Relaxing the Inclusion Property in Cache Only Memory Architecture

Jinseok Kong[1] and Gyungho Lee[2]

[1] Department of Computer Science
[2] Department of Electrical Engineering
University of Minnesota, 200 Union St. SE., Minneapolis, MN 55455, USA

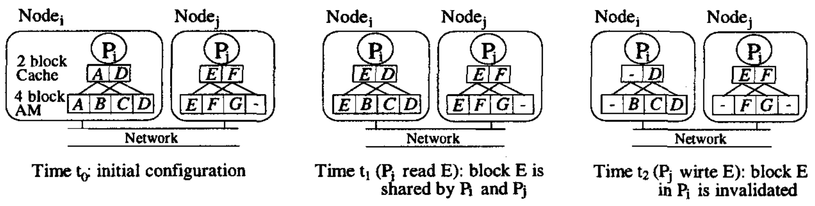
Abstract. Cache only memory architecture (COMA), even with its additional memory overhead, can incur longer inter/intra-node communication latency than cache-coherent nonuniform memory access (CC-NUMA). Some studies on COMA suggest that the inclusion property applied between the processor cache and its local memory is one of the major causes of less-than-desirable performance. The inclusion property creates extra accesses to the slow local memory. We consider the binding time of data address to the local memory to be an important factor related to the long latency in COMA. This paper considers the inclusion property in COMA and introduces a variant of COMA, dubbed Dynamic Memory Architecture (DYMA), where the local memory is utilized as a backing store for blocks discarded from the processor cache. Thus, by delaying the binding time, the long latency due to the inclusion property can be avoided. This paper examines the potential performance of DYMA compared to COMA and CC-NUMA.

1 Introduction

Cache only memory architecture (COMA) [3, 6] treats the memory local to each node, called *attraction memory* (AM)[3], as a cache to the shared address space without providing traditional main memory [6]. In cache-coherent nonuniform memory access (CC-NUMA) [11], the local memory is utilized as a portion of the space. COMA is similar to SVM (Shared Virtual Memory), which allows sharing of virtual memory space through migration and replication of pages [4, 12], but COMA is a more hardware-oriented approach and the granularity of the sharing unit, the *memory block*, is significantly finer than that of the page used in SVM.

In COMA, the huge size of the AM creates more coherence activity and longer latency for the cache miss accesses [13]. Also, organizing it as a cache requires the overhead of tag storage and additional unallocated space for replicated data. The proposed COMA machines [3, 6] and the studies [7, 13] found in open literature utilize an invalidation policy for coherency, and assert the inclusion property [1] applied between the processor cache and its AM [9]. By broadcasting invalidation signals [15] to the processor cache when an AM block is replaced or

[3] Although KSR-1 [3] and another COMA proposal DDM [6] use different terminology for various aspects of COMA, we generally follow the terminology of the DDM.



Node$_i$
Node$_j$
2 block Cache
4 block AM
P$_i$
P$_j$
A D
E F
A B C D
E F G -
Network
Time t$_0$: initial configuration
E D
E B C D
Time t$_1$ (P$_i$ read E): block E is shared by P$_i$ and P$_j$
- D
- B C D
- F G -
Time t$_2$ (P$_j$ wirte E): block E in P$_i$ is invalidated

Fig. 1. An example of ill-effects of the inclusion property

by attaching an inclusion bit [1] to each AM block entry, the inclusion property can be enforced. However, both of these schemes generate additional intra-node communication. Because of this, some studies [7, 13] on COMA consider relaxing the inclusion property somewhat to improve performance.

We propose a variant of COMA, which we call *dynamic memory architecture* (DYMA), where the local memory is utilized as a backing store for the blocks discarded from the processor cache, thereby delaying the address binding of data to the local memory from ***block-incoming*** time to ***block-discarding*** time. Delaying the address binding relaxes the inclusion property, and this allows faster data access.

Section 2 describes background materials which motivated our work. Section 3 introduces DYMA along with a coherence control. The memory access latency of DYMA is compared to those of COMA and CC-NUMA in Section 4. Section 5 reports our preliminary simulation results. Section 6 offers a conclusion.

