Reducing the Penalty of False Sharing Patterns and Replacement Operations in F-COMA

Jiang-Long Wu, Der-Lin Pean and Cheng Chen

Department of Computer Science and Information Engineering

1001 Ta Hsueh Road, Hsinchu, Taiwan, 30050, Republic of China

Tel:(886)35712121 EXT:54701~54734, Fax:(886)35724176

Abstract

The F-COMA architecture provides the benefits of data replication and migration. Thus, the requested data block by every processor can be accessed directly in the local node. However, frequently coherence misses will drop down its performance. Because remote accesses caused by these misses induce critical latencies in it. It is important to reduce unnecessary coherence misses especially the false sharing misses in the F-COMA architecture. On the other hand, because there is only cache system in the F-COMA, we have to reserve the last valid memory block in the cache system while it is replaced. In this paper, we use an effective sub-block mechanism to reduce the impact of false sharing accesses, and three replacement techniques to decrease replacement stall time. According to our evaluation results, these two methods speedup the total system performance about 5% in average under SPLASH benchmarks. The sub-block mechanism could decrease the false sharing miss ratio and miss stall time. The replacement techniques could decrease the replacement stall time.

Keywords: F-COMA, False sharing, Sub-block mechanism, Replacement stall.

1. Introduction

F-COMA [1] is a multiprocessor architecture originated from COMA [2]. It uses AM (Attraction Memory) that is a cache structure instead of the shared memory to provide data replication and migration. Hence, it generally has higher local hit rate than the CC-NUMA architecture [1,2]. It also flattens the hierarchical tree structure in COMA to an interconnection network structure. Thus, it can effectively reduce the communication latency of traversing the hierarchical tree. However there are still some drawbacks in the F-COMA architecture. Such as *false sharing* misses and *replacement* overheads [3,4]. The penalty of false sharing misses in the F-COMA architecture is larger than that of the CC-NUMA architecture, because remote accesses should query the directory about the location of the master block. Stemming from there is no memory in the F-COMA architecture, and it is necessary for every replaced block to find another processor to it while it is the last valid one. Hence, replacement technique is also an important issue in the F-COMA architecture.

In this paper, we propose a sub-block mechanism [5,6] to reduce the penalty of false sharing misses and three replacement techniques to reduce the overhead of replacement. In order to evaluate our mechanisms, we constructed a simulation environment called SEECOM (Simulation and Evaluation Environment for Cache Only Memory Architecture). By advancing sub-block degrees the number of false sharing misses can be reduced. However, sub-block degree of 2 is sufficient to reduce the penalty of false sharing misses. It can improve the system performance about 1% in average due to the limit amount of false sharing misses existed in SPLASH benchmarks. On the other hand, F-COMA with the *swap* replacement technique improves the system performance about 2% in average. It can be enhanced about 4.5% in average for the *buffer* replacement technique. With the sub-block mechanism and the *buffer* replacement technique, F-COMA can be promoted about 5% in average.

In Section 2, we will illustrate when the false sharing misses and replacement will happen in the F-COMA architecture. Section 3 introduces how to implement the sub-block mechanism and three replacement techniques. Our SEECOM and simulation results in it will be described in Section 4. Finally, we will give the concluding remarks in Section 5.

2. Related Work

In multiprocessor systems, there are probably more than two objects co-locating in the same block, and it will produce undesirable side effects. For example, we assume a data block contains two words: 'A' and 'B', and processor P1 and P2 access the same data block. If P1 and P2 write the word 'A' and the word 'B' exclusively, a series of access misses will occur. Such misses are called *false sharing misses* [3,4]. Actually, if we use one word as a data block, these misses will disappear. However, reducing of block sizes will decrease the data locality and increase the number of accesses. Therefore, we use the sub-block mechanism to reduce the penalty of false sharing misses and reserve the data locality in the same time. False sharing misses have more influences in the F-COMA than in CC-NUMA due to the following reasons 1). Cache coherence protocol is built in AM in the F-COMA, but it is built in SLC in CC-NUMA. Thus, it is necessary to access AM before remote accesses in the F-COMA. 2). It always checks the directory to see where its target processor is in the F-COMA. Hence, its remote access operation is always a 3-hops one, but it maybe 2-hop or 3-hop one in CC-NUMA. 3). There are more false sharing misses in the F-COMA as shown in Fig. 1. It results from there are more shared blocks in the F-COMA since it provides data replication. Thus, reducing the penalty of false sharing in F-COMA is more important than that in CC-NUMA.

