PAPER

Buddy Coherence: An Adaptive Granularity Handling Scheme for Page-Based DSM

Sangbum Lee[†], **Inbum Jung**[†], *Nonmembers, and* **Joonwon Lee**[†], *Member*

SUMMARY Page-based DSM systems suffer from false sharing since they use a large page as a coherence unit. The optimal page size is dynamically affected by application characteristics. Therefore, a fixed-size page cannot satisfy various applications even if it is small as a cache line size. In this paper we present a software-only coherence protocol called BCP(Buddy Coherence Protocol) to support multiple page sizes that vary adaptively according to the behavior of each application during run time.

In BCP, the address of a remote access and the address of the most recent local access is compared. If they are to the different halves of a page, BCP considers it as false sharing and demotes the page to two subpages of equal size. If two contiguous pages belong to the same node, BCP promotes two pages to a superpage to reduce the number of the following coherence activities. We also suggest a mechanism to detect data sharing patterns to optimize the protocol. It detects and keeps the sharing pattern for each page by a state transition mechanism. By referring to those patterns, BCP selectively demotes the page and increases the effectiveness of a demotion. Self-invalidation of the migratorily shared page is also employed to reduce the number of invalidations.

Our simulations show that the optimized BCP outperforms almost all the best cases of the write-invalidate protocols using fixed-size pages. BCP improves performance by 42.2% for some applications when compared against the case of the fixed-size page.

key words: page-based DSM, granularity, false sharing, sharing pattern, self invalidation, adaptive coherence protocol

1. Introduction

The distributed shared memory(DSM) across a network of workstations(NoW) gains wide interests due to good scalability and cost/performance empowered by easy programming paradigm. Among various implementation approaches, page-based DSM utilizes an existing paged virtual memory scheme for providing shared memory view. Page-based DSM has been advocated for NoW since it requires neither programmer's burden for modifying existing software nor any hardware support. Despite such advantages, its poor performance due to coherence overhead has hindered it from being widely used. Page-based DSM systems maintain data coherence at a fixed granularity size of the page size to simplify the mechanism. Variables used by different processors can be allocated into a same page since the data sizes accessed by processors are smaller than the page size in many applications. Such a page would be falsely shared among the processors and may incur unnecessary coherence activities. Large page size increases false sharing[7].

Many researchers have tried to reduce false sharing by maintaining coherence at the level of an object based on programmers' annotations. Nevertheless, they cannot seem to gain wide acceptance in real systems because it is difficult for programmers to understand the dynamic sharing behavior of each application and to modify existing application programs. A few recent systems such as Blizzard[16] and Shasta[14] suggested a framework for lightweight access control in a software-only manner, where access control is allowed at a subpage level by inserting code in an application executable that checks the state of data being accessed. They divide up the pages into fixed-size blocks of typically cache line size and successfully support fine-grain sharing with reasonable cost. Their framework is considered as one of the most viable approach that can solve granularity issue.

Reducing the size of a coherence unit, however, will increase the number of remote operations in transferring a large data. Since networks are more efficient for transferring a small number of large data than a large number of small data, smaller grain size will bring more negative effect especially for coarse-grained applications. The optimal page size satisfying various applications is difficult to decide. The size is affected by various factors and more importantly by dynamic factors such as accesses to shared variables by parallel tasks and process migration. On the other hand, the grain size of parallel processing is usually determined by application characteristics and programming style. This observation leads us to propose a DSM design that allows the page size to change adaptively depending on the behavior of parallel processes without causing programmers' burden.

In this paper we present a software-only coherence protocol to support multiple page sizes that vary adaptively according to the behavior of each application during run time. When an application starts, all the pages used are of the same given size. As the computation proceeds, each page size changes according to the sharing pattern of the page. Many of the previous researches including Shasta support multiple granularity at an object level. They, however, lack in dynamic ad-

[†] The authors are with Department of Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Kusung-Dong, Yusung-Gu, Taejon, 305-701, Korea

justment, and more importantly, depend on programmer's annotation.

Our *buddy coherence protocol(BCP)* is built on a directory-based write-invalidate protocol. In BCP, the page size changes as the workload changes like the buddy system[12] which allocates variable sizes of memory with merge/split operations. When a remote request is received, the owner node compares the address of a remote access with the address of the most recent local access. If they are to the same half of a page, the page is considered to be truly shared. Otherwise, BCP considers it as false sharing. Since such false sharing can be eliminated by splitting the page into two, BCP demotes the page to two subpages of equal size. The subpages are exclusively owned by each of local and remote nodes. After the reply for a request is delivered to the requester node, BCP checks if the buddy of the replied page is owned by the node. If two buddies belong to the same node, the number of remote operations may be reduced by transferring both of the pages for a request. Thus, BCP promotes two pages to a superpage. Performance gains come from the reduction of remote operations and the improvement in communication efficiency.