2 Background

The inclusion property in COMA causes another overhead: frequent accesses to the AM. For example, consider the simple configuration shown in Fig. 1. If processor P_i reads the shared block E at time t_1, the block A in Node$_i$'s AM is replaced by block E to maintain the inclusion. At this time, accesses to Node$_i$'s AM are needed to store block E. Also, note that if block A is the last copy in the system, two more AM accesses are needed to relocate block A (read block A from Node$_i$'s AM and save it at a remote AM). If processor P_j writes data on block E at time t_2, the copies of block E in Node$_i$ are invalidated in the write-invalidation policy. To invalidate block E of P_i's AM, access to the AM tag is necessary. By allowing direct access to the processor cache for the incoming message from the network, we can hide these extra AM accesses from the critical path of the memory access latency. However, the extra AM accesses can still make other memory access latency longer due to contention at the AM. Also, note that probing the processor cache is unavoidable in the write-back policy, since the up-to-date copy can exist only in the processor cache.

If we allow non-inclusive memory access in COMA, the extra AM accesses can be reduced. Let's consider the example again. If we relax the inclusion property for block E at time t_1, block E is saved only in the cache of Node$_i$, which removes the extra AM accesses to save block E and to relocate block A. Also, the access to the AM tag is not necessary when block E is invalidated at time t_2. Further, if block A is referenced again in Node_i, the node hit rate increases. This saves memory space and reduces extra accesses to the AM. However, this requires predicting future behavior of a block to determine whether the address of a block should be bound to a local memory frame or not. To relax the inclusion property in COMA, the prediction is required when the block is brought to the node (block-incoming time). Binding the address of a block which will be swapped or invalidated to an AM frame at block-incoming time may waste AM space. If the address of a block which will not be invalidated in the near future is not bound to an AM frame, the block can easily be replaced from the node due to the small size of the processor cache. Thus, either way limits the effective utilization of the AM, thereby diminishing the advantage of COMA.

3 Dynamic Memory Architecture (DYMA)

In DYMA, the local memory of each node has the facility of dynamic address binding to a memory block frame supported by hardware as in COMA. But DYMA uses the local memory as the backing store for the blocks discarded from the processor cache, i.e., binding the address of a block to the local memory happens at block-discarding time. Using the local memory in this way, no write access which hits on a non-shared cache block has to go to the local memory. Also, a node miss can be serviced directly from a remote cache and saved directly to its cache without accessing the slow local memory. Further, between the block-incoming time and the block-discarding time, the block of the cache can be invalidated, or the block of the local memory can be loaded again. The first case reduces the local memory accesses and global replacement traffic, and the second case increases the node hit rate. These properties can make the average latency of DYMA short while still allowing dynamic data replication and migration from node to node as in COMA.

In CC-NUMA, the data address is bound to a local memory frame when a page is loaded. COMA delays the address binding from page-fault time to memory block missing time. We call the local memory of DYMA *dynamic memory* (DM). An interesting aspect of DYMA is that there is no inclusion property between the processor cache and the DM. DYMA utilizes the processor cache and the DM for different functionality, while the processor cache is somewhat redundant in COMA. If the sum of local memories and the size of the working set of an application are the same, DYMA allows a block to be replicated to other nodes, while COMA allows no block replication.

The relationship between the AM and the DM is similar to the relationship between the *miss cache* and the *victim cache* [8]; the address of the miss cache is bound at the block-incoming time to the lower level cache as in COMA, while the address of the victim cache is bound at the discarding time from the lower level cache as in DYMA. The miss and victim caches are designed to reduce cache misses in uniprocessors, while the AM and the DM are designed to reduce node misses in multiprocessors.

3.1 Memory Access Mechanism

The memory access mechanism of DYMA described in this section assumes a directory-based write-invalidation protocol with an interconnection network. By adding one more bit, a *local bit*, in each cache block entry, we developed a scheme where only an indispensable block, when discarded from the cache, is written to its DM. The local bit is used to indicate whether the copy of the block is in its DM (local block) or not (global block). All global blocks are saved in its DM when they are replaced at the cache. If a local block is selected for replacement and modified at the cache, it must be written in the DM like the traditional write-back cache. The use of the local bit and the memory access mechanism can be described in seven steps:

Cache access step: If a write hits on a non-shared block or a read hits, give the data to the processor. If a write hits on a shared block, go to **invalidation step**. Go to **remote access step** directly if a cache coherence miss happens; otherwise, go to **DM access step**.

