Effective Techniques to Reduce Memory Access Latency for F-COMA Multiprocessor

Lan-Mao Chung, Der-Lin Pean, and Cheng Chen

Department of Computer Science and Information Engineering
1001 Ta Hsueh Road, Hsinchu, Taiwan, 30050, Republic of China
Tel: (8863) 5712121 EXT: 54744, Fax: (8863) 5724176

Abstract

The F-COMA system is designed to reduce memory access latencies of the NUMA systems while program working sets exceed the size of its cache. However, there are still many memory access overheads in it. We proposed several effective mechanisms to reduce these overheads. Two migratory sharing optimization mechanisms are presented to decrease the superfluous memory requests incurred by migratory accesses. We also use invalidation cache to omit the unnecessary AM accesses. On the other hand, we use cluster-base F-COMA to increase AM utilization and reduce memory access misses. Finally, we combine these methods to improve the performance of F-COMA. Based on our evaluation results, our combined methods speedup the total performance about 39% in average under SPLASH benchmarks.

1. Introduction

Scalable shared-memory multiprocessors are emerging as attractive platforms for application with high performance demands [1]. Recently, the main architectures in shared-memory multiprocessor are cache-coherence non-uniform memory access (CC-NUMA) [2] and cache only memory architecture (COMA) [3]. However, if the quantity of data that a processor is actively accessing exceeds the size of its cache in CC-NUMA, data loaded into the cache will be displaced and sent back to main memory before it is accessed again. In this case, the cache become ineffective and system performance will be decreased. The disadvantage of COMA is large remote memory access penalty due to the hierarchical directory structure and network. In order to improve both disadvantage of CC-NUMA and COMA, the F-COMA architecture was proposed [4].

Nevertheless, there are still many memory access overheads in the F-COMA architecture. At first, migratory sharing in the F-COMA architecture will induce unnecessary remote memory access request and thus decrease the performance of F-COMA architecture [5]. Second, there are unnecessary AM (Attraction Memory) accesses in the F-COMA architecture [6]. Then, AM utilization will influence overall system performance in some degree. In this paper, we will explore some design issues and propose effective mechanisms to reduce the above overhead. At first, we design migratory sharing optimization mechanisms named MOR (Migratory sharing Optimization and Return to exclusive) and EMOR (Enhance Migratory sharing Optimization and Return to exclusive) to reduce the accesses request. Then, We use the IVC (Invalidation Cache) in the F-COMA to bypass unnecessary AM accesses. Furthermore, we also propose a new cluster-based F-COMA architecture to increase AM utilization and decrease remote memory accesses. Finally, we combine all of optimization schemes in the F-COMA design to improve the total system performance dramatically.

The remainder of this paper is organized as follows. In Section 2, some background will be surveyed briefly, including CC-NUMA and F-COMA architecture. We propose some methods to reduce the memory access overhead in Section 3. System architecture parameters and benchmarks will be discussed in Section 4. Related performance evaluations will be shown and analyzed in detail in Section 5 based on our SEECOM. Finally, concluding remarks will be given in Section 6.

2.Fundamental Background

In this section, we survey the main features of the CC-NUMA, COMA and F-COMA

briefly.

2.1. CC-NUMA Architecture

CC-NUMA architecture provides a coherence single global address space to all processing nodes in the system. Each processing node in CC-NUMA architecture consists of one or more high performance processors, associated caches, a directory, and a portion of the global shared memory, as shown in Figure 1.

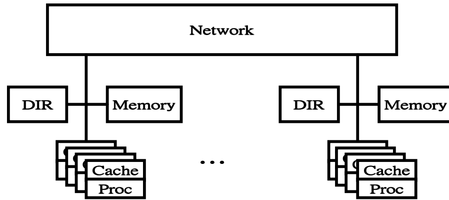

Figure 1. CC-NUMA Architecture.

