ELSEVIER

The Journal of Systems and Software 58 (2001) 135–152

The Journal of Systems and Software

www.elsevier.com/locate/jss

Design of a scalable multiprocessor architecture and its simulation

Der-Lin Pean, Chao-Chin Wu, Huey-Ting Chua, Cheng Chen *

Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 30050, Taiwan, ROC

Received 15 April 2000; received in revised form 21 September 2000; accepted 11 November 2000

Abstract

Performance enhancement and system scalability are two of the most important issues in the design of multiprocessor systems. A scalable cluster-based multiprocessor architecture and its simulation environment called SEECMA are proposed. Several new issues in our architecture, including scalable cache coherence protocols, relaxed memory consistency models, memory optimization techniques and several types of processors are considered. It has been developed to meet current trends in clustering architecture design. Additionally, the SEECMA environment is presented as a helpful investigation tool for both education and research. In addition to many simulation options, it is provided with a user-friendly graphic interface. SEECMA can automatically collect data from several simulation runs and display the results for comparison. So far, we have evaluated several vital issues of cluster-based multiprocessors on SEECMA including effective prefetching and replacement policies, and optimization of migratory sharing using both hardware and software mechanisms. On average, these enhance system performance by up to 8%, 9% and 7%, respectively. Our cluster-based multiprocessor architecture also scales more readily than the current general, or cluster-based, multiprocessor environments.

Keywords: Cluster-based multiprocessors; Shared memory; Program-driven simulation; Performance evaluation; Multithreaded processor

1. Introduction

The gap between the computing power of microprocessors and that of the largest supercomputers is shrinking, while the price/performance advantage of microprocessors is increasing. This clearly points to using microprocessors as the computation engines in large multiprocessor systems. The challenge lies in building a machine that can scale up its performance while maintaining the initial price/performance advantage of the individual processors. Scalability allows a parallel architecture to leverage commodity microprocessors and small-scale multiprocessors to build larger scale machines. These larger machines offer substantially higher performance, which provides the impetus for programmers to port their sequential applications to parallel architectures instead of waiting for the next higher performance uni-processor. Due to the rapid progress in VLSI and packaging technologies, they have become a driving force in the design and development of highly scalable parallel systems using cluster-based architecture, simultaneously exploiting local communication. Currently there are commercial supercomputers constructed with a cluster-based architecture, such as the CONVEX SPP series (CONVEX, 1994). Meanwhile, system designers must rely upon a convenient and accurate simulation environment to verify their designs or determine the most cost-effective strategies. Our cluster multiprocessor and its simulation environment, called simulation and evaluation environment for cluster-based multiprocessor architecture (SEECMA), are designed to address the above issues. SEECMA is the outcome of follow-up work on simulation and evaluation environment for shared-memory multiprocessor architecture (SEESMA) (Wu et al., 1998c).

Our architecture is designed for high performance and scalability. Our main purpose is to construct effective memory and interconnection architectures. After our evaluation, we found that our architecture has the following desirable attributes.

1. A multithreaded processor architecture in a cluster-based system is possible, which is more effective when combined with our proposed mechanisms.
2. Several memory consistency models (Wu, 1998a; Wu and Chen, 1998b) to improve the parallelism of the system were constructed, all resulting in improved system performance.

* Corresponding author. Tel.: +886-35712121 ext. 54734; fax: +886-3-5724176.

E-mail address: cchen@csie.nctu.edu.tw (C. Chen).

0164-1212/01/$ - see front matter
PII: S0164-1212(01)00034-6

3. We propose several new cache coherence protocols and directory structures in the linked-base type of cluster-based architecture, and these also perform better than existing protocols. We also demonstrate that some cache coherence protocols that perform well in non-cluster-based systems perform poorly in cluster-based systems.
4. Several new prefetching and replacement policies are presented to improve the efficiency of the cache system, and simulation results prove these to be effective.
5. With the assistance of these effective mechanisms, we find our architecture has high scalability when evaluating performance.

As SEECMA is extended from SEESMA (Wu et al., 1998c), the kernel simulator is the MINT package (Veenstra and Fowler, 1994), which was originally developed at the University of Rochester. SEECMA aims to provide a simulation and evaluation environment for cluster-based multiprocessor systems. It supports the following simulation functions:

1. two types of processor architectures;
2. two-level caches with write caches (WCs);
3. a message-passing-based interconnection network;
4. four types of memory consistency models;
5. five types of cache coherence protocols;
6. three types of cache coherence directory structures;
7. effective replacement policies; and
8. effective prefetching schemes.

SEECMA is equipped with versatile simulation options and users may customize the target memory architecture by clicking on the appropriate buttons in a user-friendly X-windows interface.

Users are provided with either a menu or graphic input environment through a Graphic User Interface (GUI). The menu-based interface is similar to conventional windows functions while the graph-based interface is more convenient for beginners. Users move the cursor around various architectural components and click on them to select the required simulation options. Each time a simulation option is located, the graph is updated. On-line help is also supported. Other than the above, SEECMA automatically collects numerical data from several simulation results, and displays the corresponding statistical graphs. So far, both bar charts and line charts are available. Using these graphs, users are able to compare the performance of different architectural parameters.

SEECMA, with its abundant simulation features, serves as a valuable research environment for system designers who are interested in cluster-based multiprocessor architecture. We have used SEECMA on many research issues related to memory subsystems, including cache coherence protocols, memory consistency models, interconnection networks, cache hierarchies, migratory sharing (Su et al., 1996), as well as prefetching and replacement policies (Pean et al., 1998a).

As a result of simulation, we found that our new mechanisms, developed in our cluster-based architecture, performed much better than current methods. We illustrate some of them in the following. Note that all the comparisons, including non-cluster system architecture and DASH (Lenoski and Laudon, 1989) system architecture, are made based on the simulations performed using SEECMA. Our inter-clustered cache prefetching mechanism performs, on average, about 8% better than the original non-inter-clustered cache system for benchmarks in the SPLASH benchmark suite (Singh et al., 1992; Woo et al., 1995). Our effective replacement policy performs, on average, about 9% better than the original replacement policy. We propose a migratory-sharing optimization mechanism; the total system performs 7% better, on average, than the system without our optimization. Other effective mechanisms also perform well, and we describe them in detail in the following sections.

The rest of the paper is organized as follows. Section 2 gives an overview of the overall system architecture. Section 3 describes the main system features and design issue considerations. An overview of the SEECMA simulation and evaluation environment is given in Section 4. Section 5 evaluates the performance of our architecture on SEECMA, and Section 6 gives our conclusions and outlines future work.

2. Overall system architecture

Our system architecture is a cluster-based distributed shared-memory multiprocessor system. The general architecture of our target machine is shown in Fig. 1. Within our clustering architecture are multiple cluster nodes interconnected by a k-ary n-cube network. Each cluster node contains local shared-memory, an inter-cluster cache, a few processor environments (PEs), and a local bus.

The inner structure of a cluster node is shown in Fig. 1(b), and either MIPS R3000 processors or PMPs (Hirata et al., 1992) are used throughout the system. There is a single Local Shared Memory (LSM) associated with each cluster node to reduce memory contention and improve data locality. The inter-cluster cache is implemented with the goals of further facilitating data sharing among the clusters and utilizing data locally; it contains data that are usually referenced by the intra-cluster processors. The two components inside the Inter-Cluster Processor (ICP) are the input and output controllers; they are responsible for packeting request and acknowledgement. The local bus acts as an intra-connection network among intra-cluster PEs, the inter-cluster cache and LSM. Scalability is

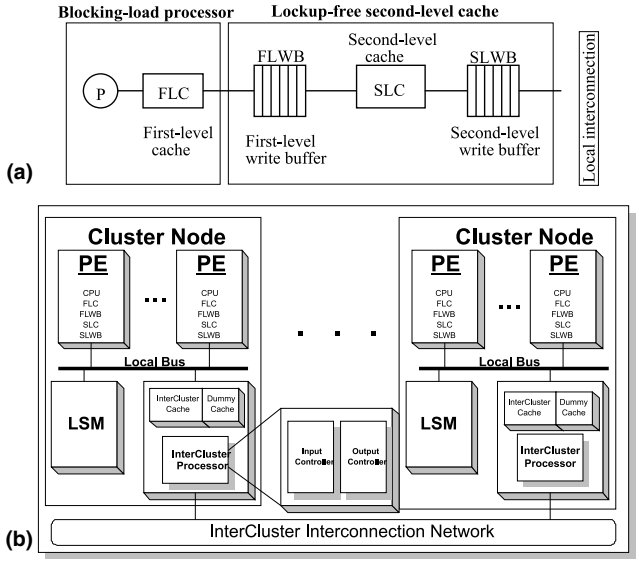


Fig. 1. (a) Processor environment. (b) The overall architecture of our cluster-based multiprocessor.

not a major concern in small-scale architectures, and we have therefore adopted a well-known snoopy-based protocol, Berkeley Protocol (Mazin et al., 1995), to maintain intra-cluster cache coherence. On the other hand, scalability is maintained through the global interconnection network by using the enhanced IEEE SCI Standard (IEEE SCI, 1992).

Shared memory is divided into several equal-size pages that are allocated to nodes in a round-robin fashion. Each processor environment (PE) includes a two-level cache hierarchy associated with write buffers, as presented in Fig. 1(a). In order to support released memory consistency models, we have added a lockup-free second-level cache (SLC) (David, 1995). The first-level cache (FLC) is a write-through on-chip cache whereas the SLC is a copy-back cache. Both caches are direct-mapped with the same line size in both, and full inclusion is supported. If a block is presented in the FLC, it will also be present in the SLC. On the other hand, the SLC is lockup-free while the FLC is blocking with an invalidation pin, so that a block can be invalidated outside the processor. All coherence actions associated with the system-level cache coherence protocol are handled by the SLC, inter-cluster cache and memory controller. Between the FLC and the SLC a first-level write buffer (FLWB) is used to avoid processor stalls on write accesses. Moreover, to make the SLC lock-free, for when multiple outstanding write requests are allowable, a second-level write buffer (SLWB) is also included to store all of the writes that induce global actions.