A page can be truly shared even when false sharing is detected. Therefore, the demotion based only on the false sharing detection may increase overhead without any benefit for the following accesses. In order to reduce useless demotions and promotions, we optimized BCP by adding a feature utilizing data sharing patterns. It detects and keeps the sharing pattern for each page by a state transition mechanism. States of a page include read-only, producer-consumer, migratory sharing and the mixture of those patterns. By referring to those patterns, BCP selectively demotes the page of mixed patterns to reduce useless demotions. Self-invalidation of the migratorily shared page is also employed to reduce the number of invalidations [17].

Our simulations show that a small fixed page size degrades the execution time by 42.2% for some applications when compared against the best performance using a fixed size page. And an optimized BCP outperforms almost all the best cases of other protocols using fixed size pages. The sharing pattern is unclear when several different patterns are irregularly mixed in the same page [9]. The self-invalidation feature of the BCP, however, is shown to improve performance upto 19.2%.

BCP and the sharing pattern detection mechanism are simple enough to be easily implemented on existing systems, and do not preclude other DSM techniques such as relaxed memory models. Also the right size of coherence unit is dynamically determined by the system, and thus one of the most difficult decisions in DSM design is resolved. Though our approach is assumed to be implemented in page-based DSM, we believe that it will be effective for the systems in other categories of DSMs.

This paper is organized as follows. BCP design and the optimizations are described in Section 2. Section 3 covers implementation issues. Simulation framework and the results are presented in Section 4. Some related works are reviewed in Section 5. Finally, Section 6 concludes this paper.

2. The Buddy Coherence Protocols

2.1 Basic Design

The workstations interconnected by a conventional network is assumed as the platform for our scheme. Each workstation is called as a node that will also stand for a processor interchangeably. Virtual address space of each node is divided into private and shared regions like other DSM systems such as Shasta[14]. Only the shared region is managed by user-level DSM routines and may be cached by multiple nodes. The DSM routines of each node keep a page table maintained by the dynamic distributed manager algorithm without write-back [13].

When neither a demotion nor a promotion happens, BCP behaves like other coherence protocols such as a write-invalidate protocol or a write-update protocol. Though BCP may be built on any of them, we choose and analyze the write-invalidate protocol as the base protocol of BCP since it is more popular in conventional DSM systems. The base protocol works as follows: if a reading or writing node(requester) does not have the page, it issues a read or write request to the node owning the page(owner) and proceeds the operation after it receives the copy from the owner; a writing node issues an invalidation request to all other nodes sharing the copy(sharers), and then, it owns the page and is free to update the page until another node asks for it. The page in each node is in one of the four states: exclusive, shared-own, shared and invalid.

We call the page of a size defined by the operating system a *base page*. The smaller chunk of a base page and the larger size block merging multiple pages are called a *subpage* and a *superpage* respectively. In BCP, a page is said to be falsely shared between two nodes if the currently or the most recently accessed offset of each node belongs to different half of the page.

The operations of BCP can be described from the views of the owner and the requester. The owner of a page checks false sharing when it receives a read or a write request from a requester. If false sharing is detected, the owner splits the page into two subpages(buddies) of equal size and migrates the requested half to the requester. This operation is called a demotion. When a requester receives the reply, it checks if it already owns the buddy(the other half) of the replied page. If it does, it merges two buddies to a superpage. This is a promotion. The size of a page at an instance would be 2^n or $1/2^n$ times of a base page with an arbitrary integer, n. Each of a demotion and a promotion may change the value of n by only one at a time in order not to cause rapid change in the locality.

The control information on a page such as the page size and the owner should be consistent among all the sharers. Thus, the information changed by a demotion or a promotion should be propagated to all the sharers, and the sharers' copies may be invalidated if the page is being shared by multiple nodes. Such updates or invalidations incur considerable overhead which limits the scalability of BCP as the number of nodes increases. In order to guarantee scalability, BCP allows a demotion or a promotion only when such overhead is not incurred. That is, a demotion for a read request and a promotion happens only when a page is exclusively owned by the owner. A demotion for a write request happens even when a page is shared by the other nodes since all of the sharers would be invalidated not by a demotion but by a write itself. Consequently, there is no overhead for distributing information about demotions and promotions, and the same degree of scalability is guaranteed as the base protocol. We call this protocol a basic BCP.

2.2 Protocol Optimizations

A demotion or a promotion cannot guarantee performance gain if the change in page size cannot satisfy the sharing behavior of the following accesses. For example, a demotion of a read-only page can provide little benefit since the read-only sharing does not need the migration of ownership. Even when a page is updated by multiple nodes, if the whole page is accessed only by one node at a time, a demotion may increase the number of remote requests. If a promotion of an actively shared page is allowed, a number of demotions and promotions may happen repeatedly and cause performance loss rather than gain. In order to cope with these weaknesses, we optimize the basic BCP by utilizing data sharing patterns.

2.2.1 Classification of Sharing Patterns

Previous researchers[2], [3] have identified the following categories of shared objects: read-only, producer-consumer, migratory and general read-write. We add two more categories, *private* and *mixed* resulting total six types of sharing patterns.

