Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Enhancing the SCI Cache Coherence Protocol for Multiprocessor Clusters

Kelvin Lin, Neng-Pin Lu, Yeong-Chang Maa*, and Chung-Ping Chung

Institute of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

*Advance Technology Center
Computer and Communication Research Laboratories
Industrial Technology Research Institute
Hsinchu, Taiwan, R.O.C.

Abstract

In this paper, we designed a cache coherence protocol for clustered multiprocessors. This protocol combines the IEEE SCI cache coherence protocol [7] with Write-Once protocol [5]. To evaluate and verify this protocol, we implemented this protocol on the PROTEUS parallel-architecture simulation system [2]. In addition, we used the enhanced PROTEUS system to investigate the performance differentiation between clustered, bus-based, and network-based multiprocessor systems. Through simulations, we found that the performance of the clustered multiprocessor is the best on heavily memory-loaded benchmarks.

1. Introduction

Multiprocessors have become a way to implement computer systems able to deliver high computing power, which is increasingly demanded by users today. This class predominantly encompassed tightly coupled systems, often called shared-memory multiprocessors. Such systems consist of a number of processors that access shared memory through a communication subsystem. Shared memory, which usually contains multiple memory modules, enables efficient and low-cost sharing of code and data among processors. The interconnection path between processors and common memory can be realized as a shared bus, or through some kind of interconnection network.

The predominant advantages of shared-memory multiprocessors are their cost-effectiveness and their very simple and general programming model, making them increasingly popular today. However, due to the contention on using shared resources, the average latencies in accessing shared memory tend to be longer, which degrades the performance. Therefore, the usual way to shorten the memory access in computer systems is the use of cache memory. Two types of cache memories can be found in multiprocessors: shared and private caches. Shared caches can reduce the average memory access time, though, they can not alleviate the contention for shared resources. Private caches are able to satisfy most of the memory references locally, eliminating the need to access shared memory. Both types of caches inherently impose another serious problem. Multiple copies of the same data can exist in different caches simultaneously, and they will cause cache coherence problem. There must be some mechanism to solve this problem, that is the cache coherence protocol.

Scalability is also an important design issue for multiprocessor systems. Clustering is one of the scaleable interconnection architectures. Clustering technique aggregates a few processors into a cluster node by a bus to provide high computation power and connects these cluster nodes by a inter-cluster network. In this paper, we proposed a two-level hierarchical cache coherence protocol for multiprocessor clusters, and established the PROTEUS simulation environment to verify the protocol and to evaluate the performance of multiprocessor systems that interconnected by bus, general network, and clustered architecture. The remaining of this paper is organized as follows. In Section 2, we review some existing cache coherence protocols, including the bus-based cache coherence protocols, directory-based cache coherence protocols, and the distributed directory cache coherence protocol defined by IEEE SCI std P1596-1992 [7]. In Section 3, we present our cluster architecture and the associated cache coherence protocol. Section 4 evaluates the performance differences among bus-based, network-based, and cluster-based multiprocessor systems. Finally, we conclude this paper in Section 5.

2. Cache Coherence Protocol

2.1 Bus-based Protocol

The bus-based cache coherence schemes depend on each cache controller by observing the bus transactions of all other processors in the system, taking appropriate actions to maintain coherence and the state of each cache line in the system is encoded in a distributed way among all cache controllers. Figure 1 illustrates a typical bus-based cache coherence protocol: Write-Once [5].

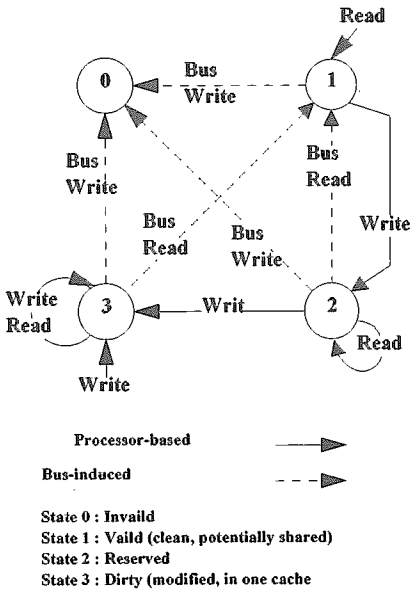

Fig. 1 Write-Once Cache Coherence Protocol

The Write-Once protocol works as follows:

(1) Read miss. If another copy of the block exists and is in state DIRTY, the cache with that copy inhibits the memory from supplying the data and supplies the block itself, as well as writing the block back to main memory. If no cache has a DIRTY copy, the block comes from memory. All caches with a copy of the block set their states to VALID.