This clustering system brings a few benefits, such as resource sharing, exploitable packaging technologies, and allowing processors within clusters to share data in a more effective manner.

3. Main architecture features and design issue considerations

3.1. Cluster-based multiprocessor and interconnection network

A cluster-based multiprocessor system has multiple cluster nodes interconnected through a k-ary n-cube network, where each cluster node is assembled by several PEs linked by a local bus. Each cluster node is a small-scale multiprocessor system and multiple clusters form a large-scale system. This clustering architecture benefits from both small- and large-scale architectures, and hence is extremely attractive. The number of cluster nodes and PEs per node is specified by the users. If there is a single PE for each cluster node, then the cluster-based multiprocessor architecture becomes a typical multiprocessor system.

As stated by Dally (1990), most modern concurrent computers use k-ary n-cube or isomorphic to k-ary n-cube networks, such as rings, meshes, tori, direct and indirect binary n-cubes and Omega networks. In order to guarantee generality, a k-ary n-cube network is chosen for our system and we provide specific parameter adjustments: (1) radix, (2) dimension, (3) switch delay, (4) wire delay of a link, (5) link width and (6) uni-directional or bi-directional transmission. These provide for various research purposes.

3.2. Inter-cluster prefetching scheme

Data prefetching is one of the more useful approaches to effectively exploit local communication in our system. The prefetching scheme in clustered multiprocessors is slightly different to that of non-clustered multiprocessors, because data access may go through intra- and inter-clustering interconnection networks.

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 misses occur. In addition to the original memory accesses, there are multiple repeats of such accesses when prefetching is used. On the other hand, the traffic in every level of clustering multiprocessor systems also increases several times as the amount of prefetching data is increased. This causes the prefetching accesses to be delayed at every level of the memory hierarchical levels. Thus, we implement the prefetching mechanism in the inter-clustering cache rather than the SLC of our clustering multiprocessor system as shown in Fig. 2. The inter-clustering prefetching scheme extracts some performance gains by decreasing the 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 required in this clustering node in

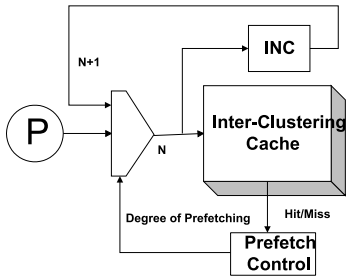


Fig. 2. Hardware structure of the inter-cluster prefetching scheme.

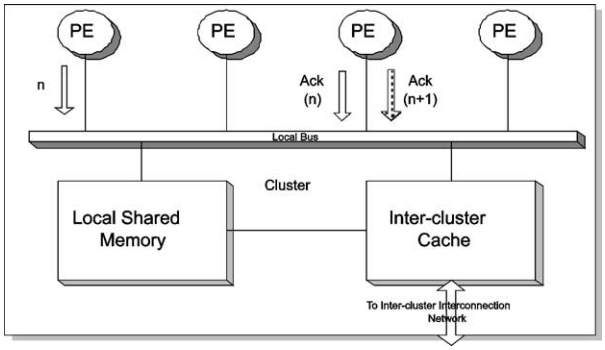


Fig. 3. Traffic request of inter-cluster prefetching scheme.

the near future. This inter-clustering prefetching technique causes less local bus requests, because it issues fewer read miss requests, and this may improve the clustering system. Here, we send only one request at every read miss request. As shown in Fig. 3, while reading for block n misses, it issues only the read miss request for block 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 can reply for these blocks. Traditional hardware prefetching mechanisms use a small cache block size; the advantages and disadvantages of the strategy are possibly avoiding false sharing misses, but bring in apparent traffic overhead. Therefore, instead of the access count widely used in previous hardware prefetching mechanisms, the inter-cluster prefetching (ICP) sends only one request during read misses.

In this way, the ICP scheme involves fewer local bus requests and extracts performance gain from decreasing network contention and traffic overloading. As a result, it will improve the locality of intra-clustering data and we show the figure of merit in Section 5.

3.3. Effective block replacement scheme

As limited cache size introduces cache conflict the replacement policy is one of the fundamental issues in cache design (Culler et al., 1999). The replacement policy adopted for our architecture is based on the Berkeley protocol (Mazin et al., 1995). Fig. 4 is the transition diagram of the Berkeley protocol, with four 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). A block in either the Dirty-Shared or Dirty-Exclusive state must be written back to main memory if it is selected for replacement. Dirty-Exclusive must be associated with only one cache. Dirty-Shared can be in only one cache, but it may be Clean-Shared in other caches. In addition, the Berkeley protocol uses the idea of ownership; if a block is not owned by any cache, memory is the default owner. The consistency rules are:

(1) *Read Miss*: If a block is Dirty-Shared or Dirty-Exclusive, the cache with that copy must provide block contents to the other cache directly, and set its local state as Dirty-Shared. Otherwise, it is loaded from main memory. In any case, the state of the requesting cache block is set to Clean-Shared. Note that the target block always sources from its owner.

(2) *Write Hit*: If a block has remained in the Dirty-Exclusive state, write will proceed without delay. If it is Clean-Shared or Dirty-Shared, an invalidation signal must be sent through the bus before write can proceed. All other caches invalidate their copies upon matching of block address, and the local state is changed to Dirty-Exclusive in the originating cache.

(3) *Write Miss*: Similar to read miss, the requested block is always provided from the owner. Any cache that has copies should transfer their state to Invalid. At the same time, the block in the requesting cache is updated as Dirty-Exclusive.

If the replacement shows up in a Clean-Shared cache line, some transaction must be done in advance. An example is illustrated in Fig. 5. Initially the states of specific blocks P0, P2 and memory are Dirty-Shared, Clean-Shared and Gone, respectively. The cache in P2 first issues a block replacement request. Once P0 receives the request, and no other processors have that

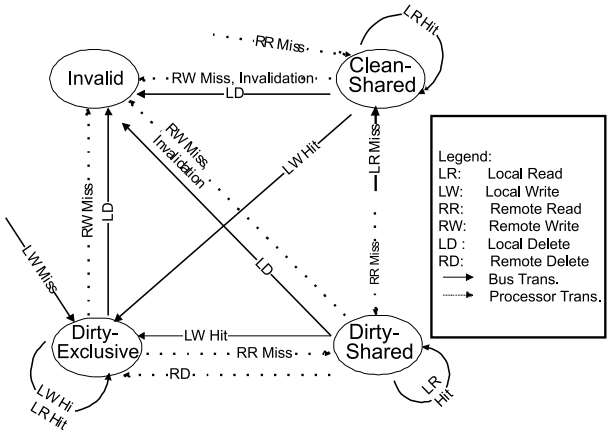


Fig. 4. Transition diagram of the Berkeley protocol.

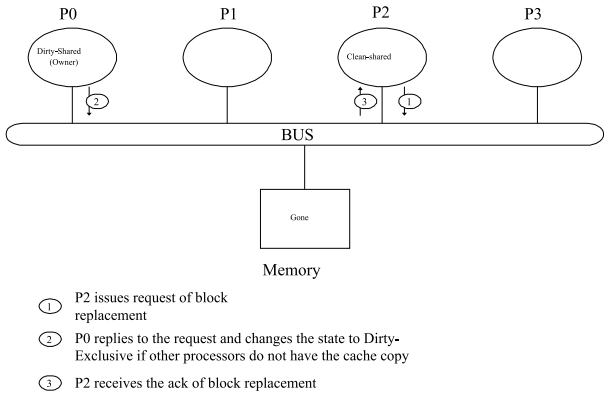


Fig. 5. Block replacement of Clean-Shared state in the Berkeley protocol.

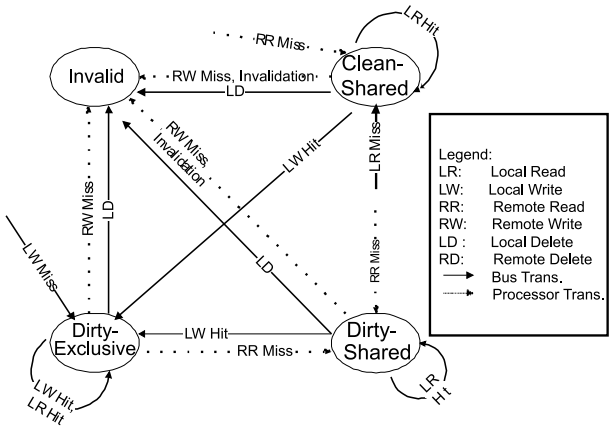


Fig. 6. The effective block replacement in the Berkeley protocol.

copy, the cache state of P0 changes to Dirty-Exclusive instantaneously. P2 will finally receive an acknowledgement that replacement is successful. If the target block is Dirty-Exclusive, it can be written immediately. Table 1 collects the total number of write requests from different states. The Dirty-Exclusive state is further divided into Pure Dirty-Exclusive and Impure Dirty-Exclusive. A state is set to Impure Dirty-Exclusive if the Dirty Exclusive state is originally in the Dirty-Shared state; otherwise it is the Pure Dirty-Exclusive state. As illustrated in Table 1, changes from Dirty-Shared to Dirty-Exclusive are far more common than write requests from Dirty-Exclusive states. As a result of the Effective Replacement Scheme, it will definitely eliminate the overhead of changing Dirty-Shared to Dirty-Exclusive states, as shown later.

In the Berkeley protocol, if some blocks of the Clean-Shared state are to be replaced, the cache controller must issue a block replacement request. The situation is identical in our replacement policy. Hence, replacement transactions of a Clean-Shared block are the same as in the previous discussion. However, our evaluation, based on the SPLASH benchmarks, shows that total number of write requests from the Dirty-Exclusive state is less than the number of block replacements from the Clean-Shared state as shown in Table 1, so that it is unnecessary to change Dirty-Shared to Dirty-Exclusive. Fig. 6 shows the new transition diagram of the Berkeley protocol.