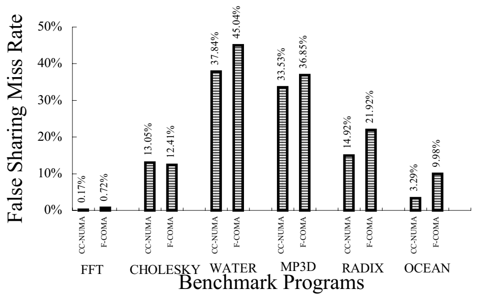


Fig. 1. False Sharing Miss Rate in F-COMA and CC-NUMA

Because there is no memory in F-COMA, it must reserve the last valid copy of the block in AM. In F-COMA, there is an exclusive operation called *relocation*. Relocation is meant when the replaced block is the last valid block in F-COMA, and it must find another processor to reserve itself. Unlike CC-NUMA, its replacement has to write back the data block to the shared memory. Furthermore, when relocation occurs, F-COMA has to move replaced block from an AM to another AM in other processor. Hence, the overhead of replacements in F-COMA is higher than that in CC-NUMA. In the next section, we will explain how we implement our sub-block mechanism and replacement techniques.

3. Sub-block Mechanism and Replacement Techniques

3.1. Sub-block Mechanism

The organization of our sub-block mechanism is shown in Fig. 2. The first row shows the organization of each data block. Every data block has a data block state field, a data block tag field and a data block content. Each data block content contains several sub-blocks. We assume a data block is divided into four sub-blocks here. The second row shows four sub-blocks, and the third row shows each field in a sub-block.

Data Block State	Tag	Data Block Content							
		Sub Block		Sub Block		Sub Block		Sub Block	
		Sub Block State	Data	Sub Block State	Data	Sub Block State	Data	Sub Block State	Data

Figure 2. The Organization of Our Sub-block Mechanism

Because the sub-block mechanism changes the cache organization, we need to modify the cache coherence protocol to operate on it. At first, we define the data block states and sub-block states. Then we present the state diagram of our sub-block mechanism. Finally, we give a scenario to explain how the protocol works. We use italic characters to present state words.

We define four states of the sub-block as below, and the first character is lowercase: (1) *exclusive*: the only valid sub-block in system; (2) *master*: the block has the ownership, but there are other processors share the data block; (3) *shared*: the block does not have ownership, and it just shares the data block; (4) *invalid*: the block is not a valid block; We also define four states of the data block as below; the first character is uppercase: (1) *Exclusive*: the data block has at least one sub-block with *exclusive* state; (2) *Master*: the data block has at least one sub-block with *master* state and no sub-block with *exclusive* state; (3) *Shared*: the data block only contains sub-blocks with *shared* or *invalid* states; (4) *Invalid*: the data block only contains sub-blocks with *invalid* state.

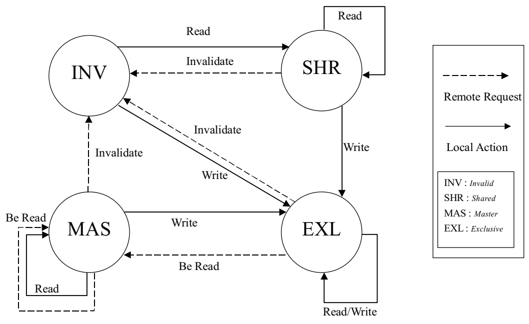


Figure 3. The stat transition diagram of our sub-block mechanism

The state transition diagram of our sub-block mechanism is shown in Fig. 3. We introduce the purpose of defining these sub-block states in the following. (1) Because the last valid block could not be replaced without storing it, we need *exclusive* state to express that the sub-block is the last valid copy in F-COMA. When it is replaced, it has to do relocation. Moreover, when an *exclusive* state sub-block is written, it is hit and does not need to invalidate other shared blocks. (2) When a processor reads some block, it should know which processor has the ownership of the sub-block. Hence, we use *master* state to make clear which processor has the ownership of the sub-block. Then, requesting processor can read data block from the processor. When it is replaced, it needs to transfer its ownership to another processor. (3) The *shared* state means that the sub-block is only a copy of the block. Other processors would not read the sub-block. When it is replaced, it just has to tell the directory to update the information about its processor number from the shared list.

