JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 18, 803-813 (2001)

# Short Paper

## A Second-Level Cache With the Distance-Aware Replacement Policy for NUMA Systems*

SUNG WOO CHUNG, JEONG-HEON SHIN, HYONG-SHIK KIM+
AND CHU SHIK JHON
*School of Electrical Engineering and Computer Science*
*Seoul National University*
*Seoul, 151-742 Korea*
+*Department of Computer Science*
*Chungnam National University*
*Taejon, 305-764 Korea*

The cache replacement policy is one of the most important factors that affect the cache performance. With the trend of increasing associativity in second-level caches, implementing an efficient replacement algorithm becomes more important than just eliminating conflict misses. The LRU cache replacement algorithm has been known to work well in a single processor system by reducing the cache miss rate, but it does not minimize the cache replacement cost on the interconnection for a multiprocessor system because it does not take the distance into account. In this paper, we suggest a *distance-aware* second level (L2) cache for scalable multiprocessors, which is composed of a traditional LRU cache and an additional *SDF (Shortest Distance First) cache*. The LRU cache selects a victim using age information, while the SDF cache does so using distance information. Both work together to minimize the overall replacement cost by keeping long-distance blocks as well as recently used blocks. The combined L2 cache reduces the cache miss rate compared to the original LRU cache in many cases. With 32 processors, a 512KB LRU/SDF L2 cache outperforms a 512KB LRU L2 cache. Moreover, the replacement traffic on an interconnection network such as the ring is suppressed by up to 69%, which is expected to bring more scalability to multiprocessor systems.

***Keywords:*** cache replacement policy, cache miss rate, memory hierarchy, parallel architecture, performance evaluation

### 1. INTRODUCTION

Associative caches need a cache replacement policy to make room for a new cache block when a conflict miss or a capacity miss occurs [1]. In a single processor system, the cache replacement policy concentrates on reducing the cache miss rate. In other words, it is the goal of the cache replacement policy to delay replacement of a cache block that may be reused soon. Many cache replacement policies have been proposed. The simplest one is the random replacement policy, which chooses a victim randomly

Received July 31, 2001; accepted April 15, 2002.
Communicated by Jang-Ping Sheu, Makato Takizawa and Myongsoon Park.
* This work was supported by Brain Korea 21 project in 2002.

among all the blocks in the set. Among the variations of the random replacement policy [2-4], the most popular replacement policy is LRU (Least Recently Used), in which the block that has been unused for the longest time is chosen as a victim to make use of the principle of locality. Although the LRU policy requires additional logic - some extra bits per cache block to store the access time and its control logic - this is not a significant burden.

In a multiprocessor system, however, a lower cache miss rate does not always lead to better performance. Since a replacement incurs network traffic among processors, the effect on the traffic should be considered together with the cache miss rate [5]. Replacing a block whose home node is local takes less time than replacing a block whose home node is remote. Moreover, replacing a block whose home node is located a short distance away takes less time than replacing a block whose home node is far away since the communication costs between each node may not be equal. When selecting a victim among the cache blocks in L2 caches, therefore, we may have to minimize not only the miss rate, but also the replacement distance.

Since a victim evicted from the L1 cache is sent to the L2 cache of the same node, we do not have to consider the replacement distance of the victim. Only when a block is replaced in the L2 cache does its replacement distance matter. Note that a long-distance victim block may increase the miss time and even delay traffic that delivers another missing block on the interconnection network. However, it is not feasible to build a replacement policy that considers both how long the block has not been used and how far it has to travel together. Instead, the *LRU/SDF*(*Shortest Distance First*) *combined L2 cache* is proposed in this paper. Its LRU cache that runs the LRU replacement policy still favors recently used blocks. Its SDF cache keeps the blocks that have to go a long distance even though they have not been used recently, since it first replaces the block that has the shortest distance. Note that the SDF cache is still another set-associative cache that uses the *SDF replacement policy* to find a victim block. The SDF cache operates just like a victim cache proposed by Jouppi [7], since it keeps blocks that were discarded by the LRU cache. However, it differs from the original victim cache in that it is as big as its victim-provider and that it is simultaneously accessed with its victim-provider. Each block of the SDF cache is expected to have information on location instead of age information. In summary, the combined cache is expected to take advantage of both replacement policies by separating the hardware unit into two.