(2) Write hit. If the block is already DIRTY, the write can proceed locally without delay. If the block is in state RESERVED, the write can also proceed without delay, and the state is changed to DIRTY. If the block is in state VALID, the word being written is written through to main memory (i.e., the bus is obtained, and a one-word write to the backing store takes place) and the local state is set to RESERVED. Other caches with a copy of that block (if any) observe the bus write and change the state of their block copies to INVALID. If the block is replaced in state RESERVED, it need not be written back, since the copy in main memory is current.

(3) Write miss. Like a read miss, the block is loaded from memory, or, if the block is DIRTY, from the cache that has the DIRTY copy. Then this cache controller invalidates its copy. Upon seeing the write miss on the bus, all other caches with the block invalidate their copies. Once the block is loaded, the write takes place and the state is set to DIRTY.

2.2 Directory-based Protocol

Three techniques exists for implementing the directory, namely the *full-mapped, limited* and *chained* directory. In the full-mapped case, information about all lines in all caches resides in memory. In recognition of the fact that usually a certain memory line will only exist in a few number of cached copies, a limited number of directory references could be implemented for each memory block. A limited pointers contain the cache ID exists along with a memory block. Instead of having a centralized directory containing information about all cache lines, the directory information can be distributed over the system as a linked list.

2.3 SCI Cache Coherence Protocol

The SCI cache coherence protocol is a chained directory-based protocol [7]. Figure 2 shows the distributed directory used in the protocol.

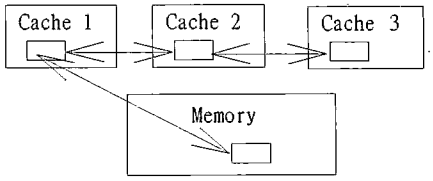

Fig. 2 A Distributed Directory in SCI Cache Coherence Protocol

The SCI cache coherence protocol supports a rich set of performance enhancement options, including *minimal set, typical set,* and *full set.* In this paper, we consider typical set only. Typical set has provisions for read sharing, purging of the list in the case of writing to a line. Subsequent read requests from other caches will achieve data form the cache having the dirty copy of the line. A cache in a list can roll out in case of a line replacement by making the previous and succeeding caches in the list point to each other. Special actions have to be taken when the cache rolling out is the head of the list. Direct memory access (DMA) controller access to the memory system is also supported by the typical set.

A memory line can either be the only copy in a system, or it can be cached in one or several caches. If no cached copies of the line exist, the memory line will always be the prevailing copy. This corresponds to the MS_HOME state in Table 1. If an active sharing list exists, the copies in the list will either be consistent

with the memory line or not. If all copies are consistent with memory, the memory line will be in the MS_FRESH state. Otherwise the memory line is said to be gone, i.e. the memory line is no longer consistent with the prevailing cached copy. The corresponding state is MS_GONE. The above memory line states are outlined in Table 1.

Name	Description
MS_HOME	No sharing list exists and memory is valid
MS_FRESH	Sharing list copies are identical with memory
MS_GONE	Sharing list copies might be different from memory

Table 1 Typical Set Stable Memory Coherence States

A cache line can either be invalid or active. If it is invalid, it can be replaced directly in case of a subsequent cache miss. The invalid state is indicated by CS_INVALID in Table 2. When a line is active, it might either be consistent with memory or not. When the line has not been written by a processor it is consistent with memory, and is therefore said to be fresh. If it furthermore is the only cached copy, the corresponding state is CS_ONLY_FRESH. If, however, the processor attached to the cache performs a write to the line, it will enter the CS_ONLY_DIRTY state. The cache will inform memory about the write, but memory will not be updated. Further writes to line by the cache in the CS_ONLY_DIRTY state will not be reported to memory.