Data block state is decided by the state its sub-block. We define the data block state in order to eliminate the checking time of all sub-block states when a data block is replaced. If a block with *Exclusive* state is replaced, it will change its ownership and transmit data to another processor. If a block with *Master* state is replaced, it will change its ownership to another processor. If a block with *Shared* state is replaced, it will tell the directory to update the information about its processor from the shared list.

Now we give a scenario to show how the sub-block mechanism works. Initially, all sub-blocks in F-COMA are *invalid* states. We divide a data block to four sub-blocks that are referred to as *a* through *a+3*. We assume all processors access the same data block in the scenario. We use the accessing flow as P1 reads sub-block *a+2*, P2 writes sub-block *a+1*, P3 reads sub-block *a*, P1 writes sub-block *a+3*, and P1 replaces the data block at last. Each step has a data block state and its sub-blocks states are described below. The leftmost part of the data block picture is data block state, and other four regions are sub-block states.

1. Processor P1 reads sub-block *a+2*. The state of data block is *Exclusive*, and the states of all sub-blocks are also *exclusive* because only processor P1 has the data block.

P1

E	e	e	e	e

2. Processor P2 writes sub-block *a+1*. The state of sub-block *a+1* is *exclusive* and the states of other sub-blocks are *invalid* at processor P2. Consequently, the state of data block is *Exclusive* at processor P2. The sub-block *a+2* in processor P1 changes its state to be *invalid* since processor P2 invalidates it. In the same time, the state of data block does not need to be changed at processor P1.

P1

E	e	i	e	e

P2

E	i	e	i	i

3. Processor P3 reads sub-block *a*. Because processor P1 has the ownership of sub-block *a*, processor P3 reads the data block from processor P1. After the read operation is processed, the sub-blocks *a*, *a+1* and *a+3* are in *shared* states at processor P3, and the data block state is *Shared*. At processor P1, the states of sub-blocks *a*, *a+2* and *a+3* are changed to be *master* since processor P3 reads these sub-blocks. Thus, the data block state is also changed to be *Master* state.

P1

E	m	i	m	m

P2

E	i	e	i	i

P3

S	s	i	s	s

4. Processor P1 writes sub-block *a+3*. The state of sub-block *a+3* is changed to be *exclusive* at processor P1. The state of sub-block *a+3* changes to be *invalid* at processor P3 because processor P1 invalidates it. The states of data blocks do not change at processor P1 and processor P3 since the ownership of all sub-blocks are not changed.

P1

E	m	i	m	e

P2

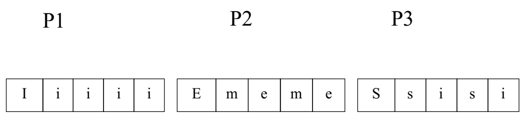
E	i	e	i	i

P3

S	s	i	s	i

5. Processor P1 replaces the data block. The target processor is selected in turn here. Thus, the target processor is the next processor

P2. After replacing the data block from processor P1 to processor P2, the state of sub-block *a* and *a+2* are changed to be *master* since the states of two sub-blocks are *master* at processor P1 before. The states of sub-block *a+1* and *a+3* are *exclusive* at processor P2 because the states of two sub-blocks are *exclusive* at processor P1 before.



We have constructed the sub-block organization and presented the modified cache coherence protocol. The performance evaluations of our sub-block mechanism will be given later.

3-2. Replacement Techniques

In F-COMA, the replacement target processor is selected in turn. This technique has three drawbacks. First, if the target processor does not have space to store the replaced block, the target processor needs to replace another block or check the next processor for free location. Second, the target processor may reuse the latter replaced block, but an unnecessary miss occurs. Third, the target processor may not use the replaced block before another processor uses it, and another replacement occurs. Hence, we will propose three new replacement techniques, named *locality*, *swap* and *buffer*, to improve the total system performance. As for the first replacement technique, after every access the *directory* records the id of the processor that accesses the data block recently. When relocation occurs, the replaced block is stored to the processor. Because the processor accesses the data block recently, it may access the data block again due to temporal locality. The overhead of this technique is that the *directory* must have enough space to record the id of the processor.