Most of CC-NUMA architectures use a non-hierarchical directory scheme to maintain coherence protocol. In the scheme, the directory is distributed to each processing node correlated with the local memory, and each directory entry corresponds to a memory block is used to record the state of it. The detailed cache coherence operations are described in [7]. When the size of second level cache is smaller than that of the program data working set in CC-NUMA architecture, it will induce many capacity misses [8]. This capacity misses will increase the memory access latency, thus lengthen the execution time seriously. Another shortcoming of the CC-NUMA architecture is that data blocks at the main memory level cannot migrate or replicate without the aid of the operating system [2]. The data allocation in CC-NUMA will affect the system performance.

2.2. COMA Architecture [3]

COMA architecture is motivated by the idea that data blocks can dynamically migrate and replicate at the main memory level directly just as it does at the cache level in CC-NUMA architecture. To support data migrating and replicating, the organization of the local memory in each processing node, called *attraction memory* (AM), is the same as the cache in the CC-NUMA architecture. A portion of memory blocks in AM is left to store the migrated and replicated memory blocks. If there is sufficient locality of memory accesses in AM, we can reduce the number of remote memory accesses. Figure 2 shows a simple block diagram of the COMA architecture.

In COMA architecture, the cache coherence scheme is based on a hierarchical directory and a hierarchical interconnection network. The primary disadvantage of COMA architecture is the increased remote access penalty. It is caused by the hierarchical directory structure and the complex search path for the memory block because each memory block may be migrated to other processing node.

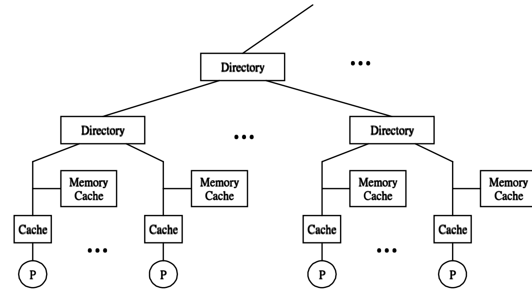

Figure 2. COMA Architecture.

2.3. F-COMA Architecture

F-COMA architecture provides the both benefits of CC-NUMA and COMA, as shown in Figure 3. This attraction memory permits the caching of frequently accessed memory blocks, increasing the local hit rate for memory references. The detail cache coherence operations described in [4].

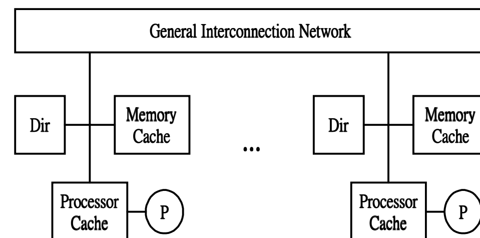

Figure 3. F-COMA Architecture.

There are still many overheads in F-COMA architecture. For example, migratory sharing pattern in CC-NUMA architecture will increase the memory access overhead [5]. The remote memory access latencies in F-COMA architecture are longer than that of the CC-NUMA architecture. Thus, migratory sharing patterns in F-COMA architecture will induce more memory access cost than in CC-NUMA architecture. When a second level cache miss occurs in F-COMA architecture, we must search the AM to make sure whether the memory block is in the F-COMA architecture or not. Therefore, if the memory block is not in AM, we would like to use some mechanisms to skip the unnecessary AM access and advance the remote memory access. Finally, the AM

utilization will affect the number of remote memory accesses in F-COMA. The remote memory accesses in F-COMA will reduce the total system performance. Hence, we want to increase the AM utilization in F-COMA.

3. Techniques to Reduce Memory Access Penalty

In this section, we will propose some methods to reduce the overhead of memory accesses in F-COMA architecture.

3.1. Reducing Migratory Sharing Overhead

Gupta and Weber have classified data structure based on the invalidation patterns they exhibit [9]. According to their definition, data structures manipulated by many processors but only a single processor at any time are called the migratory sharing object. In parallel programs, data structures which are modified within a critical section and high-level language statements such as I=I+1 exhibit migratory sharing patterns. A way of formally defining of migratory is as a sequence of read-modify-write operations by alternating processors on a data object. P. Stenstrom et al. [5] have formally defined the reference pattern of migratory memory blocks using the following regular expression:

$$\ldots(\mathbf{R}_i)(R_i)^*(\mathbf{W}_i)(R_i|W_i)^*(\mathbf{R}_j)(R_j)^*(\mathbf{W}_j)(R_j|W_j)^*\ldots \quad (1)$$

