the memory reference generator and the memory subsystem simulator as shown in Fig. 7. The memory reference generator is made of the simulation controller and the processor simulator. The simulation controller monitors the execution of simulation environment, its main functions include debugging, monitoring, event generation and management, task management and scheduling, timing controller and thread management. The processor simulator simulates the instruction interpretation and execution.

Parameter	Value
Number of cluster nodes	16
Number of processors in a cluster node	4
Size of FLC	32Kbytes
Size of SLC	256Kbytes
Size of inter-cluster cache	2Mbytes
Block size of FLC and SLC	32bytes
Number of entries in FLWB	16
Number of entries in SLWB	32

Table II Architecture Parameters

Benchmark	Description	Data sets
MP3D	Particle-based wind-tunnel simulator	5K particles,10 time steps
Ocean	Simulate eddy currents in an ocean basin	128 by 128 grid, tolerance 10^{-7}
FFT	Blocked 1-D FFT	64k complex points
Water	Water molecule dynamics simulation	343 molecules
Barnes	N-body gravitation simulation	8192 bodies, 6 steps
Radix	Integer Radix sort algorithm	1M integers, Radix 1024
Cholesky	Cholesky factorize a sparse matrix	tk14.0
Lu	factors a dense matrix	128*128 matrix, 32*32 blocks
Pthor	simulate a digital circuit	risc

Table III Benchmark Programs

Our memory subsystem simulator consists of the node simulator and the global interconnection simulator. The node simulator simulates the two-level cache hierarchy, doubly-linked directory cache coherence protocols, memory consistency models and the local interconnection. Our memory subsystem simulator can effectively and easily include the simulations of other interesting architectures. Another feature of memory subsystem simulator is that it supports many simulation options. By the options we can simulate and evaluate a lot of design issues of memory subsystem, including cache coherence protocols, memory consistency models, interconnection network, migratory sharing and cache hierarchy.

Before evaluating the performance, we give some reasonable assumptions about the architecture, as summarized in table II. Memory page size is 4 Kbytes and is mapped to the local memories in a round-robin fashion. We use several benchmark programs from the SPLASH and SPLASH2 suites [1] in our experimental evaluations. We list them in Table III along with the data sets being used. Regarding to MP3D, we run it with switching on the locking option. All applications are written in C using ANL macros and have been complied using cc with the optimization level 2. All statistics are collected only the parallel part of the benchmarks.

We first discuss the performance enhancement of our prefetching scheme. then the efficiency of effective replacement policy is also displayed. The total execution time is decomposed into five parts. Busy time is the execution time of instruction, and read miss time is the delay time of read miss accesses from the first level cache. Write stall time is the delay time of write access. Because the release memory consistency model can hide all write stall time, the write stall time is always zero in our system. Acquire time is the time for waiting locks, and buffer full is the cease work time of processor due to buffer full of the first level write buffer. The following figures for evaluation of execution time is all composed of these five components.

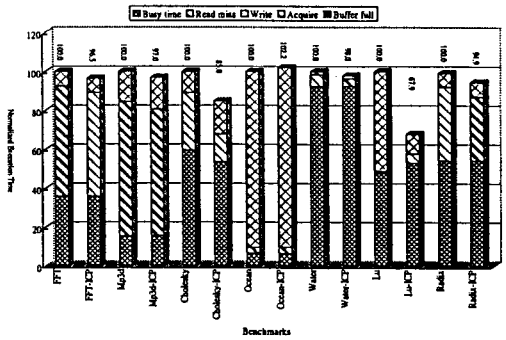

Fig. 8. The performance of our inter-clustering prefetching (ICP) scheme

5.1 Performance Evaluation of the New Prefetching Mechanism

We built our inter-clustering prefetching technique instead of conventional hardware prefetching scheme in the clustering multiprocessor system. After our evaluations, we find that the traffic overhead is reduced and the total performance is effectively improved ranging from 3% to 33% as shown in Fig. 8. In most benchmarks, the performance gain comes from reducing of read miss stall time. Since our prefetching scheme reduces read miss counts, it decreases miss ratio and reduces total access latencies. The acquire stall time of the Lu benchmark is greatly reduced due to the large traffic reduced of our inter-clustering prefetching scheme. The ocean benchmark performs worse than original condition because its false sharing characteristics. Without the serious traffic overhead and read miss penalty of conventional hardware prefetching method, the performance gain of our inter-clustering prefetching scheme is much larger.

5.2 Performance Evaluation of Effective Replace-

ment Mechanism

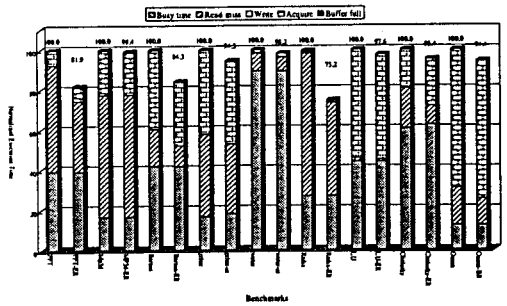

Fig. 9. The performance evaluation of the effective block replacement scheme

After evaluating the performance of effective block replacement mechanism, Fig. 9 displays that it reduces the total execution time of benchmarks substantially up to 25%. The FLC size is set to 1K and the SLC size is set to 8K respectively due to the great enhancement of our replacement policy which would be discussed later. We can find that the time reduced by the mechanism is read stall time and acquire stall time due to the efficient replacement while read miss occurs. The performance of this scheme would be better if the release memory consistency does not hide all the write miss stall time.