The rest of this paper is organized as follows. In section 2, we present the underlying memory hierarchy, including the distance-aware second-level cache, and explain the SDF replacement policy in detail. Section 3 presents the simulation methodology and the application programs used in our simulations. Section 4 and section 5 present our simulation results and related work. Section 6 concludes this paper.

## 2. PROPOSED SECOND-LEVEL CACHE

Fig. 1 shows a block diagram of the memory hierarchy with the cache hierarchy emphasized. An LRU/SDF combined cache that is composed of an LRU cache and an SDF cache together works like a traditional L2 cache. If a block is not found in the L1 cache (miss), the cache controller will try to find the block in the LRU/SDF cache instead of going to the local memory or interconnection.

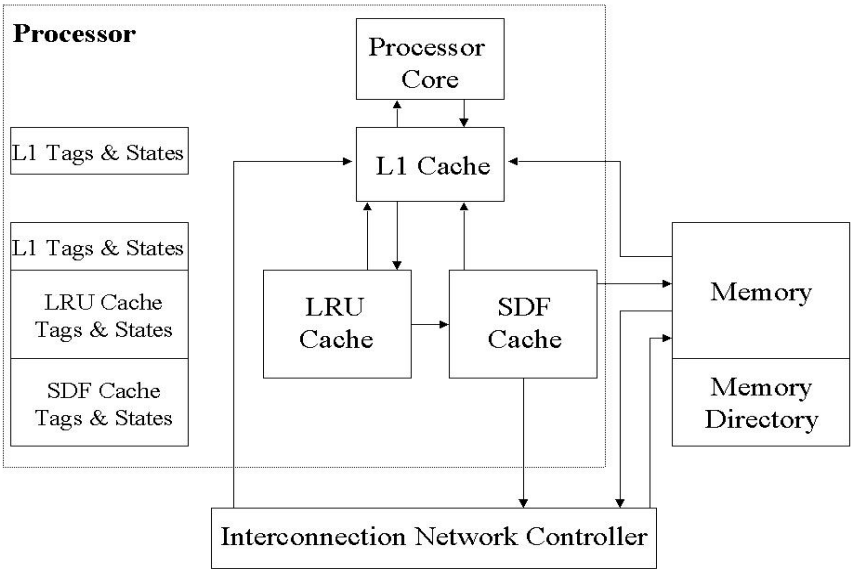


Fig. 1. Block diagram of the memory hierarchy with the proposed second-level cache.

In the traditional two-level cache hierarchy, a cache block can exist in more than one level at a given time. This *inclusion property* makes it easy to maintain the memory hierarchy, but it sometimes wastes cache capacity by duplicating data. We do not keep the inclusion property between the L1 cache and L2 cache as studied in [11]. Instead, we keep a duplicate copy of the tags and states of the L1 cache in the L2 cache controllers. An L1 cache miss that cannot be resolved in either the LRU cache or SDF cache, therefore, is filled directly from local memory or interconnection without allocating a block in the L2 caches. When a block is not found in the L1 cache (miss) but is found in the L2 cache (hit), a victim of the L1 cache is moved to the L2 cache, while the found block is used to fill the empty L1 cache block. It is known that no duplication between the L1 cache and L2 cache can result in better utilization of the cache capacity.

Upon a cache miss in the L1 cache, the cache controller accesses the LRU cache and the SDF cache at the same time. The SDF cache access time is exactly the same as the LRU cache access time since the SDF cache is just a set-associative cache like the LRU cache. If the block is not found in the combined L2 cache, the cache controller sends a memory access request to either the local memory or interconnection.

As shown in Fig. 2, we have four cases with operations of the LRU/SDF combined cache: LRU cache hit, SDF cache hit, LRU/SDF cache miss and L1 coherence miss. A cache miss in the L1 cache issues a request to the LRU cache and the SDF cache at the same time. When the data block requested by the L1 cache is found in the LRU cache (case (a)), the cache block in the LRU cache is moved into the L1 cache. If the cache miss in the L1 cache results from a capacity miss or a conflict miss, the victim of the L1 cache is moved out to the LRU cache. Otherwise, we do not find any victim in the L1 cache, resulting in no replacement.