DM access step: If a write hits on a non-shared block or a read hits, go to **cache fill step**. If a write hits on a shared block, go to **invalidation step** and then **cache fill step**. If a DM miss occurs, go to **remote access step**.

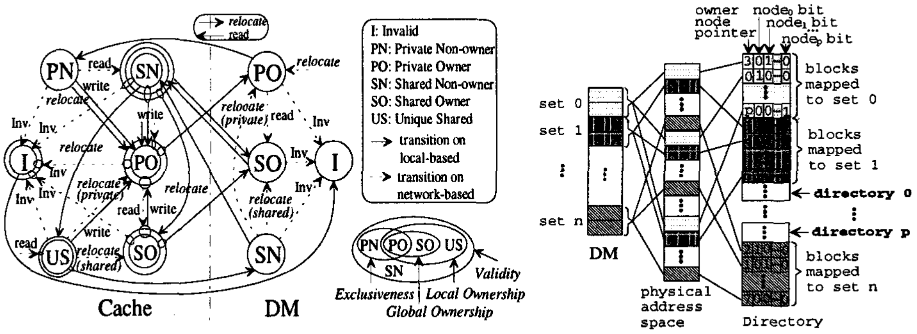
Remote access step: Find the owner node of the block at its home directory, and send a block request message to the node. At the owner node, if the block exists at the cache, copy the block from the cache. If the block exists in the DM, not in the cache, copy it from the DM. Deliver the copy to the requesting node, and go to **cache fill step**.

Invalidation step: At the home directory, send an invalidation signal to the nodes which have any copies of the block to be invalidated. At the remote node, invalidate the block if it is at the cache or at the DM but not in the cache, and send back an acknowledgment signal to the requesting node.

Cache fill step: Fill the missing block into the cache and start processing. At this time, set the *local bit* if the block is from its DM or from the network due to the cache coherence miss of a local block; otherwise, reset the local bit. If there is no space to save the block, select a block to be replaced. Here, any replacement algorithm can be used. If the selected block is a global or modified block, go to **relocating cache block step** to save the block at the DM. The relocation action is overlapped with the running of the processor.

Relocating cache block step: If a block replaced in the DM in order to save the cache block is the unique copy in the node and has ownership, go to the home directory of the DM block, select a node which can keep ownership of the block, and go to **relocating DM block step** for relocating the DM block at the remote node. When selecting a node at the home directory, try to find a node with the following priority sequence: (1) a node which has the same copy of the block, (2) a node which has unused space to save the block, and (3) a node which contains a shared non-owner block.

Relocating DM block step: If the same block exists in the cache or if the same block is in the DM but not in the cache, just change the state to owner. If there is an invalid block or a different, shared non-owner block in the DM, save the owner block at that place.



(a) coherence protocol (b) directory structure

Fig. 2. Coherence and relocation controls for DYMA.
In the directory structure, the DM is 2-way set associative. The number of DM sets is n, and the number of nodes is p.

3.2 Coherence and Relocation Controls

The coherence protocol of DYMA is an optimized one with twelve states in each cache block and four states in each DM block (Fig. 2a). The *local bit* described in the previous section is combined with the state bits. When a page is first loaded from secondary storage, all blocks of the page are saved in a private owner (PO) state. When a read access hits at the cache, there is no state change. When a block is copied from its local DM for a read access, the block becomes private non-owner (PN) if its DM state is PO; otherwise it becomes shared non-owner (SN). When a block is copied from a remote node for a read access, the block is loaded into the cache in a unique shared (US) state. At this time, the state of the remote node becomes shared owner (SO). The write access makes the cache block PO-state while all other copies of the block become the invalid (I) state.