In the expression above R_i and W_i represent a read access and a write access, respectively, by processor i, '*' denotes zero or more occurrences of the preceding string, and '|' denotes the logical OR operation. In the sequence, 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. If we can detect the migratory sharing block, we can merge the read request and write request to a read-exclusive request that can reduce network traffic by all write request are removed. Most mechanisms always use the same idea to detect the migratory sharing patterns. For example, P. Stenstrom et al. have proposed a mechanism to reduce the overhead of migratory sharing patterns for centralized-directory cache coherence protocols in CC-NUMA architecture [5] and Pean et al. have proposed a mechanism to reduce the overhead of migratory sharing patterns for linked-based cache coherence protocols in CC-NUMA architecture [10]. According to the idea of detecting migratory sharing patterns described in the above subsection, we can build the migratory sharing optimization mechanism in the directory of the home node in the F-COMA architecture. If the home node detects the migratory access sequence, then it marks this memory block to be migratory sharing block. For a migratory sharing block, the home node converts a read request to a read-exclusive ones, as shown in Figure 4.

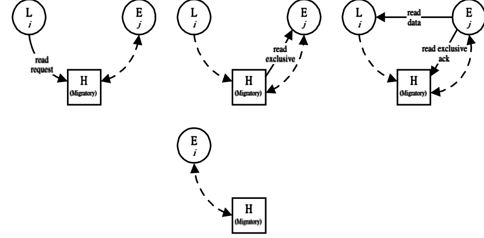

Figure 4. Read-Exclusive Request.

3.1.1. Migratory Sharing Optimization Method

In order to discover the migratory sharing memory block, we add a new migratory state in the directory to mark the migratory sharing memory block. The memory block is classified as migratory sharing when the following two requirements are fulfilled [11]:

1. $i \neq$ LW; the processor that issues the write request is not the same as the processor that most recently issued a write request to the memory block.

2. The number of memory block copies is exactly two.

Furthermore, we must change the original directory state transition to migratory sharing optimization directory state transition to detect the migratory sharing block, as shown in Figure 5. If home node finds the request satisfied the above conditions, it will mark this memory block as a migratory sharing block. When a memory block is marked as the migratory sharing block, a write request issued by other processors without issuing a read request in advance will violate the migratory sharing pattern sequence that is indicated in (1). In this situation, its state will return to the exclusive.

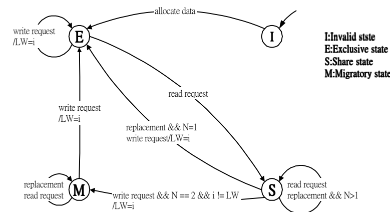

Figure 5. Migratory Sharing Optimization Directory State Transition Diagram.

3.1.2. Enhanced Migratory Sharing Optimization Method

In applications with false sharing patterns [12], the read misses will occur in our migratory sharing optimization method if two processors are involved. If we do not use migratory sharing detection method in the F-COMA architecture, these memory accesses will be read hits. In order to avoid increasing the number of read misses, we need to modify our method described above to an enhanced migratory sharing optimization method. In the enhanced method, we use a LW pointer and a new *last-last-writer* (LLW) pointer to record the last two processors that have written this memory block. When the LW pointer is updated with a new value, the old value of LW is replaced to the LLW pointer. Now, the memory block is marked to migratory sharing state when the following two new requirements are fulfilled:

1. $i \neq$ LW and $i \neq$ LLW; the processor to issue the write request must not be any of the last two processors wrote the memory block.
2. The number of memory block copies is exactly two.

These conditions not only let the memory block satisfy the migratory sharing pattern sequence, but also avoid marking false sharing patterns to be migratory sharing patterns. Thus, it can reduce the number of read misses in migratory sharing optimization method for application with false sharing patterns. Moreover, we must change the original directory state transition to enhance migratory sharing optimization directory state transition to avoid marking the false sharing patterns as migratory ones. The enhanced migratory sharing optimization directory state transition diagram is shown in Figure 6.