If several cached copies exist in the sharing list, the same principle as described above applies to the head of the list. It will either be in the CS_HEAD_FRESH or CS_HEAD_DIRTY state. The CS_HEAD_FRESH case is rather trivial, since the head will be consistent with memory. The head of the list will always be the only cache that might gain write permission to the line. Therefore, when the head and memory are consistent, all elements within the list will be so too. Elements residing in the middle of the list will be in the CS_MID_VALID state, and the tail element will be in the CS_TAIL_VALID state. If a processor predicts that it sometime in the future will need write permission to a given cache line, it might add itself as the head of the list and by the same time invalidate memory. The cached line will then enter the CS_HEAD_DIRTY state. This action will, however, not invalidate the rest of the list as long as the processors do not write to the line being the head of the list. Mid elements will still be in the CS_MID_VALID state, and the tail will be in the CS_TAIL_VALID state. Only when a write is actually performed, the rest of the list will be purged. The above cache line states are outlined in Table 2.

Name	Description
CS_INVALID	Line is invalid and can be used for caching new lines
CS_ONLY_FRESH	Only cached copy, consistent with memory
CS_ONLY_DIRTY	Only cached copy, write-able and inconsistent with memory
CS_HEAD_FRESH	Head of fresh list, consistent with memory
CS_HEAD_DIRTY	Head of valid list, write-able and inconsistent with memory
CS_MID_VALID	Mid element in valid list, possibly inconsistent with memory
CS_TAIL_VALID	Tail of valid list, possibly inconsistent with memory

Table 2 Typical Set Stable Cache Coherence States

Table 3 shows the memory states and cache states of a shared list in SCI cache coherence protocol.

Memory states	Head	Mid	Tail
MS_ HOME	-	-	-
MS_ FRESH	CS_ONLY_ FRESH	-	-
MS_ FRESH	CS_HEAD_ FRESH	-	CS_TAIL_ VALID
MS_ FRESH	CS_HEAD_ FRESH	CS_MID_ VALID	CS_TAIL_ VALID
MS_ GONE	CS_ONLY_ DIRTY	-	-
MS_ GONE	CS_HEAD_ DIRTY	-	CS_TAIL_ VALID
MS_ GONE	CS_HEAD_ DIRTY	CS_MID_ VALID	CS_TAIL_ VALID

Table 3 Stable Memory States and Cache States in a Shared List

3. Cache Coherence Protocol for Multiprocessor Clusters

3.1 Clustered Architecture

Figure 3 shows the clustered architecture. In our design, each cluster equips with a cluster local memory (LM), a inter-cluster cache (IC), four processors (P) with local caches (LC) of its own attaching on a cluster bus (CB). The clusters connect through a interconnect network (IN) in some topology. The inter-cluster cache caches the remote usually used data to reduce the long memory access latency. The main design issue of ICs is the inclusion property in memory hierarchy. That is, the data residing in a LC must have a copy in IC or in LM of this cluster, and the total data set in the LCs in a

cluster is the subset of the union of those in LMs and ICs of this cluster, while the data sets of LMs and ICs of the same cluster are disjointed. The cluster cache or local memory will send invalidation signals to the local caches when a location is no longer available or flush signals to get the dirty data in local caches. The cluster cache and local memory will serve as a cache coherency monitors for all the local caches connected to them.

3.2 Memory Hierarchy

As illustrated in Figure 4, there are five hierarchies in the clustered architecture. The first level is the processor's cache that is designed to match the processor speed. A request can not be served by the processor's cache is sent to the local cluster level which includes the other processors' caches within the requesting processor's cluster. Otherwise, the request is sent to the cluster level. This level includes the cluster cache and memories associated with this cluster. When the request can not be served in this level, it is send to the home directory level. The home directory level consists of the cluster that contains the directory and physical memory for a given memory address. If the directory entry is in a dirty state, or in a shared state when the requesting processor requests exclusive access, the fifth level must also be accessed. The remote cluster level for a memory block consists of the clusters marked by the directory as holding a copy of the block.