A *read-only* page is the page that has not experienced any write since it was created. While a page is in the read-only state, the page is better to be kept in a larger size. A *producer-consumer* page is written by one node(producer) and read by a set of other nodes(consumers). The accesses of the producer and the consumers are usually phased in. This type of sharing would favor a write-update protocol rather than a write-invalidate protocol since the result of a write by the producer is to be read again by the consumers.

A *migratory* page is the one that is accessed by multiple nodes, but only by one node at a time. The accesses of each node are usually introduced by a read and include one or more writes. Self invalidation, in which the owner does not copy but migrates a page even for a read request, can reduce the number of invalidations in handling this type of pages[17].

A *private* page is accessed only by a single node. Though a page belongs to the shared region, sharing may actually happen neither for the whole page nor during the whole execution time. If we can differentiate such a non-shared portion from a page, the number of coherence activities will be reduced. When a page shows multiple sharing characteristics, the state of the page is defined as *mixed.* It is expected to be a potential source of false sharing. A *general read-write* page means all pages that do not fall into any of the above categories. Sharing of a general read-write page will happen irregularly in fine granularity. Thus, a smaller page will show better performance for it.

Synchronization is usually provided not through shared memory but in library-like implementation since synchronization variables are accessed in a totally different way than normal data. Thus, our classification does not include a category specialized for them.

2.2.2 Detection of Sharing Patterns

Though an object may have its inherent sharing pattern, the sharing pattern at a page-level may appear differently due to false sharing or fragmentation[9]. The set of sharers and the portion accessed by each node may be changed during run time. Such changes will cause transitions in types of sharing patterns continuously. One way to detect sharing patterns is to keep a historic access stream for each page and analyze it when a decision is needed. However, such a method incurs significant overhead since the decision should be made for every shared access.

We suggest a mechanism to detect the sharing pattern of each page at run time as shown in Figure 1. Each of shaded areas means the pattern that a page is showing at an instance. A pattern is composed of one or more states which are denoted by circles. A page starts either as read-only or as private according to the first access to it. If the first access is a read, a page becomes read-only(RO) and remains in RO until a write happens. If a write is done by the owner, the page is considered as producer-consumer. Otherwise, the state of the page becomes general read-write. If the first access is a write, a state becomes private(PR). The state becomes general read-write by a remote access.

To identify the producer-consumer pattern, three more internal states are introduced: single producer(1P), single producer single consumer(1P1C) and

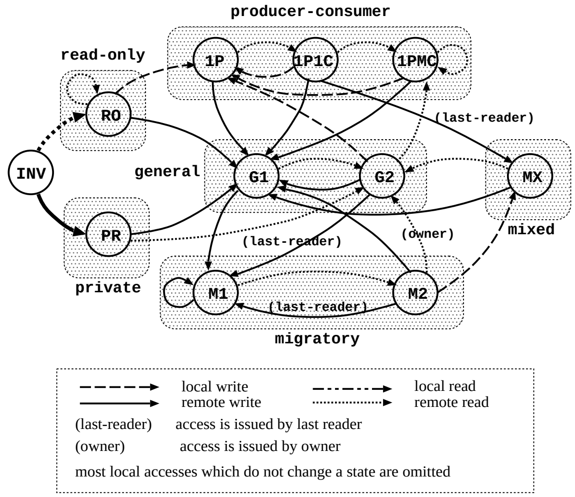


Fig. 1 Detection Mechanism of Sharing Patterns

single producer multiple consumers(1PMC). 1P means there exists no other copy but the page owned by the producer. 1PxC means x shared copies exist besides the owner's copy. 1P state is changed to 1PxC by x times of remote reads. A local write changes the state from 1PxC to 1P since the page is always owned by the producer node. If a remote write happens, this producer-consumer pattern is violated and the page becomes general read-write. When a page is in 1P1C, and if the consumer issues a write, the pattern of the page falls into the mixed category since it is a typical pattern of migratory sharing.

For a migratory pattern, the state for an exclusive copy(M1) and the state for two copies(M2) are employed. If a remote write happens for any exclusively-owned page, the page is considered as migratory and the state becomes M1. A remote read is allowed only once and it changes the state to M2. When a page is in M2, and if the sharer issues a write, the page remains as migratory and the state becomes M1. If another remote read happens, the page is considered as mixed since the pattern is the same as 1PMC. Otherwise, the page becomes general read-write.

The state for the general read-write pattern is split into G1 for an exclusive copy and G2 for two copies in order to detect a producer-consumer pattern and a migratory pattern. When a page is in G2, the page becomes producer-consumer either by a local write or by a remote read. A remote write in G1 and the sharer's write in G2 change the pattern of the page to migratory. Otherwise, the page remains as general read-write.

Finally, a mixed pattern is denoted by MX. The page remains in MX while only local accesses happen. In order not to make most pages fall into the mixed pattern, the pattern is changed into general read-write by the following remote access.