If a PO-/SO-state block is selected for replacement, the block is saved at its DM with the same state. If a US-state block is selected, the block is saved in the DM with SN-state. If a SN-(from PN)/I-state local block is selected, the DM block becomes SO-/I-state. If a copy of the DM block replaced by a global cache block exists in its cache with PN-/SN-state, the state of the cache block changes to the global PO-/(US- or SO-)state. When a DM block is relocated at a remote node, it becomes SO-state if its copy exists at more than one node. Otherwise, the block is relocated with PO-state.

DYMA possesses an overhead related to memory block relocation as COMA does. If the replaced memory block is the last valid copy in the system, the block must be relocated to some other node. To reduce the overhead of the DM block relocation in DYMA, we use a new directory structure (Fig. 2b) where each home directory has information of all the blocks which are mapped to the same DM set. Therefore, just by checking the home directory of a block to be relocated, the node which has space for the block can be chosen.

Frequent accesses to the cache state in DYMA may impede the processor's access to the cache. Thus, one may want to provide the controller with a duplicate copy of both cache tag and state; this will shield the processor's access to its own cache from the interference of unsuccessful attempts to match cache tag and state by remote accesses and invalidations. If the copy from the cache can be provided much faster than the copy from the DM, the copy from the cache is favorable in terms of total system performance because it decreases the stall time of processors generating the remote access.

4 Memory Latency Comparison

To compare the performance of CC-NUMA, COMA, and DYMA in terms of latency, we modeled memory access latency for each architecture. We divided memory accesses into four different cases; *local-cache hit*, *local-memory hit*, *remote-cache hit*, and *remote-memory hit*. Let us say that L_{lc}, L_{lm}, L_{rc}, and L_{rm} are latencies for a local-cache hit, a local-memory hit, a remote-cache hit, and a remote-memory hit respectively. Then, the *average* latency per reference can be expressed as follow:

$$H_{lc}L_{lc} + M_{lc}H_{lm}L_{lm} + M_{lc}M_{lm}H_{rc}L_{rc} + M_{lc}M_{lm}M_{rc}L_{rm}, \tag{1}$$

where H_{lc} ($= 1 - M_{lc}$) is the local-cache hit *rate*, H_{lm} ($= 1 - M_{lm}$) is the local-memory hit *rate*, and H_{rc} ($= 1 - M_{rc}$) is the remote-cache hit *rate*.

Fig. 3 shows the latencies for each individual memory access in the memory architectures. When a local-cache hit occurs in COMA, there are two possible courses of action: (i) write access to a non-shared block or read access; or (ii) write access to a shared block. In the first case, the latency is the time for cache access (t_c). In the second case, coherence action is necessary. This coherence action may need three network traversals ($3t_n$), directory processing time (α), and the invalidation time at a remote node (t_c or $t_c + t_a$).

Although, the latencies for each memory architecture are from a simple analysis of the memory access mechanism, they nevertheless show inherent latency characteristics. The average latency of CC-NUMA can be the shortest only when the partition and distribution of data and the page migration [5] are done so well that the local-memory hit rate, H_{lm}, becomes close to that of DYMA. In general, the H_{lm} of DYMA and COMA is expected to be higher than that of CC-NUMA. With higher H_{lm}, even though the average remote latency can decrease both in DYMA and COMA, the coherence action in L_{lm} and the memory tag access time (t_a) can make the average overall latency longer. Thus, the cut-off point depends on how much higher the H_{lm} of DYMA and COMA is than that of CC-NUMA, given the network latency.

By relaxing the inclusion property in COMA, the extra AM accesses can be reduced. However, generally it seems to be hard to determine whether the address of a block should be bound to the local memory or not, without knowing the future behavior of the block. DYMA can reduce the extra accesses by delaying the address binding while increasing the H_{lm}.