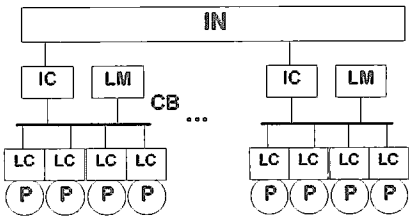

Fig. 3 Clustered architecture

Figure 4 also shows the inclusion relationships between any two levels. The data set in a larger rectangle includes that of the upper rectangle. The data sets of inter-cluster cache and local memory are disjointed while the union of these two data sets compose the third level of the memory hierarchy.

3.3 The Cache Coherence Protocol

The protocol for data consistency among inter-cluster nodes is the SCI cache coherence protocol. The directory of SCI cache coherence protocol is distributed in LMs and ICs in Figure 3. The LMs contain the data and the head pointer of directory of this memory block, while the ICs caching the same memory block in other clusters contain the foreword and backward pointers as well as the data. Data consistency within a cluster node is done by Write-Once cache coherence protocol. For snoopy protocol, there is not any state in memory system, so the ICs and LMs do not contain any state for Write-Once protocol but just monitor the bus transitions in the way similar to the snoopy protocol. LCs contain states for Write-Once cache coherence protocol.

A datum in a LC must have a copy either in IC or in LM of this cluster. The SCI state of this copy will restrict the Write-Once cache state of the datum in LC. If a datum is in IC, it means this datum is a copy of the memory block addressed in some other cluster. Then the possible combinations of SCI cache state of this copy in IC and Write-Once cache state of the datum in LC are following:

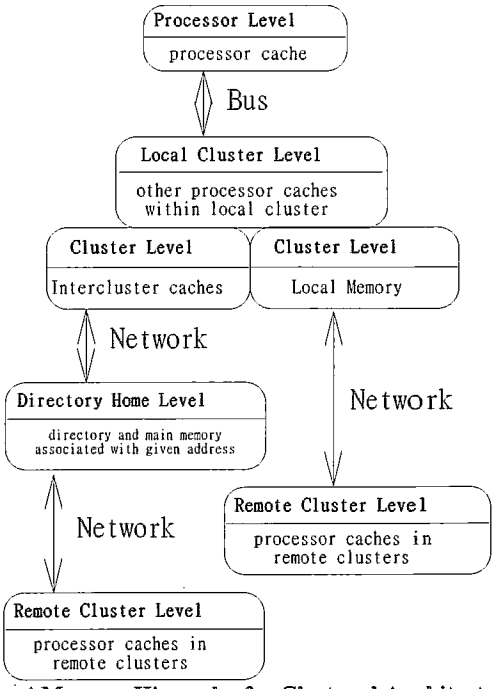

Fig. 4 Memory Hierarchy for Clustered Architecture

The notation INVALID+ means at least one cache state is INVALID. The notation INVALID* means there may be more than zero cache line in INVALID state. The notation VALID+ and VALID* have the same meanings like INVALID+ and INVALID*. The RESERVED or DIRTY means there exists only one cache in RESERVED or DIRTY state respectively.

When the datum in IC is in CS_INVALID state, it is impossible that the copy in LC is in other state rather than INVALID. When the IC state is in read-only state, i.e., CS_ONLY_FRESH, CS_HEAD_FRESH, CS_

MID_VALID or CS_TAIL_VALID, the possible LC states associated with this cache line are either in INVALID state or some in state VALID and the others in INVALID state. It is because when data in IC are readable, the copies may appear in LCs in readable states. When the IC state is in a writable state, for example in CS_ONLY_DIRTY state, the states of its copies in LC could all be in INVALID, or some are in VALID, the others are in INVALID, or one is in RESERVED or DIRTY and the others are in INVALID, since the LCs can be in writable state when IC gets the ownership of this cache line. The state CS_HEAD_DIRTY is a special case in this classification. The datum in state CS_HEAD_DIRTY is permitted to write only after it purges the shared list, so the datum in state CS_HEAD_DIRTY is seen as read-only.