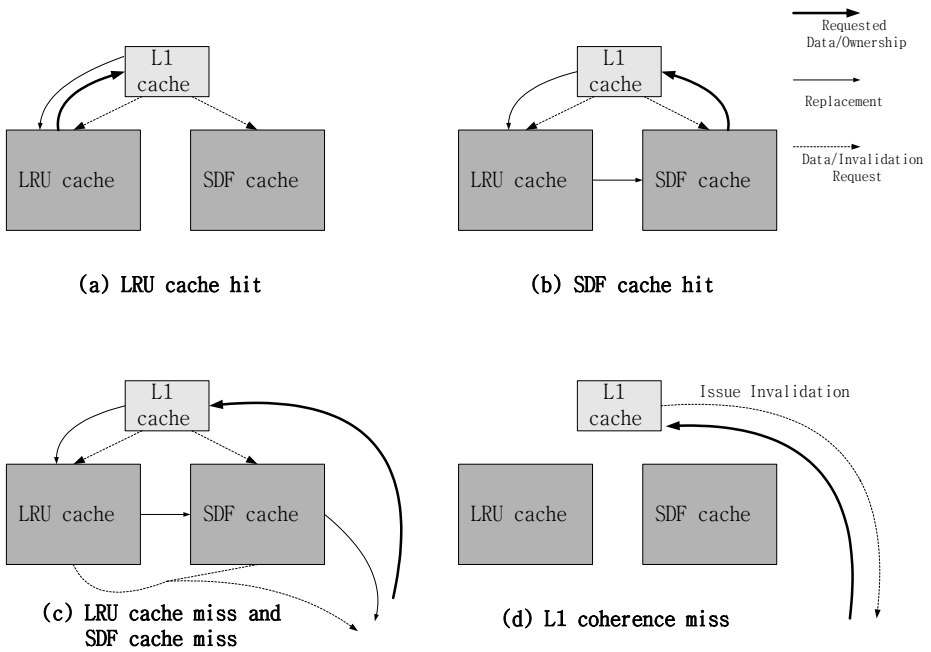


Fig. 2. Operations of the LRU/SDF cache.

When the data block requested by the L1 cache is found in the SDF cache (case (b)), the cache block in the SDF cache is moved into the L1 cache. A victim block in the L1 cache should be moved into the LRU cache if the cache miss in the L1 cache is a capacity miss or conflict miss. When the L1 cache selects a victim, it still observes the LRU replacement policy. If we have to select a victim in the LRU cache to make room for the victim from the L1 cache, the victim in the LRU cache is moved into the SDF cache. There must be room for the block in the SDF cache since a cache block has been already moved to the L1 cache. Fig. 2(b) shows a cycle from the L1 through the LRU cache to the SDF cache. Note that we do not have to consider the case where the hit happens in the LRU cache and SDF cache simultaneously since neither cache has duplicate copies. Note that case (b) does not generate any traffic on the interconnection network at all. It is the SDF cache that provides the requested block when the L2 LRU cache fails to hit, thus minimizing the replacement cost.

If the requested block is not found in either the LRU cache or the SDF cache (case (c)), the memory has to provide the block to the L1 cache directly. The requested data block is stored in the L1 cache, and we may have to select victims in the L1 cache and the LRU cache, respectively, as we did with case (b). Note that we may not have any free space even in the SDF cache when a victim block from the LRU cache finds room. This is because the data block that was moved to the L1 cache had not originated from the SDF cache as it had so in case (b). This case may generate traffic if the requested block is not found in the local memory. Note that only case (c) generates traffic with the LRU/SDF combined L2 cache, while the original L2 cache does so in case (b) as well as case (c).

In the case of an L1 cache coherence miss (case (d)), the requested block is moved to the L1 cache directly from the local memory or the interconnection network. We do not have to update the LRU cache or the SDF cache because the inclusion property is not enforced.

## 3. EVALUATION METHODOLOGY

We used an execution-driven simulator to evaluate the performance of the LRU/SDF replacement policy. The simulator consists of a front-end called MINT [12]. The MINT executes each instruction in one simulation cycle and sends the events generated by memory operations to the architecture simulator, which then performs the operations needed to simulate the system architecture, including replacement policies. To speed up the simulation, we assumed that all instructions hit in the instruction cache, and that all accesses to private data hit in the L1 cache. These assumptions have also been made in many other multiprocessor studies. The simulated systems had 8, 16 or 32 nodes composed of 1 GHz processors. The first-level cache in each processor had a size of 64 Kbytes and was assumed to run the LRU replacement policy.