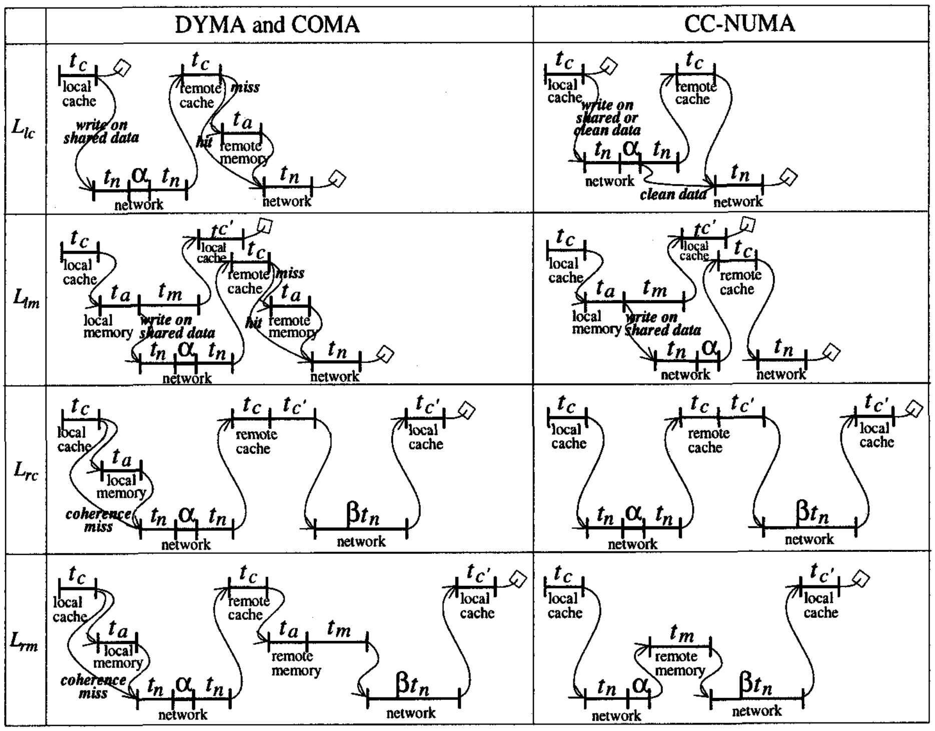


Fig. 3. Memory access latency in each architecture.
In the figure, t_c = cache access time, $t_{c'}$ = cache fill time, t_m = memory access time, t_a = memory tag access time, t_n = network traversal time without data, α = processing time at the directory, βt_n = network traversal time with data.

5 Preliminary Simulation Results

We developed a trace driven simulator using the parallelized Perfect Club Benchmark programs (Table 1a) [2] as workloads. Although our simulations were done with scaled-down data size and accordingly scaled-down cache and local memory size, they nevertheless demonstrate the potential advantage of DYMA.

We assume that instruction access is overlapped with data access. The requests for the use of resources are serviced in first-in first-out order. The processor which arrives at the barrier synchronization early is blocked until the last processor arrives at the synchronization point. The master processor (processor number 0) executes sequential portions of a program. The default simulation parameters are shown in Table 1b. We use the optimized coherence protocol [10, 14] for each architecture. We made the memory access time relatively large because we believe that the processor-memory performance gap is widening, not narrowing. Here, note that the tag of the memory is assumed to be slow DRAM.

We use the average memory access latency described in Section 4 (Eq. 1) as the performance metrics. The waiting time caused by resource contention is also considered. The *network traffic rate* is defined as the average number of network

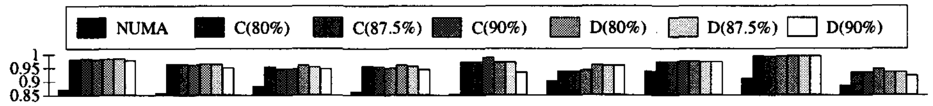
Code Name	Total Ref. (write %)	Data Sz (words)
ADM	3802575(20.2)	17255
ARC2D	60158056(15.7)	899484
DYFESM	6028514(13.9)	16451
FLO52Q	72824086(14.9)	25763
MDG	107182698(19.8)	29299
OCEAN	17254748(21.7)	1204
QCD2	31078266(16.0)	32971
TRACK	2773985(16.0)	21537
TRFD	9446379(14.6)	14950

(a) characteristics of the traces

Unit	Parameters	CC-NUMA	COMA	DYMA
coherence protocol		5 states	4 states	12 & 4 states
cache	size = 512 bytes, access time(t_c) = 1, fill time ($t_{c'}$) = 4, associativity = direct-mapped			
memory	(total) size	data size	$\frac{\text{data size}}{1-1/(\#\text{ of nodes})}$	
	associativity	-	direct mapped	
	access time (t_m) = 25, tag time (t_a) = 16			
network	protocol	directory-based		
	service time (t_n) = 10, $\alpha = 0$, $\beta = 3$			
block size = 16 bytes, # of nodes = 8				
Here, time unit is clock cycles				

(b) default parameters

Table 1. Simulation environments.