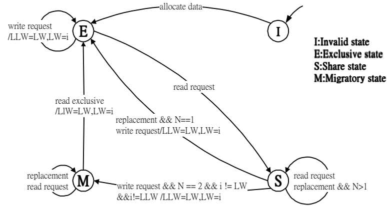

Figure 6. Enhanced Migratory Sharing Optimization Directory State Transaction Diagram.

3.2. Eliminate Unnecessary Attraction Memory Accesses

In CC-NUMA architecture, a second level cache miss triggers a message to the corresponding directory, which may be in a remote node or in the local node. The directory contains the message where the cache block is located. In F-COMA architecture, unlike in CC-NUMA architecture, a second level cache miss is always followed by a time consuming search in the local AM [4].

In F-COMA architecture, the AM access followed by a SLC (Second Level Cache) miss is possible unnecessary and it will delay remote memory access. Hence, we want to skip the unnecessary AM accesses. To avoid unnecessary accesses to AM, we add the Invalidation Cache (IVC) in F-COMA architecture [6], as shown in Figure 7. The IVC is a small direct-mapped cache that contains some addresses of the memory blocks that are not in the local AM. It will be accessed at the same time as the access to the second level cache. When the second level cache miss occurs, the IVC determines whether it contains the address of the memory block or not. If the IVC contains the address, the AM access will be skipped since the memory block is not in the local AM.

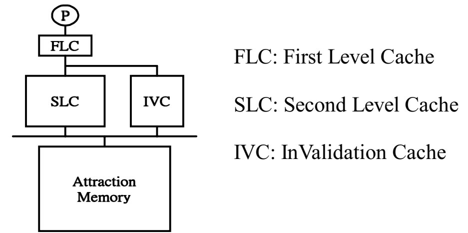

Figure 7. F-COMA architecture with IVC.

3.3. Decrease Remote Memory Accesses

In order to improve the total performance of F-COMA, we design our cluster-based F-COMA architecture as shown in Figure 8.

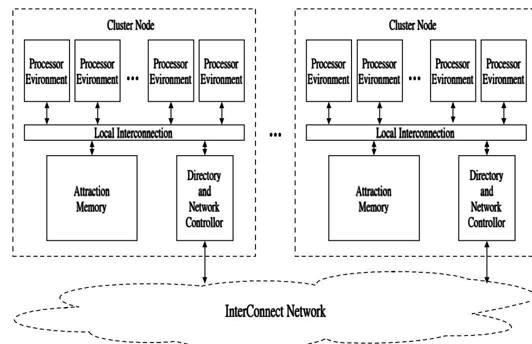

Figure 8. Cluster-based F-COMA Architecture.

The performance of the F-COMA architecture greatly depends on the efficiency of the AM. Clustering makes data be shared within

the cluster more efficient than in a non-clustered architecture. Data that is shared within the cluster will only need one copy and therefore use the AM efficiently. Hence, Cluster-based F-COMA can increase AM utilization and decrease remote memory accesses.

4. Simulation Environment and Benchmarks

Our simulation environment is a program-driven simulator named SEECOM (A Simulation and Evaluation Environment for cluster-based flat-Cache Only memory architecture) [13] and constructed based on the MINT [14] package. Our simulation system architecture is shown in Figure 8. The organization of the processor environment contains a CPU and a two-level cache, as shown in Figure 9.

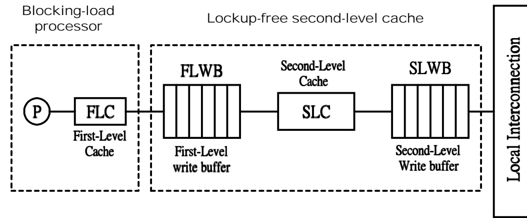

Figure 9. Processor Environment.

Our system architecture assumptions are summarized in Table1. The global interconnection network is a k-ary, n-cube network. We use several benchmark programs from the SPLASH and SPLASH2 suites [15,16] in our experimental evaluation. We list them in Table 2 along with the data sets being used.

Table 1. Simulation System Architecture Parameters.