The traditional LRU cache is 8-way set associative. To make a fair comparison, the LRU cache and the SDF cache in the combined L2 cache are 4-way set associative to reflect the fact that both caches are accessed simultaneously. The block size was 32 bytes long.

We maintained a 2-bit state field per cache block, corresponding to the MSI protocol, which is modified from a typical MESI protocol. It takes 80 ns for a processor to access the local memory. The simulated system interconnected nodes by means of dual-ring style point-to-point links, and it was assumed to run the full map directory protocol. The interconnection between the processor nodes was 16 bits wide and was clocked at 500 MHz. The preparation time for interconnection was 20 ns. We used four applications, FFT, LU, OCEAN and RADIX, from the SPLASH-2 benchmark suite [6]. We did not assume any intelligent data distribution. The characteristics of the benchmark programs are summarized in Table 1.

**Table 1. The characteristics of the benchmark programs.**

| **Application** | Type | Problem Size | Write Access Ratio |
|---|---|---|---|
| FFT | FFT computation | 65536 complex doubles | 39 % |
| LU | Blocked dense linear algebra | 512X512 matrix, block 16 | 33 % |
| OCEAN | Study of ocean movements | 130X130 ocean grid | 18 % |
| RADIX | Radix sort | 524288 integer key, radix 1K | 37 % |

## 4. SIMULATION RESULTS

Using the simulation setup described above, we compared our LRU/SDF cache with the traditional LRU cache for the L2 cache. We investigated the cache miss rate, cache miss time and replacement cost, varying total cache size and the number of processors. The cache miss rate was obtained only from the L2 cache (LRU/SDF combined cache or traditional LRU cache), excluding the L1 cache miss rate. The cache miss time was also derived only from the L2 cache miss, and we did not consider contentions. All the results are shown normalized to the results of the 128KB LRU cache to ease the understanding.

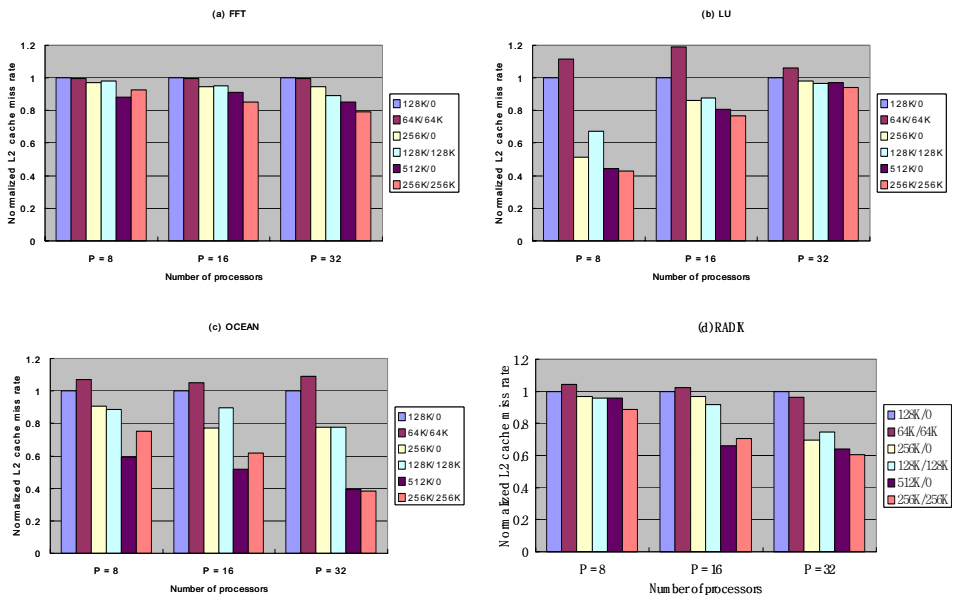


Fig. 3. Normalized L2 cache miss rate (LRU cache size/SDF cache size).