ADM ARC2D DYFESM FLO52Q MDG OCEAN QCD2 TRACK TRFD

Fig. 4. Node hit rate per reference.

traversals per memory reference. The three main components of the network traffic are data traffic, replacement traffic, and coherence traffic [10].

Fig. 4 is the node hit rate of the master node when 8 processors are used. Usually, the node hit rates of COMA and DYMA decrease as the memory pressure (data size / memory size) increases from 80% to 90%. However, at higher memory pressure in COMA, the node hit rates of some traces such as MDG and TRFD are high. This is because the relocation protocol giving high priority to the master processor in COMA allows effective block migration.

The results in Fig. 5 include the traffic generated by all 8 nodes. If the memory pressure is high and the associativity of the DM is low, COMA would like to generate replacement traffic whenever there is a node miss [7]. This makes the replacement traffic of COMA much higher than that of the others. Although not all node misses cause replacement traffic in CC-NUMA, the write-back from the processor cache frequently does. In DYMA, the replacement action at the DM is delayed until there is a relocation request from its cache. This makes the replacement traffic of DYMA lowest. The coherence traffic rate in COMA and DYMA is higher than in CC-NUMA.

The average latencies of DYMA are shorter than the others in most of the traces (Fig. 6) since the high node hit rate of DYMA and small DM accesses reduce the large overhead of remote access and replace it with the small overhead of local access. Although the remote latency of COMA becomes smaller than that of CC-NUMA, the frequent contention at the AMs makes the waiting time for an AM access in COMA longer. In a large (relative to our simulation workloads) processor cache (512 K bytes), the advantage of dynamic address mapping in the local memory becomes small and the contention at the AM of COMA shows more pronounced ill-effects (Fig. 7).

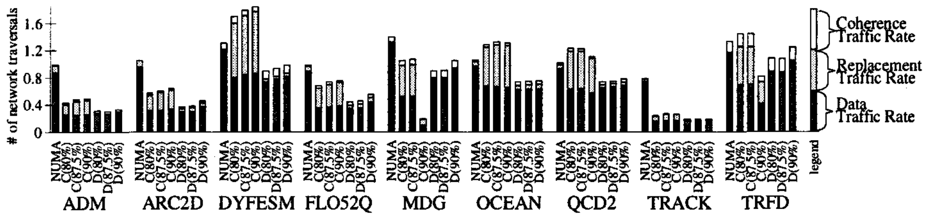


Fig. 5. Network traffic rate per reference.

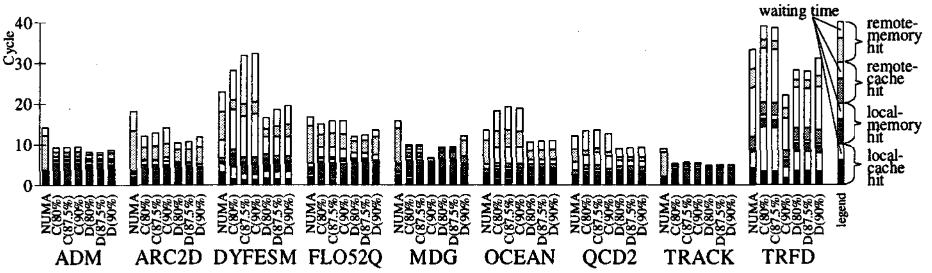


Fig. 6. Average latency (including the waiting time) per reference.