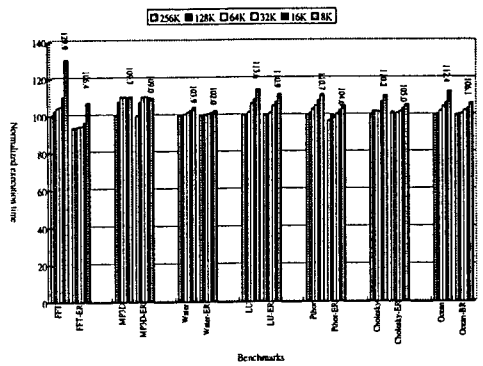

Fig. 10. The total execution time of different cache sizes with or without the effective replacement policy

In order to show that our replacement policy is cost effective, we compare the execution time of this scheme in different cache sizes as shown in Fig. 10. The total execution time increases less than 10% except radix benchmark while the scheme is implemented and the FLC size is decreased from 8K to 1K and the SLC size is decreased from 256K to 8K respectively. However, if the scheme is not implemented, the total execution time increases substantially as the cache size is decreased. Thus, our replacement policy is not only effective in reducing execution time but also efficient in reducing cost in clustering multiprocessor systems.

6. Concluding Remarks

Clustering multiprocessor system is an architecture design trend for scalable multiprocessor systems. We propose two approaches called the effective block replacement and the inter-clustering prefetching scheme to improve the performance of clustering multiprocessor systems. As the simulation results summarize, the effective block replacement policy enhance the system performance from 2% to 25% and the inter-clustering prefetching mechanism improve the execution time from 3% to 33%. They are both effective to enhance the performance of the clustering multiprocessor system not only in the intra-clustering but also in the inter-clustering system.

Sequential hardware prefetching schemes in a clustering system is not as good as it in non-clustering one due to its high traffic overhead. With our extensions, their performance is effectively improved as the traffic overhead in them is reduced. We suggest that it is effective to apply our improvements in a clustering multiprocessor system. Effective block replacement policy makes it efficient to implement bus protocols in the intra-clustering system. It reduces the great replacement overhead encountered in the intra-clustering systems of the clustering multiprocessor systems.

In the future, we will propose new schemes to improve the performance of intra- and inter-clustering system in the clustering multiprocessor systems to explore further important design issues. Besides, our inter-clustering prefetching scheme can also be extended to improve the performance of software prefetching schemes.

Acknowledgement

This research was supported by the National Science

Council of the Republic of China under contract numbers: NSC 87-2213-E009-048 and NSC 87-2213-E009-049.

Reference

[1]J-P. Singh, W-D. Weber, and A. Gupta, "*SPLASH: Stanford parallel applications for shared-memory*", ACM SIGARCH Computer Architecture News 20(1), pp.5-44, Mar 1992.

[2]Debashis Basak and Dhabaleswar K. Panda, *Scalable Architectures with k-ary n-cube cluster-c Organization.* TR28-1993, Dept. of Computer and Information Science, The Ohio State University, Aug 1993.

[3]D.Basak and D.K.Panda, *Benefits of Processor Clustering in Designing Large Parallel Systems: When and How?*. Technical Report: OSU-CISRC-6/96-TR35, 1996.

[4]J.P. Su, *A study on Memory Subsystem Design for Multiprocessor System and Implementation of Its Simulation and Evaluation Environment*, The Master Thesis of Dept. CSIE, NCTU, June, 1996.

[5]Jia-Rong Wu, *A Study on Methods of Reducing Memory Access Latency for Clustering Multiprocessor System Design and Implementation of Its Simulation and Evaluation Environment*, The Master Thesis of Dept.CSIE, NCTU, June, 1998.

[6]R.H. Katz, S.J. Eggers, D.A. Wood, C.L. Perkins, and R.G. Sheldon. "*Implementing a Cache Consistency Protocol*". Proceedings of the 12th International Symposium on Computer Architecture, June 1985.

[7]IEEE SCI draft 2.00: *SCI Scalable Coherence Interface*, Draft Document for the IEEE SCI standard,1992.

[8]J.K.Archibald. "A Cache Coherence Approach For Large Multiprocessor Systems". *International Conference on Supercomputing*, pp.337-345, July 1988.

[9]Fredrik Dahlgren and Michel Dubois, "Sequential Hardware Prefetching in Shared-Memory Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, Vol.6, No.7, pp.733-746, July 1995.

[10]David Brain Glasco, *Design and Analysis of Updated-Based Cache Coherence Protocols for Scalable Shared-Memory Multiprocessors*, Technical Report No. CSL-TR-95-670, Computer Systems Laboratory Department of Electrical Engineering and Computer Science Stanford University,Standford, California 94305, June 1995.

[11]Daniel E. Lenoski and Wolf-Dietrich Weber, *Scalable Shared-Memory Multiprocessing*, Morgan Kaufmann Publishers, 1995.

[12]Mazin S.Yousif, M.J.Thazhuthaveetil, and C.R.Das, "Cache Coherence in Multiprocessors : A Survey", In *Advances In Computers*, Vol.40, Academic Press, Inc., 1995.

[13]Debashis Basak and Dhabaleswar K.Panda, "Designing Processor-cluster Based Systems : Interplay Between Cluster Organization and Broadcasting Algorithms", *International Conference on Parallel Processing*, Aug 1996.

[14]D.Lenowshi,and J.London,*Stanford DASH Multiprocessor*,Technical Report No.CS-TR-89-403, Dec 1989.

[15]Andrew Erlichson, Basem A. Nayfeh, Jaswinder P. Singh, and Kunle Olukotun, *The Benefits of Clustering in Shared Address Space Multiprocessors : An Applications-Driven Investigation,* Technical Report: CSL-TR-94-632, 1994.