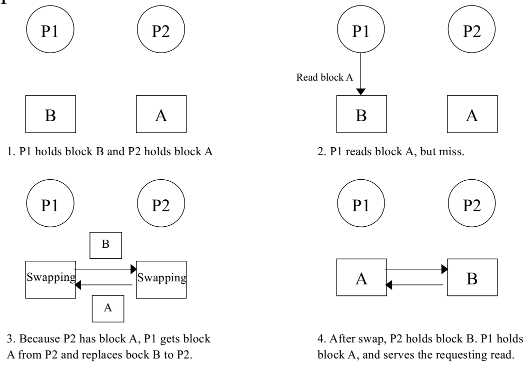

P1
P2
B
A
1. P1 holds block B and P2 holds block A
P1
P2
Read block A
B
A
2. P1 reads block A, but miss.
P1
P2
B
Swapping
Swapping
A
3. Because P2 has block A, P1 gets block A from P2 and replaces bock B to P2.
P1
P2
A
B
4. After swap, P2 holds block B. P1 holds block A, and serves the requesting read.

Figure 4. The step of *swap* replacement technique

The second replacement technique is called *swap*. When an *Exclusive* data block is replaced, it is replaced to the processor that has the data block which the requesting processor wants to read or write. For example, processor P1 holds block B and processor P2 holds block A as shown in Fig. 4. We assume block A and B have the same index. At first, processor P1 reads block A, however, there is no space to serve the read request. Hence, block B needs to be replaced. We swap the block B and A between processor P1 and P2. After swapping, processor P2 gets the block B. In the same time, processor P1 not only replaces block B but also reads hit since it gets block A from processor P2. The advantage of the *swap* replacement technique is that it always needs only one replacement and its swap time is less than replacement stall time plus the consequent access miss time.

The last technique is called *buffer*. Intuitively, we add a one-entry buffer for each processor to store the replaced block. Whether the states of replaced blocks are *Shared*, *Master* or *Exclusive*, it is buffered in the buffer of the processor in order to implement replacement. Therefore, the consequent access continues without suspending. The replaced target processor is also selected in turn. The advantage of the *buffer* replacement technique is that there is no replacement stall time as long as the buffer is empty. The overhead of the *buffer* replacement technique is the cost of replacement buffer. The performance evaluations of these replacement techniques will be given in the next section.

4. Preliminary Performance Evaluations

We have introduced sub-block mechanism and replacement techniques in the preceding section. In order to evaluate them in some detail, it is necessary to build a simulation environment [7,8]. Our simulation environment is a program-driven simulator environment that has two main parts: memory reference generator (Front-end) and memory subsystem (Back-end) simulator. MINT [9] is the front-end of our simulation environment, and our F-COMA is the back-end. Because our simulation environment is used to evaluate issues in F-COMA, we call it SEECOM (<u>S</u>imulation and <u>E</u>valuation <u>E</u>nvironment for <u>C</u>ache <u>O</u>nly <u>M</u>emory Architecture) [10]. Our F-COMA architecture is a cluster-based F-COMA, and a cluster node contains four processor elements, an AM, a directory controller and a network controller, as shown in Fig. 5. Each processor element contains a FLC, FLWB, SLC and SLWB respectively as shown in Fig. 6.

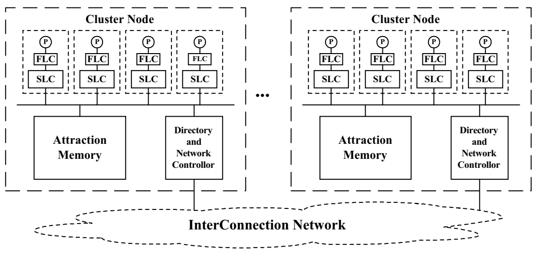


Figure 5. Our F-COMA Architecture

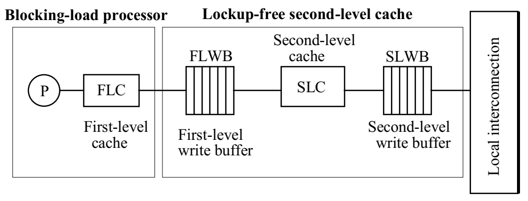


Figure. 6. Organization of a Processor Element

The parameters of our simulation environment are shown in Table 1 and access times of resources are shown in Table 2. Because the size of the AM is large, it is always constructed by DRAM other than SRAM for the purpose of cost-effectiveness. Thus, the latency of AM is much larger than that of the FLC and SLC. We evaluate six benchmarks in SPLASH and SPLASH2 benchmark suite developed by Stanford University [11,12]. They are MP3D, WATER, FFT, CHOLESKY, RADIX and OCEAN as shown in Table 3.