As shown in Figure 1, the detection mechanism works like a consistency protocol. It, however, just provides a hint for the current sharing patterns at a moment. It may be combined with the state transition of a coherence protocol.

2.2.3 Optimizations

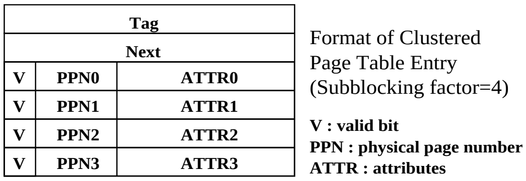
Our protocol optimizations include the selective demotion/promotion based on sharing patterns and the self-invalidation of migratory pages.

If any sharing pattern of a page is known, we can handle it in a specialized manner for utilizing its characteristics. Thus, the selective demotion/promotion is based on the basic heuristic that a falsely shared page should be demoted only when it does not show any typical pattern. In this optimization, a demotion is allowed only when the pattern of a page is either mixed or general read-write. The demotion of a mixed page is expected to derive any single pattern, while the demotion of a general read-write page is done for supporting the irregular and fine-grained sharing in such a page. Additionally, the protocol does not allow a demotion for a remote write request if the sharer exists, that is, the page is in G2. This constraint is based on the rationale that the demotion in G2 may not be more useful than keeping the detection process since G2 can be considered as a process of detection rather than a resulting state.

If a promotion is allowed when a page shows different pattern from its buddy or the pattern is either mixed or general read-write, demotions and promotions may be repeated in turns. Therefore, a promotion is allowed when two buddies are of the same pattern that is neither mixed nor general read-write. We call this version of optimized protocol BCP-sdp(Selective Demotion/Promotion).

In the protocol optimized by the self-invalidation, the owner migrates a page for a read request if the page is detected as migratory. Therefore, the arrow from M1 to M2 and the state of M2 in Figure 1 can be omitted. We add this feature to BCP-sdp and call the new protocol BCP-si(Self-Invalidation). Though the protocol can be further optimized by adding another states for checking if the following accesses keep the pattern after a self-invalidation, we left it as a future work.

BCP resets the sharing pattern of a subpage that is newly created by a demotion as follows. The requester's subpage demoted by a read request becomes read-only, and the other buddy subpage becomes private. When a demotion happens, BCP has no information on the sharing pattern of each subpage. Thus, resetting the pattern and letting the page undergo the whole detection process may be better than assuming any uncertain pattern.



Tag		
Next		
V	PPN0	ATTR0
V	PPN1	ATTR1
V	PPN2	ATTR2
V	PPN3	ATTR3

Format of Clustered Page Table Entry (Subblocking factor=4)

V : valid bit
PPN : physical page number
ATTR : attributes

Fig. 2 Clustered Page Table Entry

3. Implementation Issues

As mentioned earlier, BCPs are assumed to be built on a software DSM system with the feature of inline state checks for access control such as Blizzard[16] and Shasta[15]. We do not attempt to cover the implementation issues in the full system. We instead focus on a several engineering issues that may cause modifications on existing systems for implementing BCPs.

Access control can be implemented by adding a shared access check in the executables. Such inline state checks instead of depending on virtual memory hardware eliminate the need for expensive OS interactions for manipulating page protection. An executable editing program or a compiler inserts a code before LOADs and STOREs. The code checks if the target address is in the shared memory range which is above a certain address in each processor's address space and invokes protocol handlers. The user-level handlers execute coherence actions with their own page table and resume with a physical address for the local processor. As a result, any granularity can be supported without depending on OS and MMU. The details of the implementation and the optimization can be found in [16] and [15].

The structure and management of the page table may have potential effects on the performance since BCPs create or destroy pages frequently. If page table entries are dynamically inserted or deleted, considerable overhead is expected for restructuring the page table. If the size of the page table is statically set to contain the maximum number of entries, the space overhead will be significant. Though technology trends mitigate space overhead, a large number of page table entries will cause delay in looking up the table. Such delay is also incurred in other DSM systems that support fine-granularity through a small fixed-size page. When we take into account that most operating systems implement page tables in level-three to cover 32-bit address space, more levels of indexing would be practically required for fine-grained pages.

We suggest clustered page tables as in [18] for maintaining fine-grained pages in BCPs. The clustered page table is an extension of a hashed page table that stores mapping information for several consecutive pages with a single tag as shown in figure 2. The table lookup time can be reduced since the table walk may not be iterated. BCP can naturally represent superpages and subpages in the clustered page table. To represent a page of multiple size of a minimum subpage, only the first minimum subpage that belongs to the page keeps valid information. The number of pages sharing a tag is called subblocking factor. If a wide range of page size is required to be supported, the subblocking factor must also be increased. According to the requirement, BCP page table can be designed in either a single level or two-levels. A two-leveled structure using subblocking factor of 16 can cover the page sizes from 32 bytes to 8 Kbytes.