IC state	LC state
CS_INVALID	INVALID+
CS_ONLY_DIRTY	INVALID+ or INVALID* VALID+ or INVALID* RESERVE or INVALID* DIRTY
CS_ONLY_FRESH, CS_HEAD_FRESH, CS_HEAD_DIRTY, CS_MID_VALID, CS_TAIL_VALID	INVALID+ or INVALID*VALID+

Table 4 Possible Combinations of SCI Cache State and Write-Once Cache State of the Copies of the Same Memory Block

If a datum is in LM, it means this local memory of this cluster is the home location of this datum. Also, the SCI memory state of this copy will restrict the Write-Once cache state of the datum in LCs. The possible combinations of SCI cache state of this copy in IC and Write-Once cache state of the datum in LC are following:

LM state	LC state
MS_HOME	INVALID+ or INVALID* VALID+ or INVALID* RESERVED or INVALID* DIRTY
MS_FRESH	INVALID+ or INVALID*VALID+
MS_GONE	INVALID+

Table 5 Possible Combinations of SCI Memory State and Write-Once Cache State of the Copy

When the datum in LM is in MS_GONE state, it is impossible that the copy in LC is in other state rather than INVALID. When the LM state is in read-only state, i.e., MS_FRESH, the possible LC states associated with this cache line are either in INVALID state or some in state VALID and the others in INVALID state. It is because when data in LM are readable, the copies may appear in LCs in readable states. When the LM state is in MS_HOME state, the states of its copies in LC could all be in INVALID, or some are in VALID, the others are in INVALID, or one is in RESERVED or DIRTY and the others are in INVALID, since the LCs can be in writable state when LM gets the ownership of this cache line.

3.4 IC State Transition Diagram

The state transition table for the clustered cache coherence protocol induced by intra-cluster bus transition is shown in figure 5. Here, we omit the transient states of SCI and Write-Once states in LCs. The initial states of the cache lines are CS_INVALID. Each bus transaction requesting to IC may cause the state transition of cluster cache. The most state transitions are caused by cluster bus read (CBR) or cluster bus write (CBW). Others are caused by the replacement. A cache line watching a CBR or a CBW in CS_INVALID state will send *mread64.cache_fresh* or *mread64.cache_dirty* (first item in parentheses in figure 5) to memory controller respectively. The next step after these two transitions depends on the memory state. If the memory state is in MS_HOME, the requesting cache will be the first element of the shared list and its state will be in CS_ONLY_FRESH for read request or CS_ONLY_DIRTY for write request. Then the memory state will goes from MS_HOME to MS_FRESH (indicated by MS(H->F)) or to MS_GONE (indicated by MS(H->F)) respectively and the first cache of this shared list is born (indicate by CS(NULL->OF) or CS(NULL->OD)). If the memory state is in MS_FRESH, telling the existence of shared list, the requesting cache will be the first cache in CS_HEAD_FRESH state of the shared list for read and, otherwise, be the first cache in CS_HEAD_DIRTY state temporarily and then goes to the state CS_ONLY_DIRTY for write. The memory state will be the same if request is a read or changes to MS_GONE if request is a write.

If the memory state is in MS_GONE, any request will not change memory state. The state of the requesting cache will goes to the CS_HEAD_DIRTY for reading or CS_ONLY_DIRTY via CS_HEAD_DIRTY for writing. The cache state in either CS_MID_VALID or CS_TAIL_VALID can satisfy the request of CBR locally, but it will roll out before a write can be continued. It then goes to CS_INVALID state to competing the addition to the

head of the shared list as if it was originally in CS_INVALID state.

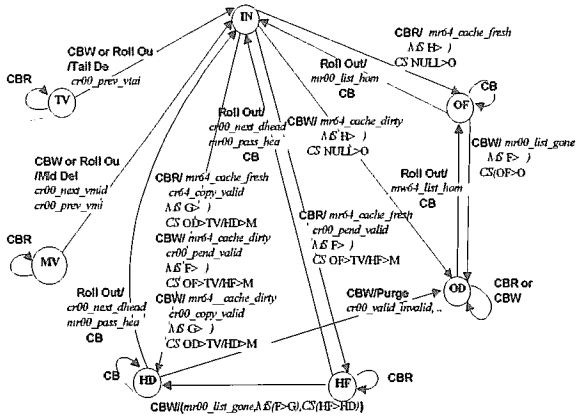

Fig. 5 State Transition Induced by Intra-Cluster Bus Transition. IN:CS_INVALID, OF:CS_ONLY_FRESH, OD:CS_ONLY_DIRTY, HF:CS_HEAD_FRESH, HD:CS_HEAD_DIRTY, MV:CS_MID_VALID, TV:CS_TAIL_VALID, CBR: cluster bus read, CBW: cluster bus write, CBI: cluster bus invalidation, Purge: head cache purge shared list, Roll Out: replacement cache line. The contents in parentheses are the packet names to be sent, the state transitions of replying memory and the state transitions of the first cache of this shared list.