Table 1. Important Parameters of Our Simulation Environment

Parameters	Value
Processor No.	64
FLC Size	32 Kbytes
SLC Size	128 Kbytes
AM Size	1 Mbytes
Block Size	128 bytes
FLWB Entry No.	8
SLWB Entry No.	16

Table 2. The Access Time of Each Resource

Description	Access Time
Fill from FLC	1
Fill from SLC	8
Fill from AM	36
Network Service Time	3
Network Transfer Time (per FLIT)	2

Table 3. Characteristics of Benchmarks in SPLASH and SPLASH2

Benchmark	Characteristic	Program Size
FFT	Fast Fourier Transform (High-radix)	64K complex points
MP3D	High network traffic. True sharing.	50000 particles
CHOLESKY	High computation and communication.	Bcsstk14
RADIX	Communicate through write	2^^20 items
OCEAN	Computation between grids regularly	130-by-130 grids
WATER	Local computation	343 molecules

4-1. Evaluations and Analysis of Sub-block Mechanism

Miss rate decreases as the sub-block degree advances as shown in Fig. 7. It indicates that false sharing effect can be reduced according to the advancement of the sub-block degree advances. Indeed, we can realize that there are little false sharing misses in the FFT, CHOLESKY and OCEAN benchmarks. Hence, the total miss rates do not reduce mostly when sub-block degree advances. The WATER, MP3D and RADIX benchmarks have more false sharing misses within local misses. Thus, their local miss rates could be reduced apparently.

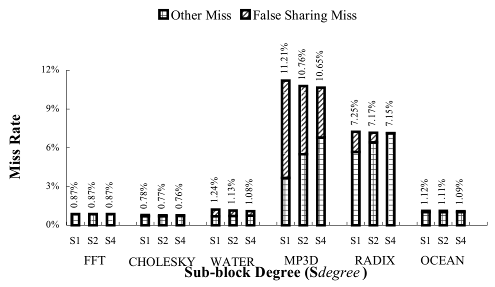


Figure. 7. Miss Rate with Sub-block Mechanism

The execution times and network traffic of those benchmarks are shown in Fig. 8. The FFT, CHOLESKY and OCEAN benchmarks cannot improve the performance while sub-block degree advances because they have little false sharing misses can be reduced However, the WATER and MP3D benchmarks have the better performance only when sub-block degree is 2. The RADIX benchmark can get better performance when sub-block degree advances. We can realize why the execution time cannot be reduced when sub-block degree advances from network traffic of these benchmarks. The network traffic of FFT, CHOLESKY and OCEAN benchmarks will increase while sub-block degree advances and other three benchmarks have less network traffic only when sub-block degree is 2. Hence, we can realize that advancing the sub-block degree can reduce read miss stall time, but higher network traffic increases the acquire stall time in the same time. If read miss stall time is thus less than the acquire stall time, the reduced miss stall time is less than the increased acquire stall time. The execution time could not be reduced while sub-block degree advances. Hence, only the RADIX benchmark can reduce execution time about 1.5% and the MP3D and OCEAN benchmarks could not because their acquire stall times are longer than read miss stall times.