3.4. Mechanism to reduce migratory-sharing access

Gupta and Weber (1992) classified data structures based on the invalidation patterns they exhibit. According to their definition, migratory data structures are manipulated by many processors, but only by a single processor at any given time. In parallel programs, data structures are modified within a critical section, and high-level language statements such as $I := I + 1$ exhibit migratory patterns. Because the modification of migratory objects usually executes the tight Read-Modify-Write operation, Dahlgren and Stenstrom (1995) have formally defined the reference pattern of migratory blocks using the following regular expression

$$\ldots \boldsymbol{R}_i(R_i)^*(\boldsymbol{W}_i)(R_i/W_i)^*(\boldsymbol{R}_j)(R_j)^*(\boldsymbol{W}_j)(R_j/W_j)^* \ldots, \quad (1)$$

where $\boldsymbol{R}_i$ and $\boldsymbol{W}_i$ represent a read reference and a write reference, respectively, by processor i; '*' denotes zero or more occurrences of the preceding string; and '|' denotes the logical OR-operation. By (1), the corresponding access action of migratory data must be that there is at least one R_i followed by at least one W_i by the same processor, i, before the next processor, j, starts accessing the block in the same way. Different mechanisms use the same idea to detect the migratory-sharing reference patterns. After the migratory-sharing block has been detected, the subsequent read reference to the block is replaced by a read-exclusive one to prevent the need of

Table 1
Total number of write requests from different states

Benchmark	Write from pure Dirty-Exclusive	Write from impure Dirty-Exclusive	Write from Dirty-Shared	Delete from Clean-Shared
MP3D	2.8 (M)	0	13 760	676
FFT	5.6 (M)	624	13 358	118 906
Ocean	14.2 (M)	42 432	603 722	107 581
PTHOR	643 344	108	12 900	209 274
Water	2.5 (M)	0	3093	3093

invalidation or update messages incurred by the subsequent write references.

We have implemented both the software and hardware approaches to handle migratory accesses on our system with linked-based cache coherence protocol. Our software schemes (Pean et al., 1998b) extend our previously presented mechanism (Su et al., 1996) to clustering-based systems. The extended method is proposed to reduce the overhead of migratory-sharing references for linked-based cache coherence protocols. As migratory-sharing references have a special access pattern, we can use read-exclusive access to avoid the necessity of subsequent explicit invalidation requests. The read penalty is thus reduced. To keep the detection mechanism simple, we identify the migratory-sharing memory blocks in run time through labels specified at compile time. Such a detection mechanism can be scalable to various architectures and protocols. Therefore, the software scheme combines both compiler labelling and run-time detection techniques using the following four steps.

(i) *Categorization and labelling*. This is done at compile time with compiler aided. Memory access is categorized in Fig. 7.

(ii) *Maintenance*. New cache and memory states are added to maintain the migratory-sharing blocks.

(iii) *Detection*. By using the information of labelling and categorization, the memory subsystem is able to detect the access pattern easily.

(iv) *Handling*. The labelled migratory-sharing access is handled as read-exclusive.

On the other hand, for the hardware approach, a block is classified as migratory if the following two conditions are satisfied.

(i) The processor that issues the write access request is not the same as the processor that most recently issued a write access request to the block.

(ii) The number of block copies is exactly two.

The hardware approach involves only two steps: detection and handling. A memory controller has been added with two new states to identify migratory-sharing blocks. A detailed description can be found in (Pean et al., 1998b).

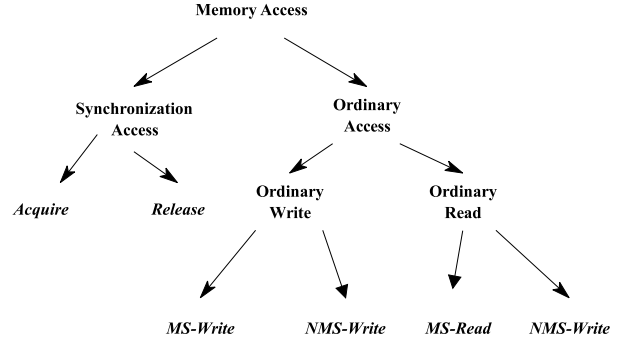


Fig. 7. Categorization of shared references for the software method.

The advantage of our hardware mechanism is that it can properly detect migratory accesses. The cost of the hardware scheme is only two extra memory states but without any extra cache state. On the other hand, our mechanism can also reduce the migratory-sharing overhead by degrading the state from the migratory to the shared mode. With extra cache bits, our scheme can avoid interference of non-migratory words named as false sharing in the migratory blocks. Meanwhile, our scheme induces less hardware cost than that implemented in the centralized directory protocols.

3.5. Two-level cache with write cache

Processors simulated on SEECMA are blocked on read misses (Dahlgren and Stenstrom, 1995). The PE with WC is shown in Fig. 8. Read accesses are handled as follows. When a read reference hits the FLC, it returns the requested words to the processor. When a read miss occurs on the FLC, the FLWB is examined to determine whether the requested block is buffered. If such is the case, the read miss request cannot be sent to the SLC until the entire write references buffered in the FLWB have been issued to the SLC. This handling procedure enables the FLC to support single-cycle access, simply and quickly. If the requested block is not in the FLWB, the read miss is sent directly to the SLC. The requested words are returned either from the SLWB, if they are found there; or from the SLC if they hit the SLC. Otherwise, a global action will take place. Replacement prior to the global action may be necessary.

Write accesses are handled as below. A write request from a processor is blocked when the FLWB is full. Once the FLWB has a free entry, the write request is pushed onto the FLWB. Meanwhile the word contents of the FLC are updated if the write request hits the FLC. Since the FLWB is a FIFO queue, the write request cannot be issued to SLC until all of the prior write accesses have been forwarded to the SLC. Whenever a write reference accesses the SLC, it will issue a global memory action (e.g., write-invalidate request) and buffer itself in the SLWB. When the local bus is available, the global action will proceed. Here, we assume that the default access times of the FLC and the SLC are one and three processor cycle times, respectively (Dahlgren and Stenstrom, 1995). However, the penalty is not constant, and depends on some other factors, such as

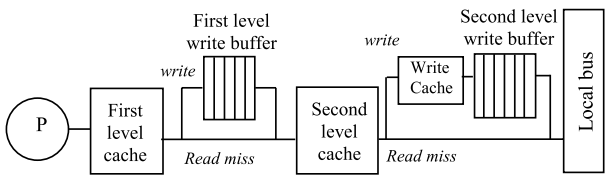


Fig. 8. Processor environment with a write cache.

state of cache block, location of memory copy, interconnection network routing time, and so forth.

The WC (Dahlgren and Stenstrom, 1995) serves as an optional simulation component. Whenever a write reference accesses the SLC, it will issue a global memory action and the transaction will be buffered on the WC instead of the SLWB. Transactions belonging to the same cache block will be merged into a single entry in the WC. Transactions in the WC will be forwarded to the SLWB when block replacement occurs on the WC.

Memory access ordering requirements must be enforced by the underlying memory consistencies model. Sequential and processor consistency require that, before any processor can perform a write access, all of the previous program order write accesses must have been performed. Consequently, WCs are not supported in these two models. For the weak consistency model, the WC can only be flushed upon arrival of a synchronization reference, and the WC can only be flushed upon the arrival of a release reference in the released consistency model.

3.6. Cache coherence protocols and directory structure

Fully mapped centralized, limited centralized and distributed directories (Stenstrom, 1990) are the three cache directory structures supported in our system. The centralized directory-based cache coherence protocols are similar to those described by Dahlgren and Stenstrom (1995). However, the distributed directory-based protocol is an extension of the IEEE SCI standard (Su, 1996). The four cache coherence protocols are write-invalidate, write-update, clean and competitive-update (Dahlgren and Stenstrom, 1995).

4. Overview of SEECMA

4.1. Basic Structure of SEECMA

SEECMA is programmed in C and implemented on a SUN workstation running under UNIX System V. To facilitate future extensions, the whole program is developed in a modular structure. The front-end of SEECMA is a memory reference generator supported by MINT, and the back-end is a memory subsystem simulator. As a general tool, SEECMA allows the user to specify and simulate his/her own designs by linking in his/her specific modules. Fig. 9 gives an overview of SEECMA.

The memory reference generator consists of a simulation controller and processor simulator. The simulation controller monitors the execution of the simulation environment, provides the functions of debugging, monitoring, event generation and management, task management and scheduling, timing control and thread management. The processor simulator simulates instruction interpretation and execution. The memory subsystem simulator includes a node simulator and a global interconnection simulator. The node simulator is responsible for the simulation of the two-level cache hierarchy, the doubly linked directory cache coherence protocols, the memory consistency models, local interconnection, the replacement policy and inter-cluster cache prefetching. The global interconnection simulator controls simulation on the interconnection network. Detailed simulation functions are presented in Fig. 10, and each of them is stated in the corresponding architectural component. The simulator's primary functions are:

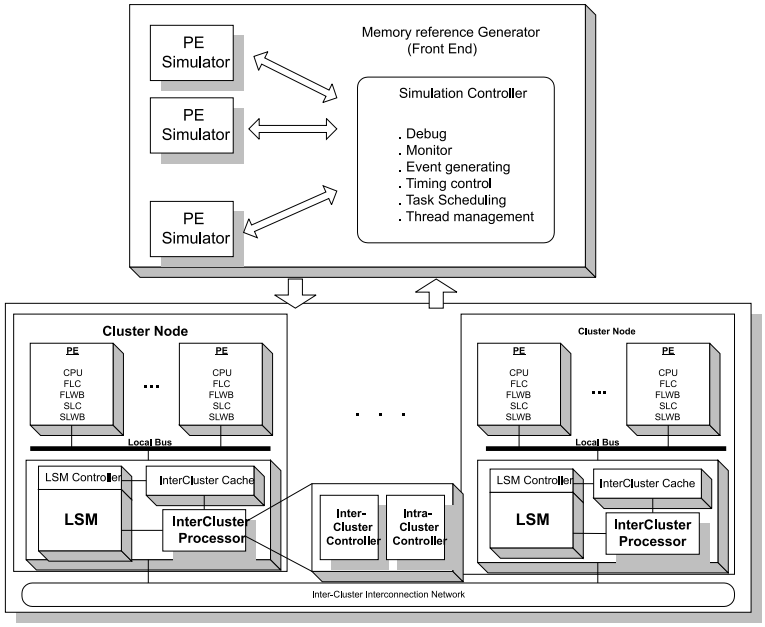


Fig. 9. Overview of SEECMA.