### 4.1 Cache Miss Rate

Fig. 3 shows the effect on the L2 cache miss rate of the total cache size and the number of processors. The results are normalized to the cache miss rate of the 128KB LRU cache (leftmost bar). When the number of processors was 8, the cache miss rate of the 128KB LRU cache was lower than that of the 128KB LRU/SDF combined cache, in most cases. Note that the 128KB LRU/SDF combined cache was composed of a 64KB LRU cache and another 64KB SDF cache. The results confirm the observation that a 64KB LRU cache is insufficient to keep all the cache access patterns. This implies that larger LRU cache is required to make use of the cache access patterns efficiently. When the L2 cache size was 512KB, the miss rates of the LRU/SDF combined cache were lower than those of the LRU cache in most cases. As the number of processors increased, the LRU/SDF combined cache outperformed the LRU cache even with small

cache sizes because a fixed size problem was distributed into smaller partial problems over processors. The reduction of the cache miss rate was up to 6%, comparing the LRU/SDF combined cache and the LRU cache with the same size. In FFT and RADIX, the LRU/SDF combined cache performs better than the LRU cache in most cases.

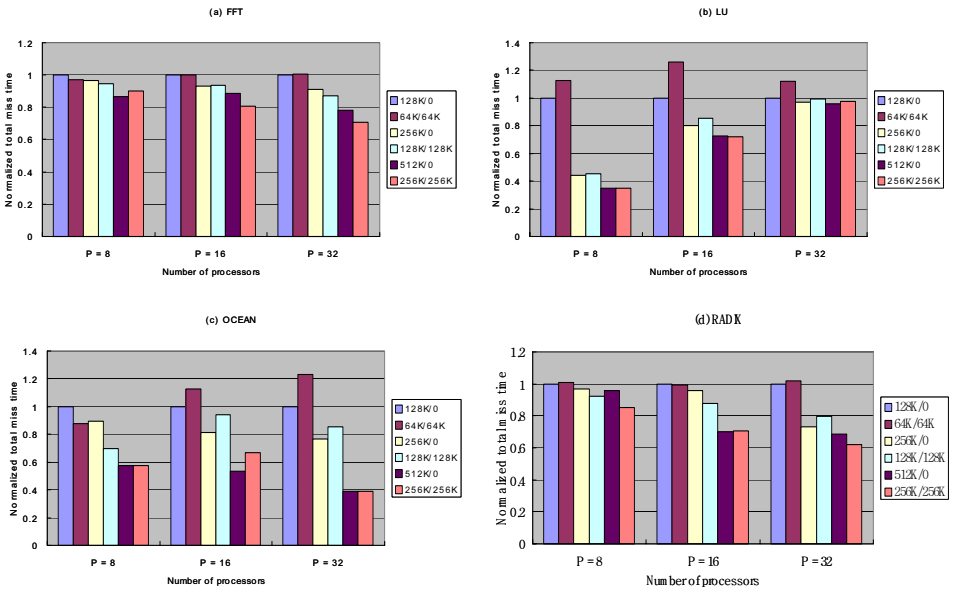


Fig. 4. Normalized total cache miss time (LRU cache size/SDF cache size).

### 4.2 Cache Miss Time

The normalized total cache miss time, presented in Fig. 4, is the total elapsed time between the time when an L2 cache miss occurs and the time when the requested block is filled in the L1 cache. Compared to the total cache miss time of the LRU cache, the total cache miss time of the same size LRU/SDF combined cache was up to 9% less. However, when we exclude the gains from the cache miss rate, the reduction is at most 3%.

Before running the simulations, we thought that the LRU/SDF combined cache would reduce the total cache miss time significantly. For example, assume that the replacement distances of blocks A and B are 1 and 4 units, respectively, and that we have only one block that keeps either A or B. If block A has recently been used, we have to keep block A with the LRU cache, and it costs (0 + 4)/2 = 2 units to access both blocks. With the SDF cache, block B is kept due to its long replacement distance, which costs (1 + 0)/2 = 0.5 unit.

In the applications, however, the number of cache hits the SDF cache contributes was found not particularly large. Thus the gains from the reuse of cache blocks in the SDF cache were not significant, leading to a small reduction in the total cache miss time.