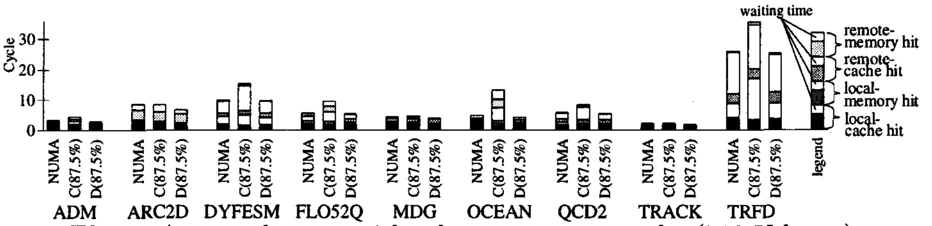


Fig. 7. Average latency with a large processor cache (512 K bytes).

6 Conclusion

COMA allows higher local memory utilization by organizing the local memory as a cache (AM). However, the inclusion property applied between the processor cache and the AM creates extra accesses to the slow AM. Relaxing the inclusion property in COMA requires determining whether a missing block should be bound to a AM frame or not when it is brought to the node (block-incoming time).

This paper proposed a variant of COMA, which we call DYMA, to reduce the latency associated with COMA. The local memory of DYMA, called dynamic memory (DM), is utilized as a backing store for blocks discarded from the processor cache. There is no inclusion property between the processor cache and its DM. By delaying the data address binding to the DM frame from block-incoming time to block-discarding time, the extra accesses to the DM can be reduced while effective utilization of the DM is increased.

We compared the latencies of DYMA, COMA, and CC-NUMA based on simple analysis of the memory access mechanism and based on the results of a simulation. Although DYMA may need more complex hardware control, our preliminary results show that DYMA can provide an opportunity to have relatively small average latency in a large-scale distributed shared-memory multiprocessor system.

Acknowledgments. We would like to thank the members of DICE and HPCC groups at the University of Minnesota and the anonymous referees for discussions, suggestions, and encouragements. This work has been supported by funding from Samsung Electronics.

References

1. J. L. Baer, W. H. Wang, "On the Inclusion Property for Multi-Level Cache Hierarchies," *Proc. of the 15th ISCA*, pp. 73-80, 1988.
2. M. Berry *et al.*, "The Perfect Club Benchmark: Effective Performance Evaluation of Supercomputers," *Int'l Journal of Supercomputing Apps.*, Vol. 3, No. 3, 1989.
3. H. Burkhardt III *et. al.*, "Overview of the KSR-1 Computer System," Technical Report KSR-TR-9202001, Kendall Square Research Corporation, 1992.
4. J. B. Carter, J. K. Bennett, W. Zwaenepoel, "Implementation and Performance of Munin," *13th Sym. on Operating System Principles*, Oct. 1991.
5. R. Chandra *et al.*, "Scheduling and Page Migration for Multiprocessor Computer Servers," *Proc. of the 6th ASPLOS-VI*, Oct. 1994.
6. E. Hagersten, S. Haridi, A. Landin, "DDM - A Cache-Only Memory Architecture," *IEEE Computer*, pp. 44-54, Sept. 1992.
7. T. Joe, "COMA-F: A Non-hierarchical Cache Only Memory Architecture," Ph.D. Dissertation, Stanford University, Mar. 1995.
8. N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. of the 17th ISCA*, pp. 364-373, 1990.
9. G. Lee, "An Assessment of COMA Multiprocessors," *Proc. of the 9th Int'l Parallel Processing Symp.*, Santa Barbara, CA., Apr. 1995.
10. G. Lee, J. Kong, "Prospects of Distributed Shared Memory for Reducing Global Traffic in Shared-Bus Multiprocessors," *Proc. of the 7th IASTED/ISMM Int'l Conf.*, pp. 63-67, Oct. 1995.
11. D. E. Lenoski *et al.*, "The Directory-Based Cache Coherence Protocol for DASH multiprocessor," *Proc. of the 17th ISCA*, pp. 148-159, 1990.
12. K. Li, P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Trans. on Computer Systems*, 7(4):321-359, Nov. 1889.
13. A. Saulsbury, T. Wilkinson, J. Carter, and A. Landin, "An Argument for Simple COMA," *Proc. of 1st IEEE Symp. on High Perform. Comp. Archi.*, 1995.
14. P. Sweazey, A.J. Smith, "A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus," *Proc. of the 13th ISCA*, 1986.
15. A. W. Wilson Jr., " Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors," *Proc. of the 14th ISCA*, 1987.