When a cache line is selected for replacement, this cache must send at least two packets (if no retry happens) to inform its neighbors to modify the pointers to link each other. After done the modifications, its neighbors will send acknowledged messages to this cache line. This cache line then goes to a transient state and issues a flush transaction to the bus to invalidate the data copies in LCs.

Figure 6 shows the state transition diagram induced by inter-cluster operations of a cluster. The main actions are shown in bold form. These actions are caused by receiving the packets (shown in italic form in figure 6) from neighbors. The difference between this protocol and SCI protocol is that the purge of the shared list and the flush of the shared list may issued by the memory controller (Mem purge, Mem Flush in figure 6). When the cache state in CS_ONLY_DIRTY goes to any other state rather than CS_INVALID, it must issue a flush transaction to flush the dirty data possibly residing in LCs from the OWNED state to SHARED state. When a caches copy in IC in any state rather than CS_INVALID changes to the state CS_INVALID, the IC controller must issue an invalidation transaction to invalidate the data possibly residing in LCs from SHARED or OWNED state to INVALID state before it go to the CS_INVALID state.

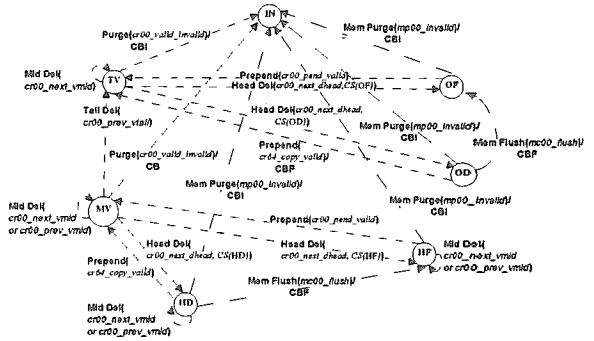

Fig. 6 State Transition induced by Inter-Cluster Operations. IN:CS_INVALID, OF:CS_ONLY_FRESH, OD:CS_ONLY_DIRTY, HF:CS_HEAD_FRESH, HD:CS_HEAD_DIRTY, MV:CS_MID_VALID, TV:CS_TAIL_VALID, CBI: cluster bus invalidation, CBF: cluster bus flush. The bold style describes the actions and the contents in parentheses are the packet name to be sent, and an optional state that is included in reply packet.

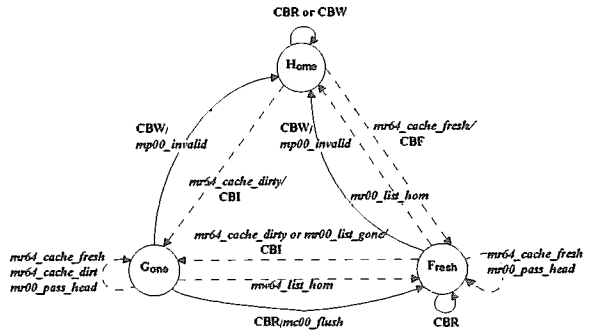

Fig. 7 Memory State Transition Diagram. Home: MS_HOME, Fresh: MS_FRESH, Gone: MS_GONE, CBR: cluster bus read, CBW: cluster bus write, CBI: cluster bus invalidation, CBF: cluster bus flush

3.5 LM State Transition Diagram

Figure 7 shows the state transition of memory states. The solid line indicates the active operation caused by the cluster bus transition. The dashed line indicate the passive operation caused by the reception of the packet from inter-cluster interconnection. These transitions includes some undocumented messages. Because the LM must have the ability to access the ownership of a memory block for further write of the processor within the same cluster, it must have the mechanism different from SCI protocol to purge shared list or clean the dirty shared list. These are done by sending message *mp00_invalid* or *mc00_frush*. The memory state then goes from MS_FRESH to MS_HOME or MS_GONE to MS_HOME for write or goes from MS_GONE to MS_FRESH for read, respectively. When receiving the read request from other IC, the local cluster bus controller must issue a flush transition to guarantee to get the latest version of

data in this cluster and the memory state changes from MS_HOME to MS_FRESH. If the request is for write, the local cluster bus controller must issue an invalidation transition to invalidate all the copies in LCs and the memory state goes from MS_HOME to MS_GONE or from MS_FRESH to MS_GONE.