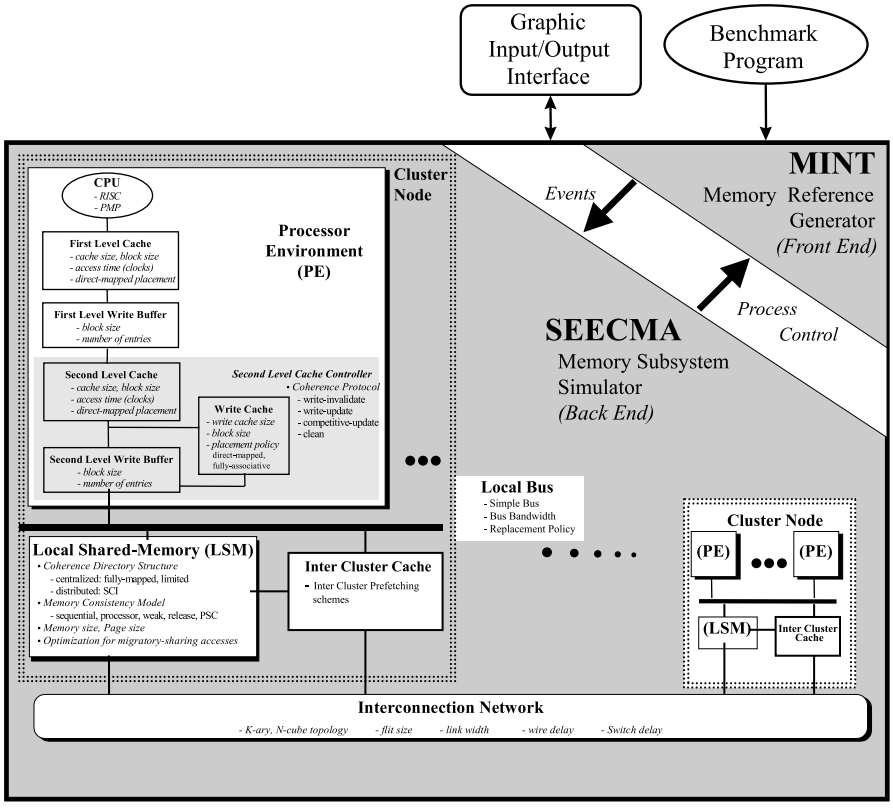


Fig. 10. The complete structure of SEECMA.

(i) two CPU types – Reduced Instruction Set Computer (RISC) and Parallel Multithreaded Processor (PMP) (Hirata et al., 1992; Wu, 1998);
(ii) four memory consistency models sequential (Lamport, 1979), processor (Goodman, 1989), weak (Dubois et al., 1986), and release (Gharachorloo et al., 1990);
(iii) two-level cache hierarchy with FLWB, SLWB, and WC (Dahlgren and Stenstrom, 1995);
(iv) three cache directory structures – centralized fully mapped, centralized limited (Stenstrom, 1990), and distributed SCI (Scalable Coherence Interface) structures (Gjessing et al., 1991);
(v) five cache coherence protocols – SCI, write-invalidate, write-update, competitive-update, and clean (Grahn et al., 1995);
(vi) *k*-ary, *n*-cube interconnection network (Dally, 1990);
(vii) optimization for migratory-sharing accesses by software and hardware mechanisms (Su et al., 1996; Pean et al., 1998b); and
(viii) effective replacement and prefetching schemes (Pean et al., 1998a).

The designer can use any combination of these to simulate and evaluate his/her target machine.

After the back-end is constructed, the whole system is linked on a SUN workstation as shown in Fig. 11. Following success in linking both the memory reference generator and libraries we obtain an executable simulator. The target benchmark programs must be statically linked Irix-executable files, specially designed for the MIPS R3000 processor. At run time, the simulation options of interest ought to be specified completely, otherwise default values will be used.

SEECMA is capable of providing the following considerable evaluation information:

(i) parallel execution time, including busy time, stalled time for read miss, acquire-stall time, stalled time for write-buffer-full, and contention time for accessing FLC;
(ii) memory accesses for FLC and SLC, including number of read and write requests;
(iii) miss ratios for read and write requests;
(iv) cold and coherence misses;
(v) distribution of invalidation/update count;
(vi) write run distribution;
(vii) hit ratios in WC;
(viii) write run distribution in WC;
(ix) number of read and write misses to invalid memory copies;
(x) amount of network traffic;
(xi) number of LOCK and Barrier operations; and
(xii) delayed time for acquire and release accesses.

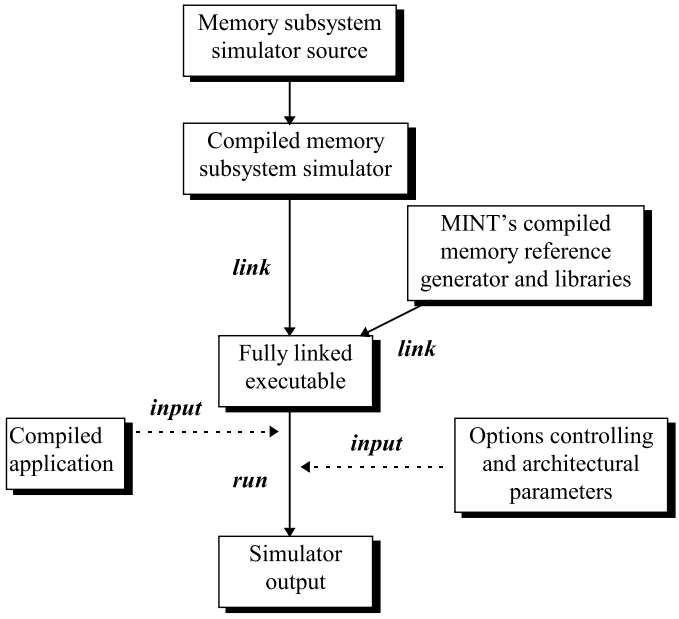


Fig. 11. Construction of MINT-based simulator.

4.2. Simulation correctness

Correctness is a key issue for any simulator development. For small, or even medium input domains, correctness verification is often not a difficult task. However, for large input domains it is much more difficult, and perhaps impossible. However, we have applied the system proposed in Pong (1995) to verify cache coherence protocols in our system. The SSM system is based on a symbolic state model, which exploits the symmetry and homogeneity of cache protocols to reduce the size of the state space. SSM has the general advantage of verifying cache protocols independently of model size. We need to adapt the method for our clustering architecture because it is difficult to abstract the linked list (Pong et al., 1995).

We have also compared the execution result obtained from SEECMA with reported data. For instance, Dahlgren and Stenstrom have studied the impact of WCs on several cache coherence protocols for shared-memory multiprocessors (Dahlgren and Stenstrom, 1995), and we have adopted this evaluation for our PMP-MP (processor-based multiprocessor) system. The result on the PMP-MP system with one thread per processing element is identical to the result reported by Dahlgren and Stenstrom. However, an exception was found on the Ocean benchmark. This is due to Dahlgren and Stenstrom using the benchmark from the SPLASH suite (Singh et al., 1992), while ours is from the SPLASH-2 suite (Woo et al., 1995).

Our architecture provides two processor types: RISC (Culler et al., 1999) and PMPs (Hirata et al., 1992). PMPs allow multiple threads to be executed simultaneously, and parallel running threads share a single cache in each PE. It therefore shows superior utilization of resources. Users can choose either processor architecture based on their research interests (Wu, 1998). The memory subsystem supported in SEECMA is a cache-coherent non-uniform architecture (CC-NUMA) (Culler et al., 1999). It comprises a variety of important design issues to be explored and then evaluated.

4.3. Graphic interface capabilities

SEECMA supports a user-friendly GUI that enables users to operate in the simulation and evaluation environment easily and efficiently. The GUI consists of two portions: input and output interfaces.

As we enter the simulation environment, we have the window shown in Fig. 12. There is a menu bar, architecture graph and status region ready to communicate with the user. Menu bars collect five major selection items; each item carries a menu hierarchy based on functionality attributes. The architecture graph shows the corresponding architecture outlook, and it is updated according to the architectural parameter setting. For instance, if we incorporate a WC into the system, there will be a connection between WC and the cache hierarchy. At the bottom left of the window, there is a status region that summarizes the characteristics of the architecture component at the cursor position. In Fig. 12, for example, the cursor is currently located at the cluster node; the status region therefore lists all of the related information. The status region is helpful for double-checking and reconfirming the parameters.

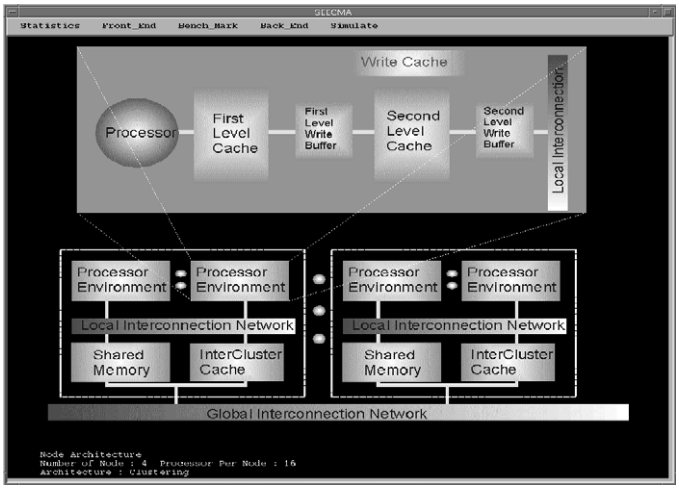


Fig. 12. Graphical user interface of SEECMA.