The format of a page table entry should also be modified to include page size information and the most recently accessed offset. Additionally, the type of sharing pattern should be kept for BCP-sdp and BCP-si. The offset and the type of sharing pattern require new fields of 12 bits and 4 bits respectively for every page table entry. The number of bits for the offset assumes 4 Kbyte base page. Even though it is required to know which node is the last-reader of a page, it does not need extra information since the copy sets of most DSM systems keep the sequence information.

However, the page size information should be maintained with the tag field. Otherwise, BCPs repeat references to each entry of intermediate-sized pages to reach a subpage. Such a repetition can be discarded with a stream of bits representing each minimum subpage. If we set only the bits for the first subpage of each page, the stream is divided into several substreams. When an address falls into a substream and if the number of zeros in the substream is N, current size of the page including the address will be $2^{\log(N+1)}$ times of the minimum size.

Though BCPs can start with a base page of any size, the size would affect the resulting performance since the efficiency in converging to a proper size can changes. If DSM designer needs a large base page for any reason, we can tune up BCPs performance by loosening the demotion condition and by tightening the promotion condition, and vice versa. The conditions described in the previous section are not tuned for any page size or any application. The optimal base page sizes under various conditions are suggested in the next section.

4. Simulations

4.1 Framework

For the performance study, we developed a program-driven simulator on MINT[20]. Six parallel applications from SPLASH/SPLASH-2[21] suites are used as the workload for the simulator. We tried to include the applications of various characteristics. Their characteristics are already well-studied in many previous works[9] as summarized in Table 1.

Four coherence protocols, base write-invalidation

Table 1 Workload Specification

Application	Problem Size	Sharing	Granularity
FFT	64K double	1P-1C	coarse
LU	512 by 512 matrix, 16 by 16 blocks	1P-MC	coarse
MP3D	10000 molecules, 10 iterations	migratory	fine-medium (varying)
Ocean	130 by 130 ocean	1P-1C	fine read coarse write
Pthor	1000 ticks 100genT, 10cyc	MP-1C	fine
Water	512 molecules	1P-MC	coarse

protocol, basic BCP, BCP-sdp and BCP-si, are simulated with a base page size varying from 64 bytes to 4 Kbytes. We use the total execution time in processor cycle as the metric. The execution time is the elapsed time from program start to program end.

A NoW of 32 nodes is assumed as an architectural environment. Each node uses 200 MHz CPU and main memory sufficient to hold the simulated problems without paging. Two-leveled clustered page table is assumed, and consequently a page size can vary from 32 bytes to 8 Kbytes regardless the base page size. The nodes are interconnected by an ATM network with a point-to-point bandwidth of 155 MBit/second. The simulations conservatively account for the minimum communication delay (2.74 μsec/53 bytes on the wire and 3000 cycles for socket overhead) and the worst case of the software overhead of handling a page table in the user-level (70 cycles and additional 10 cycles for keeping sharing patterns) and demoting / promoting pages (150 cycles). The instruction counts of our experimental codes are used as the software overhead. Instruction references are assumed to take one cycle.

4.2 Results and Analysis

4.2.1 Overall Performance

Figure 3 shows the execution times of the applications using the base page of seven different sizes. In the base protocol, the execution time sensitively changes for every application according to page sizes. When a page is smaller than the best size, the extra delay is caused by the increased number of remote operations due to both of true and false sharing. When a larger page is given, the delay is increased not only by the sharing that potentially happens falsely but by the transmission delay due to its large messages. FFT and LU of Figure 3-(a) and (b) inherently show only a small number of coherence operations and no false sharing. Only FFT shows a few fragmentations. In this case, we can expect only a reduction in communication overhead by a demotion. Basic BCP fails in the speedup for them when small base pages are given. Its loose conditions for a demotion and a promotion result in a number of ineffective demotions and promotions and their overhead exceeds performance gains. However, BCP-sdp and BCP-si reduce the number of operations and perform well for all page sizes. In LU, they achieve the speedup upto the level of the best case even for a 4 Kbyte page.

The granularity of MP3D varies during run time since it is proportionally determined by the amount of work a node does between barrier synchronizations. An adaptive approach in page size, therefore, is expected to satisfy the inherent dynamic feature. Figure 3-(c) supports such a hypothesis. BCP-si is superior to the base protocol for all sizes of base pages while basic BCP is not for a few small pages. BCPs also show efficient convergence to smaller pages when large base pages are given.

In Ocean, the dominant sharing happens in borders to the neighbor for each partition. While write accesses are local and coarse-grained, remote read accesses are very fine-grained especially along column-oriented borders. Thus, there is little false sharing but significant fragmentation when larger pages are used. We expect BCPs can solve the inherent multiple granularity. Figure 3-(d) shows the expected results. Basic BCP cannot provide speedups for most page sizes. However, the other protocols reduce the execution time through the utilization of sharing patterns and the self-invalidation.