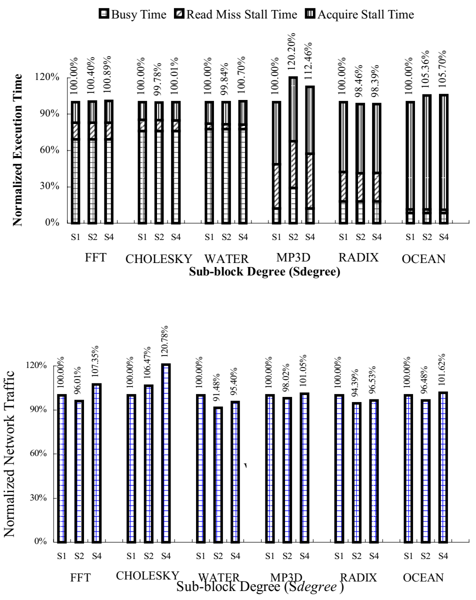


Figure.8. Execution Time and Network Traffic with Sub-block Mechanism

In summary, sub-block mechanism cannot reduce the execution time of most benchmarks except the RADIX because their false sharing misses are too less to affect the performance of F-COMA. The main function of the sub-block mechanism is to reduce read miss stall time, but acquire stall time may increases since network traffic grows up. If the acquire stall time is longer than the read miss stall time, advancing the sub-block degree will result in side effects. As illustrated in above figures, we know that network traffic is small while sub-block degree is 2, and network traffic is higher while sub-block degree is bigger than 2. In addition, advancing one sub-block degree is expensive. Therefore, we conclude that it is cost effective to divide 128 Kbytes block into two coherence blocks.

4-2. Evaluations and Analysis of Replacement Techniques

Because there is no replacement in the WATER benchmark, we do not discuss it in this issue. The *locality* replacement technique is used to explore data locality in order to reduce the number of replacements. We assume that the replaced block being accessed will be accessed again by the processor, so we replace the block to the processor. If the benchmarks do not have the property of locality with our mechanism, the number of replacements will increase. Hence, their execution time is longer than original replacement technique such as the FFT and RADIX benchmarks as shown in Fig. 9. Other benchmarks have shorter execution times, but the mechanism does not reduce their execution time apparently. The advantage of the *swap* replacement technique is that its swap stall time is less than replacement stall time plus consequent access miss stall time despite its distance. However, if the processor that we replace to does not use the replaced block before another processor accesses it, it will be replaced again. Furthermore, if the distance between local processor and the target processor is too long, the swap stall time will be longer than replacement stall time plus consequent access miss stall time. Thus, the performance of the *swap* replacement technique is not absolutely better than original replacement technique. Because the OCEAN benchmark has high communication and the *swap* replacement technique can get the requested block quickly, the stall time will be reduced about 9%. About the *buffer* replacement technique, if the buffer of the replaced block is not full, the replacement stall time will be eliminated. Hence, the *buffer* replacement technique has the shortest execution time. Because the OCEAN benchmark has higher replacement ratio and communication and the *buffer* replacement technique can free the space of the replaced block quickly, its replacement stall time can be reduced apparently about 22%.

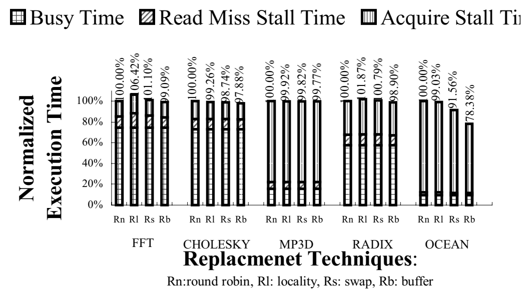


Figure. 9. Execution Times of Different Replacement Techniques

4-3. Evaluations and Analysis of Scalability

We can realize that sub-block degree of 2 and the *buffer* replacement technique are the better designs for our F-COMA. Now, we compare our F-COMA with these mechanisms with original F-COMA to evaluate their scalabilities as shown in Fig. 10. Because our sub-block mechanism makes more shared blocks exist in F-COMA, there will be more replacements. Hence, our sub-block mechanism will get better performance with replacement techniques. Furthermore, more processors have more false sharing misses since access patterns are distributed into more processors. Thus, the effect of sub-block mechanism is more apparent. As the same reasons above, the combined method gets better performances in the WATER and MP3D benchmarks when the number of processors is 128. In other benchmarks, the combined method has about the same scalabilities as original.