The two input interfaces are menu-based and graphic-based. With menu-based input, the selection items are collated as a pull-down menu and the dialogue window of each item pops up when the mouse is clicked. With the graphic-based input interface, when users click on any architecture block, they will be directed to a dialogue window without going through the pull-down menu.

Each simulation outcome is either shown in text mode or directed to an output file. Normally, we are interesting in several different comparison issues, parameter types or quantity adjustments. SEECMA provides a robust output function that is capable of collecting numerical results from different runs, and presents the final result as a bar or line chart according to the user's specification.

5. Preliminary performance evaluations of our architecture on SEECMA

In this section, we give a complete illustration of how to use SEECMA to evaluate several design issues of cluster-based multiprocessor systems, including ICP, effective replacement schemes and migratory sharing with both software and hardware approaches. Some reasonable assumptions about the evaluation environment are summarized in Table 2. The memory page size is 4 Kbytes and is mapped to the local memories in a round-robin fashion.

Some benchmarks were chosen from SPLASH for experimental evaluation and others from the SPLASH-2 suite (Singh et al., 1992; Woo et al., 1995). These programs are written in C using the Argonne National Laboratory (ANL) macros (Boyle et al., 1987) and compiled using CC under IRIS version 3, optimization level 2, on a SGI workstation. Table 3 lists a series of selected benchmark programs, together with their brief description and data sets. Note that only the parallel portions of the benchmarks are used here, and therefore only the statistical graphs show this result. The performance metric is execution time. And the execution time is further broken into five sections: busy time, read-stall time (i.e., time consumed for servicing cache misses), acquire-stall time (i.e., time spent on waiting locks to be acquired), contention time (i.e., waiting time to access FLC) and buffer-stall time (i.e., processor stall time due to FLWB full).

Table 2
Architecture parameters

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

5.1. Scalability of our architecture

The scalability of our architecture is shown in Fig. 13. The related performance data of non-cluster and DASH systems can be obtained by selecting appropriate simulation parameters in the SEECMA simulation and evaluation environment. In the water benchmark, our architecture always scales better than the non-cluster multiprocessor environment. Our architecture also scales better than the DASH cluster system because our architecture has speedup 41 while DASH has speedup 37 when the number of processors is 48. In the MP3D benchmark, our architecture scales better than

Table 3
Benchmark programs

Benchmark	Description	Data sets
MP3D	Particle-based wind-tunnel simulator	5 K particles, 10 time steps
Ocean	Simulate eddy currents in an ocean basin	128 by 128 grid, tolerance 10^{-7}
FFT	Blocked 1D FFT	64 K complex poins
Water	Water molecule dynamics simulation	343 molecules
Barnes	N-body gravitation simulation	8192 bodies, 6 steps
Radix	Integer Radix sort algorithm	1 M 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

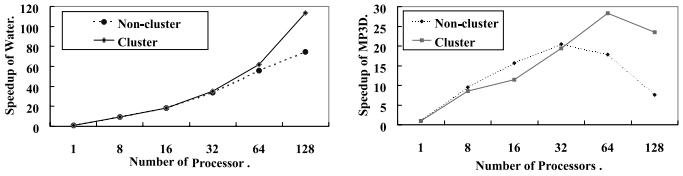


Fig. 13. Scalability of our architecture compared with non-cluster multiprocessor.

non-cluster multiprocessor environments when the number of processors is greater than 40. Our architecture also scales better than the DASH cluster system (Leno et. al., 1989) because our architecture has speedup 20 while DASH has speedup 7 when the number of processors is 48.

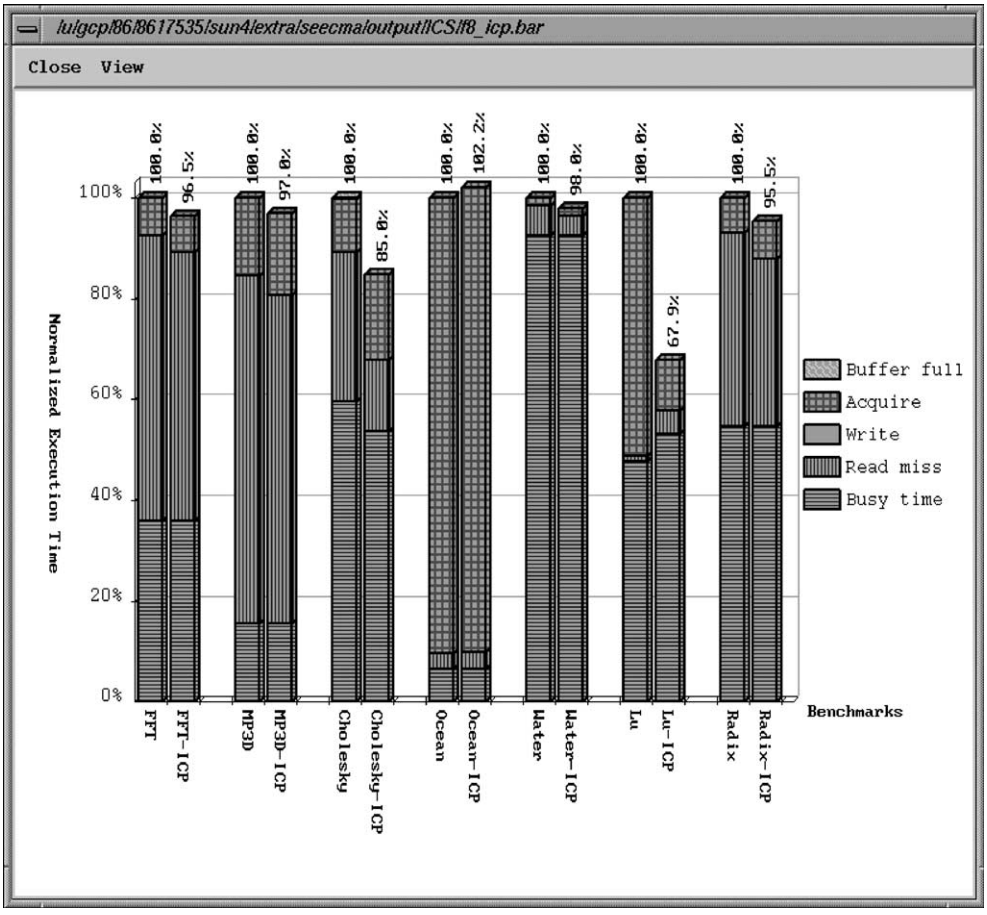


Fig. 14. Performance evaluation of the ICP scheme.

5.2. Performance evaluation of ICP

Fig. 14 shows the execution time of several benchmark programs running with and without the ICP scheme. We find that the performance gain of having ICP is about 3–33%, as a result of the traffic overhead reduction, as long as prefetching offers a sharp drop in read miss counts, and thus decreases the miss stall time. Among them, LU is the most outstanding, since it takes full advantage of reducing inter-cluster traffic overhead. The acquire-stall time of the LU with ICP is far less than LU without ICP. Meanwhile Ocean is an exceptional case, ICP causing execution time to become worse than the original, due to serious false sharing miss. In summary, having an inter-cluster traffic overhead penalty and the number of read misses sharply been reduced, the ICP scheme is superior to the conventional hardware prefetching method.

5.3. Performance evaluation of effective block replacement scheme

The effective block replacement scheme improves total execution time substantially, by up to 25% as shown in Fig. 15, when FLC size is 1 K and SLC size is 8 K. The major performance gain comes from both read stall time and acquire-stall time. Furthermore, when the released memory consistency model fails to hide all of the write stall time, the effective block replacement scheme will further improve the final performance. Table 4 illustrates the percentage of read stall time against total execution time, and number of Clean-Shared block replacement counts. Compared to Fig. 15, the higher the read stall time and replacement counts on Clean-Shared block, the greater the performance improvement. Radix, Barnes, FFT, Pthor, Cholesky and Ocean are cited to show this.

To verify the cost effectiveness of the replacement policy, we varied the FLC size from 8K to 1K, and the SLC size was decreased from 256K to 8K. Figs. 16(a) and (b) shows that, if the replacement scheme is not implemented, total execution time increases substantially as the cache size decreases. Thus, the effective replacement scheme has not only reduced the execution time, but also reduced the cost for the cluster-based multiprocessor system.

5.4. Performance evaluation on optimization of migratory-sharing data

In the software scheme, since program codes that are protected by the critical sections but which do not