Architecture Parameters	
Parameter	Value
Number of Processing Nodes	64
Size of FLC	32Kbytes
Size of SLC	256Kbytes
Block size of FLC and SLC	64bytes
Number of entries in FLWB	8
Number of entries in SLWB	16

Table 2. Benchmark Programs.

Benchmark	Description	Data sets
FFT	Blocked 1-D FFT	64k complex points
MP3D	Particle-based wind-tunnel simulator	5K particles, 10 time steps
RADIX	Integer Radix sort algorithm	1M integers, Radix 1024
WATER	Water molecule dynamics simulation	343 molecules
CHOLESKY	Cholesky factorization	bccstk14
OCEAN	Simulate eddy currents in an ocean basin	128x128 grid, tolerance 10^{-7}

5. Performance Evaluations and Results Analysis

As described in Section 3, we have designed the following four migratory sharing detection methods.

(1) MOR: migratory sharing optimization method with returning to exclusive state while the pattern of memory access violates migratory sharing sequence.

(2) MONR: migratory sharing optimization method without returning to exclusive state while the pattern of memory access violates migratory sharing sequence.

(3) EMOR: enhanced migratory sharing optimization method with returning to exclusive state while the pattern of memory access violates migratory sharing sequence.

(4) EMONR: enhanced migratory sharing optimization method without returning to exclusive state while the pattern of memory access violates migratory sharing sequence.

In Figure 10, we observe that the F-COMA with EMOR performs consistently better than the F-COMA without EMOR. The reason is that EMOR effectively detects migratory sharing memory blocks and reduces unnecessary write requests. However, the read miss stall time increase in MP3D, because there are some false sharing patterns such as false sharing data is shared over two processors in MP3D that EMOR cannot detect. The read accesses in the false sharing patterns will incur access misses that will increase the read miss stall time. We observe that using EMOR can also reduce the acquire stall time in F-COMA architecture because EMOR mechanism can reduce unnecessary memory access requests. Thus, acquire accesses can be performed early. In summary, the EMOR mechanism can substantially improve the performance of the system about 10% in average. We observe that the EMOR perform well than that of MOR for MP3D because the EMOR can eliminate the overhead of false sharing patterns involve with exactly two processors that are marked to be migratory sharing patterns. Hence, the EMOR mechanism can reduce the number of read misses more than that of the MDR mechanism. We also observe that the EMOR is performed better than that of the EMONR in all the benchmark programs. When the memory access patterns of the memory block violate the migratory sharing sequence, it will not be a

migratory sharing block in the future accesses. If we do not change the state of the memory block, the succeed read accesses will produce unnecessary invalidation to the same memory block in other AMs. It will increase the number of read misses. Hence, our migratory sharing optimization method needs the directory state of the memory block to be returned to exclusive state while memory access patterns violate the migratory sharing sequence.

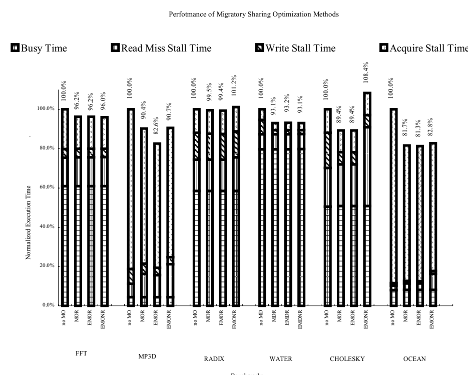

Figure 10. The Performance of Migratory Sharing optimization.

In Figure 11, we observe that using IVC can effectively reduce the normalized execution time because it can skip the unnecessary AM accesses. We find that the performance of F-COMA with 4k entries IVC is closed to that of F-COMA with 16k entries IVC. Thus, 4k entries IVC is enough to reduce the unnecessary AM accesses. The performance of IVC depends on the number of invalidations and replacements that benchmark program contains. In summary, the F-COMA with IVC can improve the performance of the system about 5% in average.