Pthor has an irregular computation containing pointers to linked lists, which has made it difficult to utilize regular patterns. However, we expect the adaptive feature of BCPs can solve this problem of irregularity in Pthor. Figure 3-(e) shows that BCPs perform well in a very regular fashion as expected. Each optimization achieves stepwise speedups.

Water-spatial shows very similar behavior to LU without any false sharing or fragmentation. Another characteristic is that each water molecule is allocated in a separate page, and thus the 1P-MC sharing pattern is expected. In Figure 3-(f), BCPs perform well by reducing communication overhead. However, the self-invalidation of BCP-si provides no performance gain since its producer-consumer pattern is kept even for larger pages and the number of migratory patterns detected in execution is not large enough to affect the performance.

We conclude that BCPs outperform the base protocol, especially for the applications containing irregular sharing behavior.

4.2.2 Practical Speedup

Table 2 summarizes two cases of comparisons between the base protocol and BCPs. When we compare the best cases of the base protocol and BCP-si without considering any given size of a base page, BCP-si is superior to the base protocol for all of the applications. The best case comparison, however, has no meaning since the base page must be fixed at the time when the

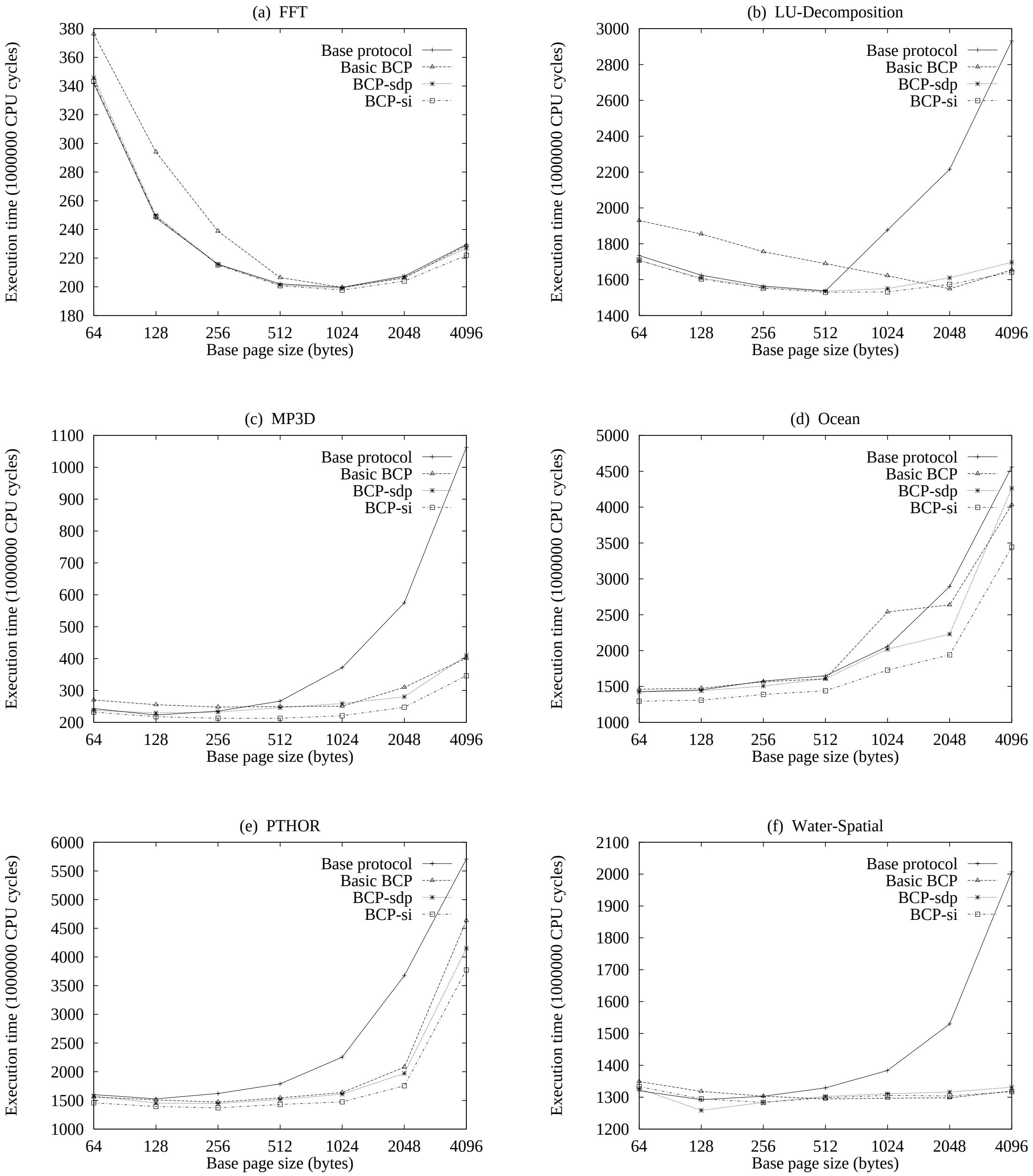


Fig. 3 Changes in Execution Time According to Base Page Size

Table 2 Speedup of BCPs Against Base Protocol

Size	FFT	LU	MP3D	Ocean	Pthor	Water
Base(best) page size	199.67 1K	1537.40 512	222.81 128	1427.98 64	1527.39 128	1292.23 128
BCP-si(best) page size	197.72 1K	1529.12 512	212.68 512	1298.74 64	1370.89 256	1283.65 256
Base(64)	341.87	1734.21	242.95	1427.98	1600.38	1320.23
BCP-sdp(512) (speedup,%)	201.28 41.12	1534.81 11.50	245.83 -1.18	16.9.48 -12.71	1515.44 5.31	1302.37 1.35
BCP-si(512) (speedup,%)	200.75 41.28	1529.12 11.83	212.68 12.46	1439.52 -0.81	1429.10 10.70	1298.73 1.63

(10^6CPU Cycles, bytes)

DSM system starts up. The comparison should be done in the designer's view in determining the size of a base page. BCP-sdp and BCP-si perform well for most applications when the base page size is in the range from 256 bytes to 1 Kbyte. We assume 512 bytes as the base page size for BCP-sdp and BCP-si, and compare them with the base protocol using a 64 byte fixed-size page.

Lower rows in the table shows the execution times of the base protocol, BCP-sdp and BCP-si when the base protocol uses a 64 byte base page and BCPs use a 512 byte base page. BCP-sdp fails in speedup for MP3D and Ocean since the optimizations of BCP do not allow enough number of demotions to converge to the optimal size for these fine-grained applications. BCP-si

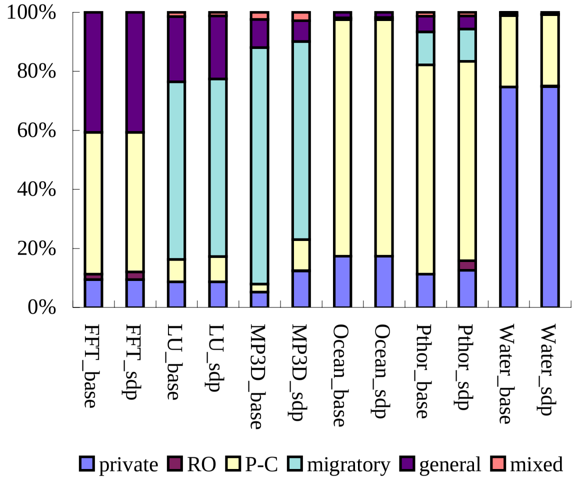