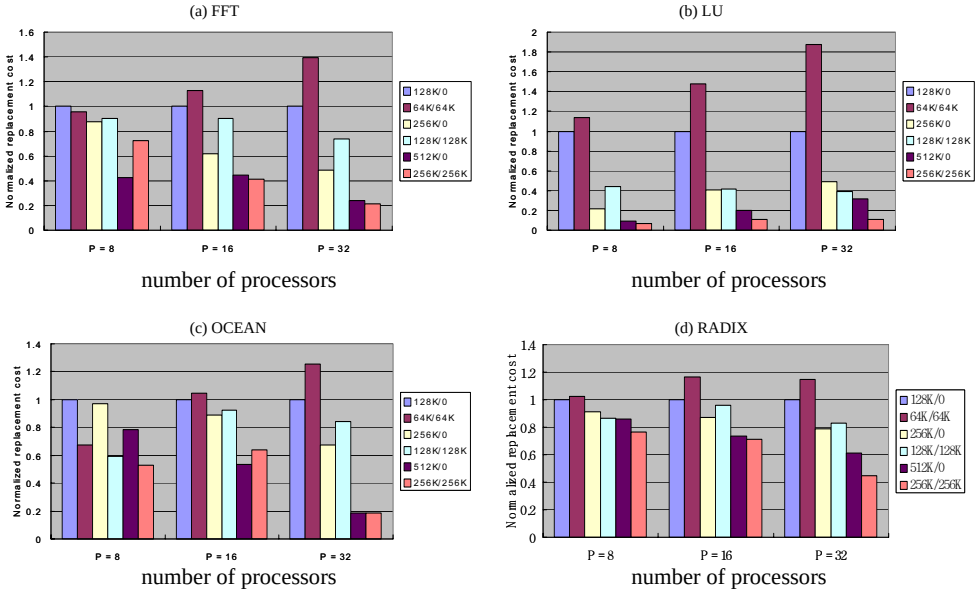


Fig. 5. Normalized replacement cost on the interconnection network. (LRU cache size/SDF cache size).

### 4.3 Replacements Cost

The replacement cost is defined as the sum of estimated elapsed times when the interconnection network is busy delivering data blocks for replacement, not including the resource conflicts. Most processors have a write-back buffer [1, 13], where victim blocks being replaced can be temporarily stored before the actual write-back has been completed. A reduction of the replacement cost, therefore, does not affect the cache miss time directly. However, the replacement cost on the interconnection network, shown in Fig. 5, is one of the major sources of network traffic. Heavy traffic on the interconnection network increases the probability of delaying data block delivery service. Therefore, system performance may improve if the replacement cost on the interconnection network is reduced.

The replacement cost is directly affected by the SDF policy. The replacement cost with the LRU/SDF combined cache, whose size was 256K/256K, was found to be significantly smaller than that with the LRU cache whose size is 512K by up to 64%. This is because the LRU/SDF combined cache replaced the shortest distance cache block using the SDF policy.

As the number of processors increased, the number of alternatives also increased. This led to more reduction of the replacement cost. With small sized caches, however, we found that the LRU/SDF combined cache had a higher replacement cost than the LRU cache. This was due to the higher cache miss rate resulting from the limited space for the LRU cache in the LRU/SDF combined cache. If considering the resource conflict on the interconnection network, the gains must be much larger.

## 5. RELATED WORKS

To our knowledge, hardware-based replacement algorithms with distance information in the context of multiprocessors have not been proposed. To compare our work with others, we have presented a hardware-based replacement algorithm for multiprocessors, a software-based replacement algorithm with distance information and a cache organization to reduce the remote memory latency and traffics on the interconnection network.

Toussi and Lilja presented a new replacement policy, in which replacement priority is assigned to each cache block within a set based on its state [8]. They also showed that the proposed replacement policy could outperform the random replacement policy and in some cases could even outperform the LRU replacement policy. Jeong and Dubois proposed a CSOPT (cost-sensitive optimal replacement algorithm) [9], which is based on the OPT (OPTimum) replacement algorithm. However, as stated in the paper, CSOPT is unrealizable in real systems. In order to reduce the remote memory latency and traffic on the interconnection network, Iyer and Bhuyan introduced the Switch Cache [10]. The main idea is to implement small fast caches in crossbar switches of the interconnect medium to capture and store shared data as it flows from the memory module to the requesting processor. They found that the switch cache was capable of improving the performance of multiprocessors by reducing the number of reads served at distant remote memories by up to 45%. All the above-mentioned works were targeted at reducing the cache miss time and traffic on the interconnection network in multiprocessor systems.