4. Performance Evaluation

4.1 PROTEUS System

PROTEUS [2] is a execution-driven simulation environment. It is not actually a simulator; rather, it is a simulation engine that combines with architecture-specific modules and user applications to create a simulator. The resulting executable provides high-performance simulation of the user's application on the target architecture.

We have modified PROTEUS environment to simulate the hierarchical architecture with inter-cluster connection network of ring and the intra-cluster connection of bus system. The main differences between this simulator and original version are the memory hierarchy as well as cache coherence protocols. The 7 stable states and more than 20 transient states make SCI protocol very complex. The most important implementation of SCI protocol is the forward progress of shared list creation. That is, the transient state must be able to change to a stable state and the retry message must be overcome in a finite times of retry.

4.2 Benchmark Programs

We choose two applications from the SPLASH suite [11] as our benchmark programs.

MP3D solves a problem in rarefied fluid flow simulation. Two large arrays of structures account for more than 99% of the static data space used by MP3D. The amount of data accessed by MP3D is largely determined by the number of molecules simulated. The user specifies the initial number of active molecules as an argument to the application, and over the course of a simulation run the number of active molecules typically increases by about 25%. There are about 71% for shared variable reads (of all reads) and 80% for shared variable writes.

Barnes-Hut simulates the evolution of a system of bodies under the influence of gravitational forces. It is a classical gravitational N-body simulation, in which every body is modeled as a point mass and exerts forces on all other bodies in the system. There are about 25.5% for shared variable reads (of all reads) and 1.3% for shared variable writes.

4.3 Simulation Results

For MP3D, the input file *test.geom* is used to evaluate the performances. We use 1000 particles and run 100 steps, to increase the parallel execution time. The cache size (LC) is reduced to 4k bytes compared to the problem size. The bus access time and the memory access time are all 6 cycles in bus system. The packet length of SCI is 16 bytes (plus the cache line size if with data). The IC is 16k bytes of 2 way associative cache with 16 bytes per line. The size of LM is 1M~2M bytes. The IC access time is the same as that of LM and is 6 cycles. For the parameters of the cluster, each cluster contain 4 processors with LCs of its own attaching to a cluster bus. The network topology among the cluster nodes is a ring.

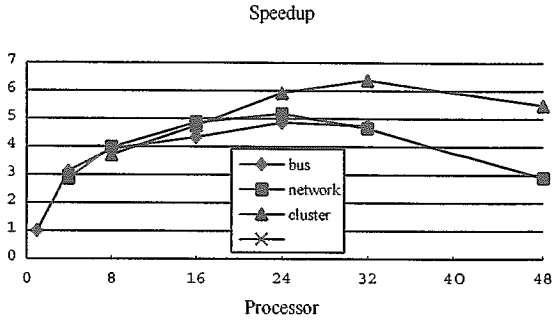

Fig. 8 Speedups of MP3D for Bus, Network and Cluster

From the figure 8, we can watch out the performance speed up on these three types of architectures. MP3D is a heavily memory loaded program. The scalability of clustering is better than those of network system and bus system. The degradation of each line means the saturation of the resources. For the bus system, the bus is saturated on the point of 24 processors that is caused by serial bus access. The network system is saturated on the point of 24 processors connected by ring using SCI cache coherence protocol. It almost needs more 4 messages to complete a read or a write (writes may need more messages due to the invalidation). The topology is also a reason that the network saturation on such a point. The messages need more time to transfer in a ring structure in a serial matter. Cluster behaves better than the others. Because the reduction of the network size makes the cluster speedup more.

For Barnes-Hut, the input parameters are following:

initfile:null, nbody:1024, seed: default, outfile: null, dtime: 0.025, eps: 0.05, tol: 0.6, fcells: 0.8, tstop: 0.25, and dtout: 0.25. The other architecture parameters are the same as that used to simulate MP3D.