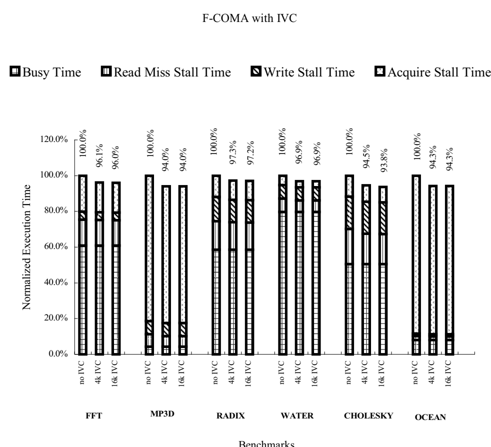

Figure 11. The Performance of F-COMA with IVC.

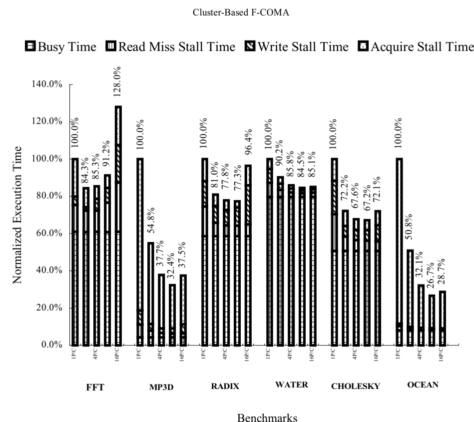

Figure 12. The Performance of Cluster-based F-COMA

Figure 12 shows the normalized execution time of the cluster-based F-COMA for different number of processors in each cluster node. Basically, clustering several processors in a node can increase the utilization of AMs. Thus, if there are more processors in the same cluster node, the performance will improve more. However, when the number of processors in the cluster nodes is more than 8 processors, the performance will decrease for FFT, MP3D, RADIX, CHOLESKY, and OCEAN benchmarks because it causes more resource contentions. In words, the processors in the same cluster node must wait for the resource until other processors in the cluster release it. The waiting time of processors will increase the memory access stall time. Another reason reduced the performance is the destructive interference [17,18]. When one of the processors in the cluster node replaces the memory block in the AM that another processor in the same cluster node needs to access, the number of memory access misses will increase. Furthermore, memory access misses will decrease the advantage of clustering. In summary, 4 or 8 processors in a cluster node is suitable for the cluster-based F-COMA architecture.

So far, we know that the migratory sharing optimization mechanism, F-COMA with IVC, and cluster-based F-COMA can improve the performance of F-COMA architecture in some degree. Now, we will evaluate the performance improvement by combining these effective mechanisms. In Figure 13, we find that the combined mechanism improves the system performance about 39% in average than that of F-COMA without using those optimizations. Hence, F-COMA with the combined mechanism can indeed speedup the total system performance of the F-COMA architecture in some degree.

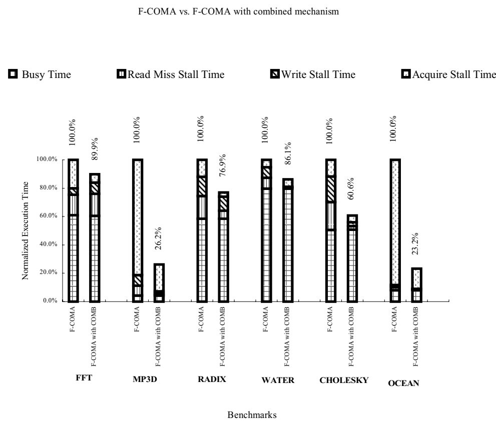

Figure 13. The Performance of F-COMA with COMB.

6. Concluding Remarks

In this paper, we have proposed several effective mechanisms to reduce memory access latencies, including migratory sharing patterns optimization method, invalidation cache, cluster-based F-COMA architecture, and combined methods. Moreover, we have constructed a simulation environment called SEECOM, a software platform used to evaluate our F-COMA performance.

The F-COMA with EEMOR can increase the performance about 10% in average, since our migratory sharing optimization methods can effectively reduce write stall time and acquire stall time. The IVC in the F-COMA can improve the system performance about 5% due to the elimination of unnecessary AM accesses.