## 6. CONCLUSIONS

We have proposed a new hardware-based cache replacement policy for L2 caches which reduces the cache miss rate and cache miss time together and which minimizes the replacement cost on the interconnection network for a multiprocessor system. The suggested LRU/SDF combined cache improves the performance by keeping long-distance blocks as well as recently used blocks in two separate caches.

We have shown that the performance of the suggested LRU/SDF combined cache is better than that of the original LRU cache unless we have a limited amount of space for the L2 cache. As far as the replacement cost is concerned, the suggested L2 cache achieves reduction of up to a 64%. As the number of processors increases and the cache size also increases, we can expect even more performance improvement with the proposed L2 cache. Therefore, the LRU/SDF combined cache is believed to provide more scalability for multiprocessor systems with large-scale caches by reducing network traffic.

We plan to evaluate the effects on the total execution time soon. We also plan to evaluate the system performance with various interconnection networks, including bus and hierarchical structures.

## ACKNOWLEDGEMENTS

We would like to thank Craig B. Stunkel at the IBM T. J. Watson Research Center for his helpful comments.

## REFERENCES

1. D. Patterson and H. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., 1995.
2. Y. Deville, "A low-cost-based replacement algorithms for cache memories," *Computer Architecture News*, 1990, pp. 62-58.
3. E. D. Granston and A. V. Veidenbaum, "An integrated hardware/software solution for effective management of local storage in high-performance systems," *International Conference on Parallel Processing*, 1991, pp. 83-90.
4. K. So and R. N. Rechtschaffen, "Cache operations by MRU chance," *IEEE Transaction on Computer*, Vol. 37, 1988, pp. 700-707.
5. D. Burger, J. R. Goodman, and A. Kagi, "Limited bandwidth to affect processor design," *IEEE Micro*, 1997, pp. 52-62.
6. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," *International Symposium on Computer Architecture*, 1995, pp. 24-36.
7. N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *International Symposium on Computer Architecture*, 1990, pp. 388-397.
8. F. M. Toussi and D. J. Lilja, "The effect of using state-based priority information in a shared-memory multiprocessor cache replacement policy," *International Conference on Parallel Processing*, 1998, pp. 217-224.
9. J. Jeong and M. Dubois, "Optimal replacements in caches with two miss costs," *ACM Symposium on Parallel Algorithms and Architectures*, 1999, pp. 155-164.
10. R. Iyer and L. N. Bhuyan, "Switch cache: A framework for improving the remote memory access latency of CC-NUMA multiprocessors," *International Symposium on High Performance Computer Architecture*, 1999, pp. 152-160.
11. L. A. Barroso, et. al., "Piranha: a scalable architecture based on single-chip multiprocessing," *International Symposium on Computer Architecture*, 2000, pp. 282-293.
12. J. E. Veenstra and R. J. Fowler, "MINT: A front end for efficient simulation of shared-memory multiprocessors," *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 1994, pp. 201-207.
13. David E. Culler, et. al., *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers Inc., 1999.

**Sung Woo Chung** is a Ph.D. candidate in the School of Electrical Engineering and Computer Science at Seoul National University in Korea. His research interests include

multiprocessor system, computer architecture and real-time system. He received BSE and MSE in Computer Engineering from Seoul National University in Korea. He was an academic visitor at the IBM T. J. Watson Research Center from Jan. 2002 to July 2002.

**Jeon-Heon Shin** is a Ph.D. candidate in the School of Electrical Engineering and Computer Science at Seoul National University in Korea. His research interests include multiprocessor system and computer architecture. He received BSE and MSE in Computer Engineering from Seoul National University in Korea.

**Hyong-Shik Kim** is currently an assistant professor in the department of Computer Science at Chungnam National University in Korea. His research interests include computer system architecture, high-performance computing, internet architecture, and performance measurement. He received BSE, MSE, and Ph.D. degrees in Computer Engineering from Seoul National University in Korea. He is a member of the IEEE Computer Society, the ACM, and the KISS (Korea Information Science Society).

**Chu Shik Jhon** is a professor in the Electrical Engineering and Computer Science at Seoul National University in Koera. His research interests include VLSI/CAD and parallel computer architecture. He received his B.S. (1974) degree in Applied Mathematics from Seoul National University, his M.S. (1976) degree in Computer Science from Korea Advanced Institute of Science and Technology, and his Ph.D. (1982) degree in Computer Science from University of Utah. From 1982 to 1984, he worked as a faculty member of University of Iowa.