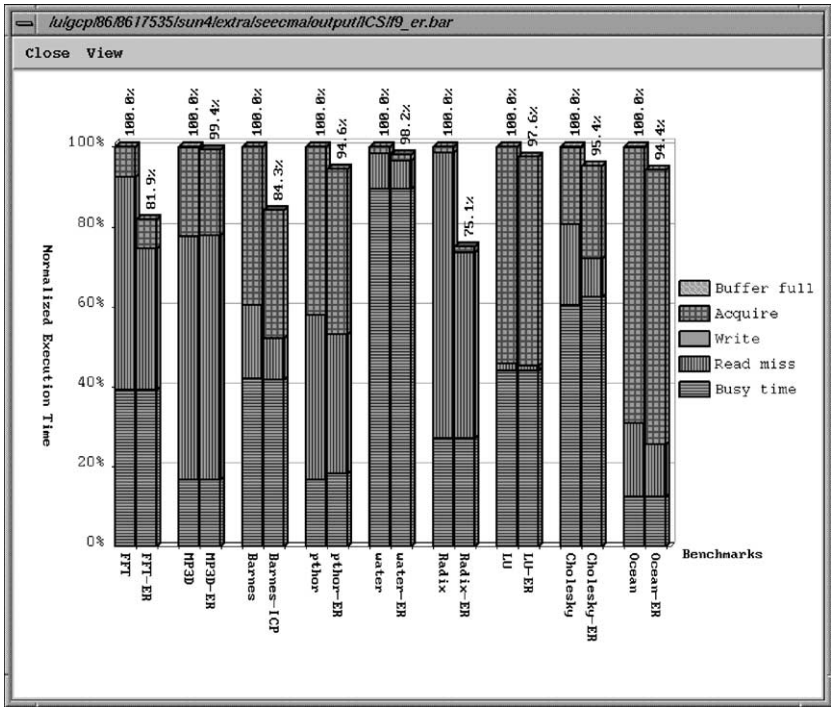
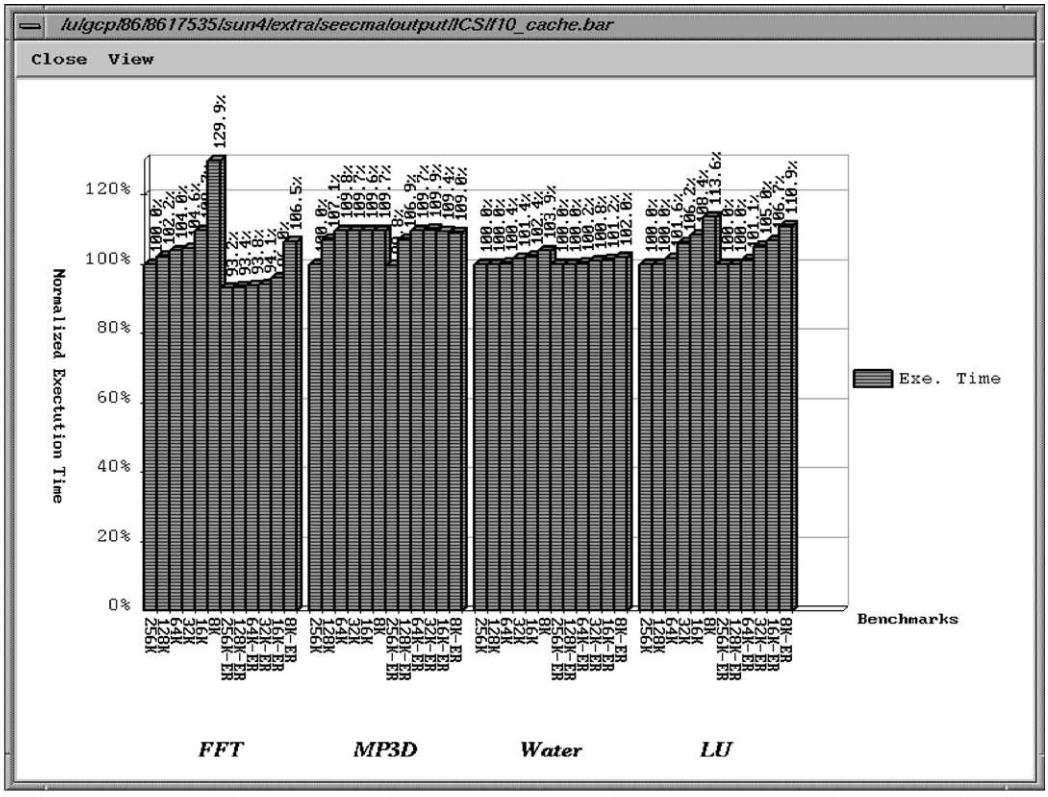


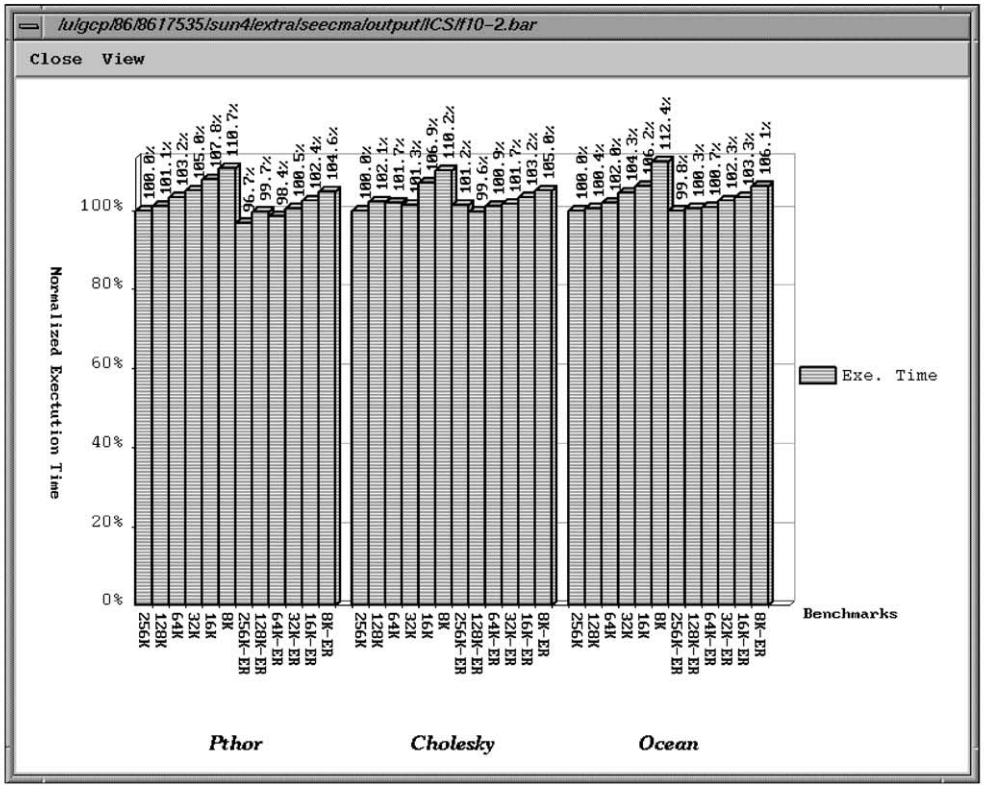
Fig. 15. Performance evaluation of effective block replacement mechanism.

Table 4
Percentage of read stall time compared to total execution time

	FFT	MP3D	Barnes	Pthor	Water	Radix	Lu	Cholesky	Ocean
Read stall time %	53.2	60.9	18.4	41.2	8.8	71.2	1.9	20.2	18.4
Replacement #	335 K	13 K	1.2 M	1.7 M	0.78 M	2.5 M	44 K	0.36 M	2.7 M



(a)



(b)

Fig. 16. (a) and (b) Total execution time without and with the effective block replacement scheme on different cache sizes.

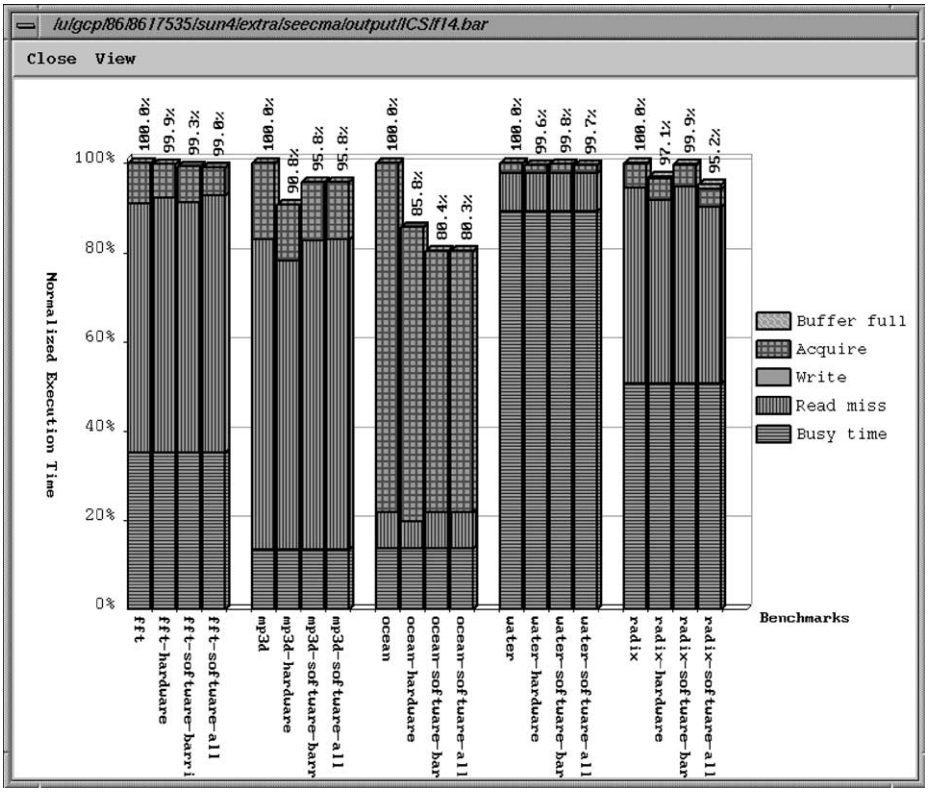


Fig. 17. Performance evaluation of hardware and software migratory optimisation schemes.

Table 5
The statistics and characteristics of references for the benchmark programs

Benchmark	Shared reads (M)	Shared writes (M)	*R*/*W* ratio	MS-access in locks	MS-access in barriers
MP3D	5.11	3.40	1.5	19 168	11 400
Ocean	64.99	15.22	4.27	57 572	354 920
FFT	6.85	5.74	1.19	256	2660
Water	37.23	2.78	13.39	117 735	4940
Barnes	7.8	0.26	30	169 193	6460
Radix	11.5	6.5	1.77	426 240	3040

belong to Barrier codes are typically large, many non-migratory-sharing memory blocks will be detected as migratory blocks. Due to incorrect labelling and the Read-and-Self-invalidate policy for NMS-Reads, the read penalty becomes larger and degrades system performance. The effect is obvious for MP3D, but is slight in FFT and Water, as shown in Fig. 17. However, Ocean is an exception because it has a lower percentage of critical sections for implementing non-Barrier codes, as depicted in Table 5. On the other hand, software mechanisms with all critical sections perform better than those with only barrier sections, particularly in the Barnes and Radix benchmarks. The scheme increases the system performance by up to 7.5- and 3.2-fold in the Barnes and Radix benchmarks, respectively, compared to the barrier code mechanism. Hence, the software schemes can always improve system performance in terms of execution time, for all benchmarks.