Fig. 4 Distribution of Sharing Patterns

Table 3 Effect of Self-Invalidation in Speedup

Case	FFT	LU	MP3D	Ocean	Pthor	Water
Average (%)	0.82	1.09	10.19	11.83	7.28	0.21
Best Case(%)	2.22	3.22	15.57	19.23	10.95	0.94
Worst Case(%)	0.14	-0.11	2.26	7.70	4.11	-0.71

provides better performance than the base protocol for most applications. Though BCP-si cannot reduce the execution time for Ocean, the performance is within 1% of the base protocol, where the 64 bytes page provides the best performance for Ocean. We did not try to tune up the BCPs for any application or any size of a base page in order to obtain good coverage. We believe BCPs can be further optimized and the performance can be improved.

4.2.3 Utilization of Sharing Patterns

As discussed in the previous sections, the protocol optimizations utilizing sharing patterns reduce the execution time for most application. In this section, we focus on the effect of introducing sharing patterns and show the feasibility of more extensive utilization of them.

BCPs may change the distribution of sharing patterns since demotions and promotions create or destroy a page. Figure 4 illustrates the distributions of sharing patterns detected by our detection mechanism for the base protocol and BCP-sdp when the base page size is 512 byte. In the comparison of a pair of bars for each application, BCP-sdp increases the portions of private, read-only and producer-consumer patterns while decreasing general and mixed patterns. For examples, the portions of private patterns and producer-consumer patterns in MP3D increase upto 235% and 392% respectively, and the portion of read-only patterns in Pthor increases upto 214123%. These increased portions are another source of performance gains since they invoke no or fewer number of remote operations.

Table 3 shows the effect of the self-invalidation in BCP-si when compared with the performance of BCP-sdp. It achieves speedups against BCP-sdp for most applications upto 19.23% while it shows negative effects for LU and Water. The negative effects are caused by the penalty for wrong detections of migratory patterns. Such cases may happen in every application. In other applications, the loss can be hidden by performance gains. Unfortunately, performance gain is not enough to cover the loss in the worst cases of LU and Water. However, BCP-si keeps the loss within 1% of the performance of BCP-sdp.

5. Related Work

In the area of cache coherence protocols, several approaches to solve the problem of granularity can be found: the partial invalidation protocol[4], the data placement optimization[19] and the adjustable cache[6]. The adjustable cache adopted the buddy system in a similar manner as our study. However, all of these approaches are inapplicable and/or ineffective to page-based DSM since they are based on hardware implementation.