On the other hand, the cluster-based F-COMA architecture is effective to increase utilization of AMs and speedup the system performance. It is appropriate for the cluster-based F-COMA architecture with 8 processors in a cluster node. The F-COMA with combined mechanism can speedup the total system performance of F-COMA architecture about 39% in average and it spends reasonable cost. Thus, it is an efficient and cost effective integrated method to improve the performance of the F-COMA architecture.

When a processor issues the read request to the remote AM, the processor must stall until the memory block is available. In the future, we can use multithread technique to promote the predicament. When a processor is stalled by the read request, it can switch to another thread and still keep running. It can effectively hide the read stall latencies. On the other hand, we can use extra memory area called the unallocated memory to store remote migrated and replicated memory blocks in CC-NUMA architecture. The unallocated memory has the ability as AM in the F-COMA architecture. This architecture has the advantage of reducing the design complexity of F-COMA architecture.

Acknowledgment

The authors would like to thank the reviewers for their helpful comments. This research was supported by the National Science Council of the Republic of China under contract numbers: NSC89-2213-E009-026.

References

[1] K. Hwang, *Advanced Computer Architecture*, McGraw-Hill, 1993.

[2] D. Lenowshi, and J. London, *Stanford DASH Multiprocessor*, University of Stanford, Technical Report: CS-TR-89-403, 1989.

[3] F. Dahlgren, J. Torrellas, “Cache-Only Memory Architectures”, *IEEE Computer*, 32(6):72-79, June 1999.

[4] T. Joe, *COMA-F: A Non-Hierarchical Cache Only Memory Architecture*, Stanford University, Ph.D. thesis, March 1995.

[5] P. Stenstrom, M. Brorsson, L. Sandberg, “An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing”, In Proceedings of the 20th *International Symposium on Computer Architecture*, pp. 109-118, 1993.

[6] L. Yang, J. Torrellas, “Speeding up the Memory Hierarchy in Flat COMA Multiprocessors”, In 3rd *International Symposium on High Performance Computer Architecture*, pp.4-13, 1997.

[7] D. B. Glasco, *Design and Analysis of Updated-Based Cache Coherence Protocols for Scalable Shared-Memory Multiprocessors*, Stanford University, Technical Report: CS-TR-95-670, June 1995.

[8] P. Stenstrom, T. Joe, A. Gupta, “Comparative Performance Evaluation of Cache-Coherence NUMA and COMA Architectures”, In the 19th Annual *International Symposium on Computer Architecture*, pp.80-91, 1992.

[9] A. Gupta, W.D. Weber, “Cache Invalidation Patterns in Shared-Memory Multiprocessors”, *IEEE Trans. On Computer*, 41(7):794-810.

[10] D. L. Pean, J. R. Wu, C. Chen, "Effective Mechanisms to Reduce the Overhead of Migratory Sharing for Linked-Based Cache Coherence Protocols in Clustering Multiprocessor Architecture", In Proceedings of the *International Conference on Parallel and Distributed Systems*, pp. 511-518, December 1998.

[11] H. Nilsson and P. Stenstrom, "An Adaptive Update-based Cache Coherence Protocol for Reduce of miss rate and traffic", In Proceeding of *PARLE Conference*, pp. 336-374, July 1994.

[12] Torrellas J., M. S. Lam, and J. L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches", *IEEE Transactions on Computers* 43(6): 651-663, June 1994.

[13] L. M. Chung, *A Study on Reducing Memory Access Latency for F-COMA System Design and Implementation of Its Simulation and Evaluation Environment*, Chiao Tung University, Master thesis, June 2000.

[14] E. J. Veenstra, R. J. Fowler, *MINT Tutorial and User Manual*, Rochester University, Technical Report 452, 1994.

[15] J. P. Singh, W. D. Weber and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory", *Computer Architecture News*, 20(1): 5-44, March 1992.

[16] S. C. Woo, M. O'Hara, E. Torrie, J. P. Singh and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations", In Proceedings of the 22nd Annual *International Symposium on Computer Architecture*, pp.24-36, June 1995.

[17] A. Erlichson, B. A. Nayfeh, J. P. Singh, K. Olukotun, *The Benefits of Clustering in Shared Address Space Multiprocessor: An Applications-Driven Investigation*, Stanford University, Technical Report: CSL-TR-94-632, 1994.

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