If the hardware mechanism is implemented, a dynamic detection scheme will solve the problem of incorrect labelling, as shown in Fig. 17. For FFT and MP3D, the read miss penalty with the hardware

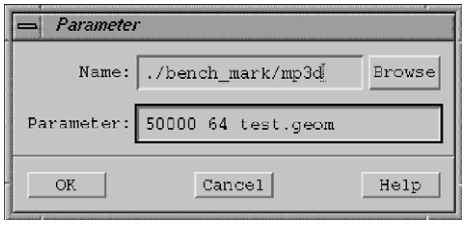


Fig. 18. Input name and parameters of the benchmark program.

Fig. 19. Input name and parameters of the benchmark program using browsing.

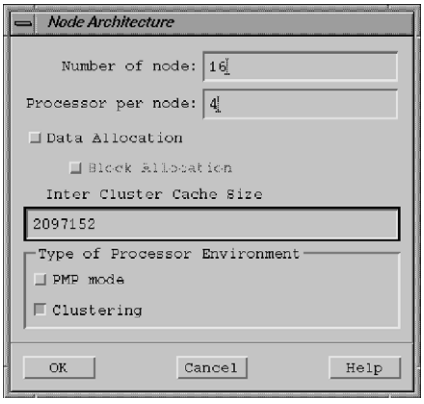


Fig. 20. Input parameters of a node architecture.

scheme is always less than for that with software schemes. These benchmarks also perform better with the hardware scheme than with the software schemes. However, migratory-sharing blocks may contain migratory and non-migratory words in the same block. The software mechanism uses a non-cache method to avoid the interference overhead of these non-migratory-sharing accesses, caused by false sharing. As illustrated in Fig. 17, conflicting accesses of false sharing make the software mechanism more efficient than the hardware mechanism for the Ocean, Barnes and Radix benchmarks.

In summary, both the software and hardware schemes can substantially improve the performance of the system, by up to 20% and 15%, respectively. The performance of the system can be improved even more if the architecture of the system has more coherence delays.

6. Conclusion and future work

System designers usually rely on simulation to verify their conceptual designs and to understand the interaction among the system components. We have constructed a simulation and evaluation environment called SEECMA for research and education, derived from cluster-based multiprocessor systems. It provides versatile simulation functions in an integrated environment with a user-friendly graphical interface. The primary simulation options include:

(1) two CPU types – RISC and PMP architecture;
(2) cluster-based multiprocessors with arbitrary number of cluster nodes and processors per node;
(3) *k*-ary *n*-cube interconnection networks;
(4) inter-cluster prefetching;
(5) an effective block replacement scheme;
(6) migratory sharing with software and hardware approaches;
(7) four memory consistency models – sequential, processor, weak, and released models;
(8) a two-level cache hierarchy with FLWB, SLWB and WCs;

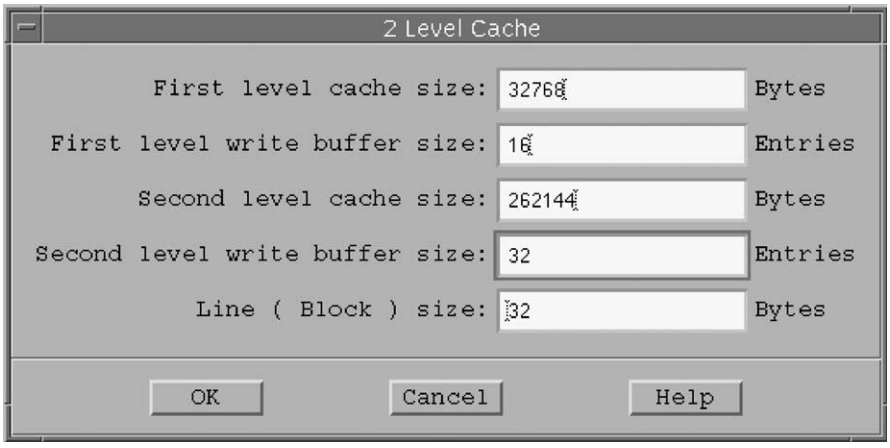


Fig. 21. Input the two-level cache and write buffer sizes.

(9) four cache coherence protocols – write-invalidate, write-update, competitive-update and clean protocol; and
(10) three cache directory structure – scentralized fully mapped, centralized limited, and distributed SCI structures.

We can specify the target system architecture through a menu-based or a graph-based input interface. The simulation results are dumped to a file. Optionally, we can set up the simulation parameters to view the specified evaluation results through statistical graphs. SEECMA also allows users to input their own designs by linking in specific modules and then evaluating them. In conclusion, SEECMA serves as a convenient and friendly research environment for those who are interested in system designs for cluster-based multiprocessor architectures.

In the future, we will continue to explore several new design issues to improve the total performance of our architecture, such as more relaxed memory consistency models or software optimization mechanisms for parallelization. Subsequently, related enhanced mechanisms will also be developed to enhance these architectures. We can thus provide a more powerful simulation platform of multiprocessor architecture design for research and educational purposes.

Acknowledgements

This research was supported by the National Science Council of the Republic of China under contract number NSC 87-2213-E009-049.

Appendix A. An illustrative example of manipulating SEECMA

This is a guide for how to use the GUI to specify the simulation parameters and output statistics graph on SEECMA. The Adhere, ICP scheme is examined, and our target architecture parameters are shown in Table 3. We assume that users are familiar with the window as shown in Fig. 12.

Step 1. Click the ''Bench_Mark'' button on the menu bar.

Step 2. Key in the name and parameters of the benchmark program from the window as shown in Fig. 18, or by using the ''Browse'' function via the window as in Fig. 19. In this example, we choose MP3D as our benchmark program.

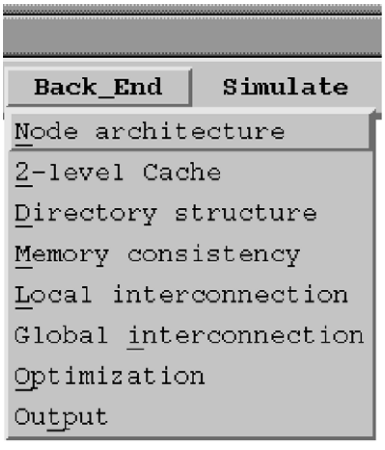


Fig. 22. Menu of ''Back_End'' item.

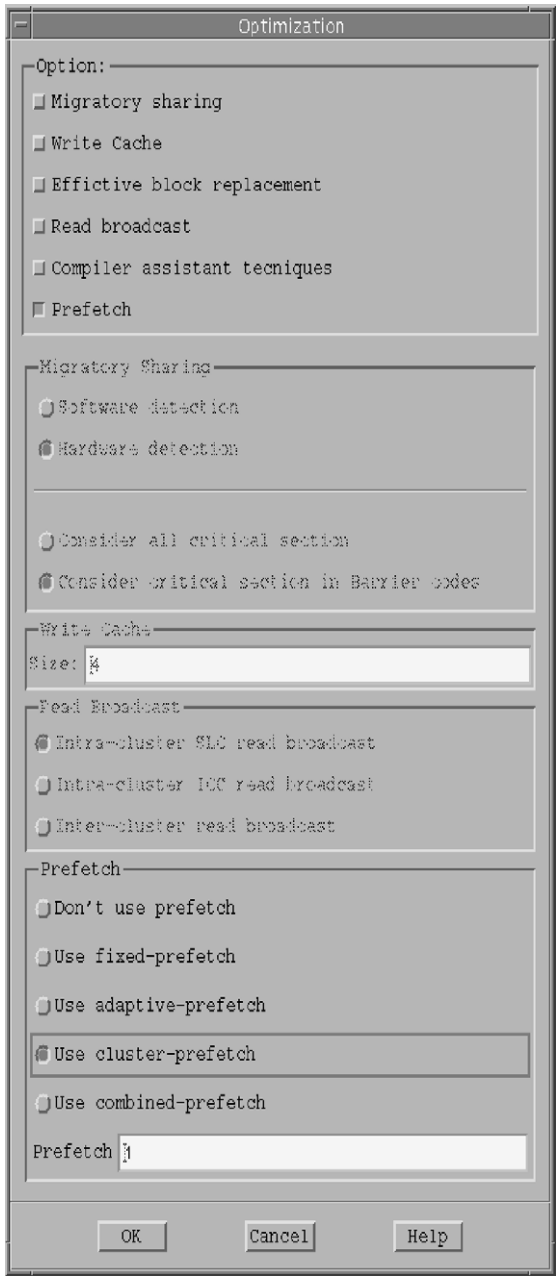


Fig. 23. Optimization options.