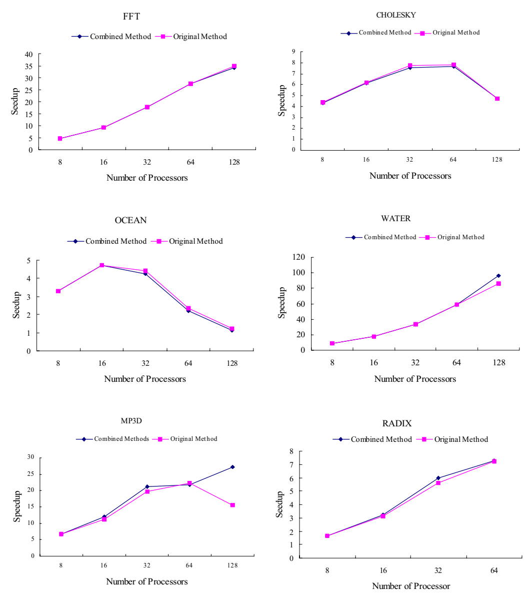


Figure. 10. Scalabilities of Different Replacement Techniques

5. Concluding Remarks

In this paper, we address sub-block mechanism to reduce the penalty of false sharing and some replacement techniques to reduce replacement stall time. In addition to study these two issues, we have constructed the SEECOM simulation environment. It includes following functions: (1) Two memory consistency models: sequential and release models, (2) Two-level cache hierarchy with FLWB and SLWB, (3) Different degrees of AM set-associative, (4) K-ary, N-cube interconnection network, (5) Subblock mechanism, (6) Four replacement techniques, (7) Non-cluster and cluster-based F-COMA architecture, (8) Two migratory sharing detection methods use hardware mechanisms and (9) Invalidation cache.

After analyzing and evaluating those simulation results, we get some conclusions as follow: (1) sub-block degree of 2 is sufficient to reduce the penalty of false sharing, and is cost-effective. It can improve about 1% in average. (2) If the benchmark has no the property of locality with locality mechanism, the replacement stall time may be longer than that of the original technique. (3) In most benchmark programs, F-COMA with the *swap* replacement technique could improve the performance about 2% in average. Because the network transfer time depends on the distance between processors, it could not get the better performance while requesting processor is far from target processor in the FFT and RADIX benchmarks. (4) The *buffer* replacement technique uses a one-entry buffer and it can serve almost all the replacements. F-COMA with it can improve the performance about 4.5% in average. (5) sub-block degree of 2 combined with the *buffer* replacement technique is the most effective solution for improving F-COMA performance according to our simulation results. It could improve the performance about 5% in average.

Because advancing sub-block degree will increase network traffic and acquire stall time, we want to eliminate unnecessary sub-block. Thus, we may use parallel compiler to mark which block need to be divided into some sub-blocks to reduce number of false sharing misses for future. By this way, there will be no unnecessary network traffic and acquire stall times due to unnecessary sub-blocks. On the other hand, we will evaluate other classes of computing, such as commercial database and multiprogramming environment applications. They have significantly different characteristics from scientific workloads and the results obtained here may be substantially different.

Acknowledgement

This research was supported by the National Science Council of the Republic of China under contract numbers: NSC 89-2213-E009-026.

References

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

[2] F. Dahlgren and J. Torrellas,

"Cache-Only Memory Architectures," *IEEE Computer*, 32(6):72-79, June 1999.

[3] Josep Torrellas, Monica S. Lam and John L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches," IEEE Transactions of Comuters. VOL. 43, NO. 6, JUNE 1994.

[4] Randall L. Hyde and Brett D Fleisch, "An Analysis of Degenerate Sharing and False Coherence," Journal of *Parallel and Distributed Computing* 34, pp. 183-195 , 1996, Article No. 0054.

[5] Craig Anderson and Jean-Loup Baer. *Design and Evaluation of A Sub-block Cache Coherence Protocol for Bus-based Multiprocessors.* Technical Report 94-0502, University of Washington, 1994.

[6] Murali Kadiyala and Laxmi N. Bhuyan. "A Dynamic Cache Sub-block Design to Reduce False Sharing," 1995 IEEE International Conference on , Page(s): 313 –318, Oct. 1995.

[7] J. P. Su, *A Study of Memory Subsystem Design for Multiprocessor System and Implementation of Its Simulation and Evaluation Environment*, National Chiao Tung University, Master thesis, June 1996.

[8] C. C. Wu, D. L. Pean, J. P. Su, J. R. Wu, H. W. Hwang, J. L. Huang, J. L. Lee, H. T. Chua and C. Chen, "SEESMA: A Simulation and Evaluation for Shared-Memory Multiprocessor Architecture," In Proceedings of *the National Science Council*, Vol. 22, No. 4, pp. 524-538, July 1998.

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

[10] J. L. Wu, *A Study on Reducing False Sharing and Replacement Stall for F-COMA and Implementation of Its Simulation and Evaluation Environment*, Chiao Tung University, Master thesis, June 2000.

[11] 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.

[12] 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.