Figure 9 shows the speedups of executing Barnes-Hut program on these three types of architectures. The scalability of clustering is better than that of the network system but worse than that of the bus system,

and the overall speedup is more than that of the MP3D. This is because the Barnes-Hut program has a light percentage of shared variable references compared with MP3D. The load on memory system in this program is small due to the little amount of shared variable references. The access time in bus system is the bus transaction time plus the arbitration time while the access time in network grows with the size of the network. The arbitration time is small compared to the propagation time for sending a packet in the increased nodes. If the needed data are distributed randomly in all memory modules, the bus transaction time for a bus system from different memory modules does not increase, but the time of the network system will. This may cause that the speedup of the cluster is worse than the bus system no matter how small of the network size.

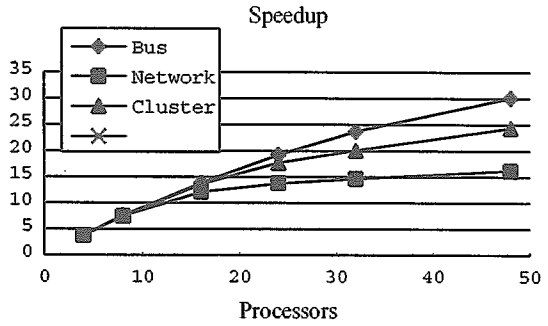

Fig. 9 Speedups of Barnes-Hut for Bus, Network and Cluster

5. Conclusions

In this paper, we have enhanced the SCI cache coherence protocol for clustered architecture and implemented a cluster simulation environment based on PROTEUS system. Then, we use the modified PROTEUS system to investigate the behavior of the real applications and evaluate the performance on different hardware configurations. It shows that the cluster really has benefits in system architecture. It is the trade off between bus-based system and network-based system in fast computing power and scalablility. From the simulation results, we obtain the following conclusions:

(1) Memory access time through network system is usually longer than that through bus, but the bus must serially serve these requests. Network packets can be transfer in parallel, thus hiding the more packet transfer time. Therefore, the scalability of a network system is better than bus system.

(2) Cluster interconnection is a more scalable architecture than a bus-based system or a network-based system when the memory system is heavily loaded. It is because of the reduction of bus contention by clustering fewer processors into a node and the reduction of network size by interconnecting these nodes. Thus, the performance can be improved.

(3) Clustered architecture is also an approach to scale the machine size when the memory system is lightly loaded. Though the performance of the bus system is the best, bus-based system has its potential problems, such as the long circuit propagation delay due to large number of devices connecting to it. At this situation, cluster architecture may be a good extension of a bus system.

References

[1] J. Archibald and J.-L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," ACM Trans. Computer Systems, Vol. 4, No. 4, Nov. 1986, pp. 273-298.

[2] E.A. Brewer and C.N. Dellarocas, "PROTEUS User Documentation Version 0.5," Dec. 1992.

[3] M. Dubois and S. S. Thakkar edited, Cache and Interconnect Architectures in Multiprocessors, Kluwer Academic Publishers, 1990.

[4] D.D. Gajski and J.-K. Peir, "Essential Issues in Mulitprocessor Systems," IEEE Computer, pp. 9-27, Jun 1985.

[5] J.R. Goodman, "Using cache-memory to reduce processor-memory traffic," in Proc. of 10th Int'l Symp. Computer Architecture, pp. 124-131, June 1983.

[6] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill Inc., 1993.

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

[8] R. L. Lee, P. C. Yew, and D. H. Lawrie, "Multiprocessor Cache Design Considerations," in Proc. 14th Symp. Computer Architecture, pp. 253-262, 1987.

[9] D. Lenoski, et al., "The Directory-based Cache Coherence Protocol for the DASH multiprocessor," in Proc. 17th Symp. Computer Architecture, pp. 148-159, May 1990.

[10] D. Lenoski, et al., "The Stanford Dash Multiprocessor," IEEE Computer, pp. 63-79, March 1992.

[11] J.P. Singh, W.-D. Weber and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," Technical Report, Stanford University.

[12] P. Stenstrom, "A Survey of Cache Coherence Schemes for Multi-processors," IEEE Computer, Jun. 1990.