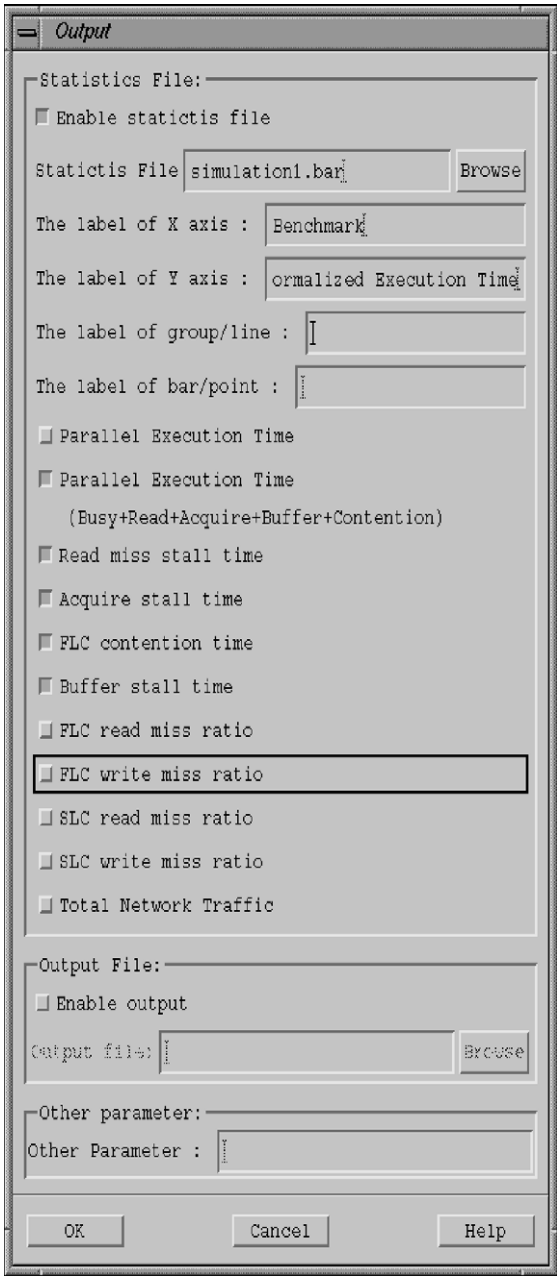


Fig. 24. Input parameter of statistical graph.

Step 3. Position the mouse on cluster node and click once; the pop-up window is shown in Fig. 20. We choose clustering architecture with four nodes and 16 processors per node. At the same time, we are required to provide appropriate inter-cluster cache size.

Step 4. Click ''First-Level Cache'' and input the two-level cache and write buffer size as shown in Fig. 21.

Step 5. Click the ''Back_End'' item from the menu bar, and select ''Optimization'' from the pull-down menu, as shown in Fig. 22. When you find the window shown in Fig. 23, click the ''Prefetch'' option, and then ''Use cluster-prefetch''.

Step 6. Working with the ''Back_End'' item again, this time we specify the ''Output'' item, as shown in Fig. 24. The parameters are set according to the expected statistical graph. In this example, we would like to create the graph shown in Fig. 12, and therefore the parameters must be consistent for each benchmark program.

Parameters that are not specified above are given a default value; for example, memory consistency model is the Released Consistency Model.

References

Boyle, J., Bulter, R., Disz, R., Glickfeld, B., Luck, E., Overbeek, R., Patterson, J., Stevens, R., 1987. Portable Programs for Parallel Processors. Holt, Rinehart & Winston, New York, USA.

CONVEX, 1994. Computer Corporation, CONVEX Exemplar Architecture, 2nd ed. CONVEX Press, Texas, USA.

Culler, D.E., Jaswinder, P.S., Gupta, A., 1999. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann, Los Altos.

Dahlgren, F., Stenstrom, P., 1995. Using write caches to improve performance of cache coherence protocols in shared-memory multiprocessors. Journal of Parallel and Distributed Computing 2 (26), 193–210.

Dally, W.J., 1990. Performance analysis of *k*-ary *n*-cube interconnection networks. IEEE Transactions on Computers 39 (6), 775–785.

David, B.G., 1995. 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, Stanford, California, USA.

Dubois, M., Scheurich, C., Briggs, F., 1986. Memory access buffering in multiprocessors. In: Proceedings of the 13th Annual International Symposium on Computer Architecture. Tokyo, Japan, pp. 434–442.

Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A., Hennessy, J., 1990. Memory Consistency and event ordering in scalable shared-memory multiprocessors. In: Proceedings of the 17th Annual International Symposium on Computer Architectures. Seattle, WA, USA.

Gjessing, S., Gustavson, D.B., Goodman, J.R., James, D.V., Kristiansen, E.H., 1991. The SCI cache coherence protocol. In: Dubois, M., Thakkar, S. (Eds.), Scalable Shared Memory ultiprocessor. Kluwer Academic Publishers, Norwell, MA, USA, pp. 1991.

Goodman, J.R., 1989. Cache consistency and sequential consistency. Computer Sciences Technical Report #1006, Computer Sciences Department, University of Wisconsin, Madison, WI, USA.

Grahn, H., Stenstrom, P., Dubois, M., 1995. Implementation and evaluation of update-based cache protocols under relaxed memory consistency models. Future Generation Computer Systems 11 (3), 247–271.

Gupta, A., Weber, W.D., 1992. Cache invalidation patterns in shared-memory multiprocessors. IEEE Transaction on Computers 41 (7), 794–810.

Hirata, H., Kimura, K., Nagamine, S., Mochizuki, Y., 1992. An elementary processor architecture with simultaneous instruction issuing from multiple threads. In: Proceedings of the 19th International Symposium on Computer Architecture. Gold Coast, Qld., Australia, pp. 136–145.

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

Lamport, L., 1979. How to make a multiprocessor computer that correctly executes multiprocessor programs. IEEE Transactions on Computers C 28 (9), 241–248.

Lenoski, D., Laudon, J., 1989. Stanford DASH multiprocessor. Technical Report No. CS-TR-89-403.

Mazin, S., Yousif, M.J., Thazhuthaveetil, Das C.R., 1995. Cache coherence in multiprocessors: a survey. Advances in Computers, 40.

Pean, D.L., Huang, H.W., Wu, J.R., Chen, C., 1998a. Effective prefetching and replacement policies in the scalable clustering-based multiprocessor system design. In: Proceedings of 1998 International Computer Symposium, Tainan, Taiwan, ROC.

Pean, D.L., Wu, J.R., Chen, C., 1998b. Effective mechanism to reduce the overhead of migratory sharing for linked-based cache coherence protocols in clustering multiprocessor architecture. In: Proceedings of 1998 International Conference on Parallel and Distributed Systems. Tainan, Taiwan, ROC.

Pong, F., 1995. Symbolic state model: a new approach for the verification of cache coherence protocols. Ph.D. Dissertation. University of Southern California.

Pong F., Nowatzyk A., Aybay, G., Dubois, M., 1995. Verifying distributed directory-based cache coherence protocols: s3.mp, a case study. In: First International EURO-PAR Conference.

Stenstrom, P., 1990. A survey of cache coherence schemes for multiprocessors. IEEE Computer 23 (6), 12–24.

Singh, J.P., Weber, W.D., Gupta, A., 1992. SPLASH: Stanford parallel applications for shared-memory. Computer Architecture News 20 (1), 5–44.

Su, J.P., 1996. A study of memory subsystem design for multiprocessor system and implementation of its simulation and evaluation environment. Master Thesis, National Chiao Tung University, Hsinchu, Taiwan, ROC.

Su, J.P., Wu, C.C., Chen, C., 1996. Reducing the overhead of migratory-sharing access for the linked-based directory coherence protocols in shared-memory multiprocessor systems. In: Proceedings of International Computer Symposium. Taiwan, ROC, pp. 160–167.

Veenstra, J.E., Fowler, R.J., 1994. MINT Tutorial and User Manual. Technical Report No. 452, The University of Rochester, New York, USA.

Woo, S.C., Ohara, M., Rorrie, E., Singh, J.P. and Gupta, A., 1995. The SPLASH-2 programs: characterization and methodological considerations. In: Proceedings of the 22nd Annual International Symposium on Computer Architecture. Santa Margherita Ligure, Italy, pp. 24–36.

Wu, C.C., 1998a. PMP Memory consistency models. Ph.D. Dissertation, National Chiao Tung University, Hsinchu, Taiwan ROC.

Wu, C.C., Chen, C., 1998b. A new relaxed memory consistency model for shared-memory multiprocessors with parallel multithreaded processing elements. Journal of Information Science and Engineering, accepted.

Wu, C.C., Pean, D.L., Su, J.P., Wu, J.R., Huang, H.W., Huang, J.L., Lee, J.L., Chua, H.T., Chen, C., 1998c. SEESMA: a simulation and evaluation environment for shared-memory multiprocessor architecture. In: Proceedings of the National Science Council, Republic of China, Part A: Physical Science and Engineering, vol. 22(4). pp. 524–538.

Cheng Chen is a professor in the Department of Computer Science and Information Engineering at National Chiao Tung University, Taiwan, ROC. He received his B.S. degree from the Tatung Institute of Technology, Taiwan, ROC in 1969 and M.S. degree from the National Chiao Tung University, Taiwan, ROC in 1971, both in electrical engineering. Since 1972, he has been on the faculty of National Chiao Tung University, Taiwan, ROC. From 1980 to 1987, he was a visiting scholar at the University of Illinois at Urbana Champaign. During 1987 and 1988, he served as the chairman of the Department of Computer Science and Information Engineering at the National Chiao Tung University. From 1988 to 1989, he was a visiting scholar of the Carnegie Mellon University (CMU). Between 1990 and 1994, he served as the deputy director of the Microelectronics and Information Systems Research Center (MIRC) in National Chiao Tung University. His current research interests include computer architecture, parallel processing system design, parallelizing compiler techniques, and high performance video server design.

Der-Lin Pean is a Ph.D. candidate in Computer Science and Information Engineering at the National Chiao Tung University, Taiwan, ROC. He received his B.S. degree in Information and Computer Engineering at Chung Yuan Christian University, Taiwan, ROC. He served as a lecturer in the Department of Computer and Information Engineering, as well as Employment and Vocational Training Administration, Council of Labor Ministry, Taiwan, ROC. His current research interests include computer architecture, personal computer system architecture design, parallel processing system design, parallelizing compiler techniques, and microprocessor system design.

Chao-Chin Wu was born on 26 February 1968 in Taichung County, Taiwan, Republic of China. He received the B.S. degree in Computer Science and Engineering from Tatung Institute of Technology, Taiwan, in 1990, and the M.S. degree in Computer Science and Information Engineering from National Chiao Tung University, Taiwan, 1992. He received the Ph.D. degree in Electrical Engineering and Computer Science from National Chiao Tung University, Taiwan, 1998. His research interests include computer architecture and parallel processing.

Huey-Ting Chua received the B.S. and M.S. degrees in 1997 and 1999, respectively, both in Computer Science and Information Engineering from National Chiao Tung University, Taiwan. Her major research interest is parallel compiler.