In order to support a fine-grain or variable size coherence unit in a software-only approach, the object-based DSM and the region-based DSM [1],[10] have been proposed. However, they sacrifice the ease of programming since a lot of responsibility is given to a programmer. Moreover, they lack run time adaptivity due to the dependency on compilers.

The write-shared protocol[11] allows multiple threads to write on a same page without intervening any synchronization because the programmer is responsible for its arbitration. The word-by-word comparison(diff) of a page and its twin guarantees only the dirty portions are propagated among nodes. Its handicaps lie in the programmers' burden and the overhead in handling the diff. The writer-owns protocol[8] detects false sharing in run time and migrates falsely-shared data at the release point. As a result, the information propagated among nodes is not data but the changes in mapping information. Its approach and effect are similar to those of the write shared protocol.

Blizzard-S[16] and Shasta[14] present the good basis for implementing fine grain access control in page-based DSMs. Still, their major interests are in reducing the checking overhead without causing a page fault, so that they do not include the adaptive feature presented in this paper. While Shasta allows object-level coherency, it requires programmers' burden in using special memory allocation primitives.

Till now, most researches on the utilization of sharing patterns have been concentrated only in the level of a small coherence unit such as a cache line since the sharing patterns in a larger unit are distorted by false sharing and fragmentation[9]. However, our results show that the sharing patterns can be utilized even in a large page. A pattern may change frequently and may show different shape from its inherent behavior. Our detection mechanism traces such changes and determines the most probable type of sharing pattern that is the most probable for a given page.

6. Conclusion

We have presented a family of buddy coherence protocols for page-based DSM systems, where multiple page sizes can be adaptively supported during run time according to the behavior of applications. We suggest a sharing pattern detection mechanism that can work well even when a large coherence unit is used. Simulation results show that our protocols are outperform a write-invalidate protocol for most applications even when the write-invalidate protocol uses a small-sized coherence unit. The execution time can be reduced by upto 41% when compared with the case of base protocol using a 64 byte base page. Selective demotion/promotion and self-invalidation for migratorily-shared pages successfully cut down overhead resulting in better performance.

BCPs and the sharing pattern detection mechanism can be further tuned up. We expect BCPs can achieve better performance if the conditions for a demotion and a promotion are elaborated. The detection mechanism can also be refined. We expect it can be extensively utilized in not only BCPs but any other protocol. The effect on memory models and other existing techniques should also be studied. We are interested especially in how BCPs affect the write-shared protocol. BCPs are expected to partially provide a similar effect to the write-shared protocol without any annotation by a programmer.

In addition to performance gains, our idea and approach are simple enough to be implemented easily in the existing environments. Also our scheme is tolerable to a mistake in selecting the right size for a DSM system.

References

[1] B. N. Bershad, M. J. Zekauskas and W. A. Sawdon, "Midway : distributed shared memory system," *Proceedings of the IEEE CompCon*, pp.528-537, February 1993.

[2] J. K. Bennett, J. B. Carter and W. Zwaenepoel, "Adaptive software cache management for distributed shared memory architectures," *Proceedings of the 17th ISCA*, pp.125-135, May 1990.

[3] J. B. Carter, J. K. Bennett and W. Zwaenepoel, "Techniques for reducing consistent-related communication in distributed shared memory system," *ACM Transactions on Computer Systems Vol.13 No.3*, pp.205-243, August 1995.

[4] Y. S. Chen and M. Dubois, "Cache Protocols with partial block invalidations," *Proceedings of the 7th International Parallel Processing Symposium*, pp.16-23, April 1993.

[5] A. L. Cox and R. J. Fowler, "Adaptive cache coherency for detecting migratory shared data," *Proceedings of the 20th ISCA*, pp.98-108, May 1993.

[6] C. Dubnicki and T. J. LeBlanc, "Adjustable block size coherent caches," *Proceedings of the 19th ISCA*, pp.170-180, May 1992.

[7] S. J. Eggers and R. H. Katz, "The effect of sharing on the caches and bus performance of parallel programs," *Proceedings of ASPLOS III*, pp.257-270, April 1989.

[8] V.W. Freeh, D. K. Lowenthal and G. R. Andrews, "Distributed filaments: efficient fine-grain parallelism on a cluster of workstations," *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pp.201-212, November 1994.

[9] L. Iftode, J. P. Singh and K. Li, "Understanding application performance on shared virtual memory systems," *Proceedings of the 23rd ISCA*, pp.122-133, May 1996.

[10] K. L. Johnson, M . F. Kaashoek and D. A. Wallach, "CLR : high-performance all-software distributed shared memory," *Proceedings of the 15th ACM Symposium on Operating System Principles*, pp.213-228, December 1995.

[11] P. Keleher, A. L. Cox, S. Dwarkadas and W. Zwaenepoel, "TreadMarks : distributed shared memory on standard workstations and operating systems," *Proceedings of Winter USENIX Conference*, pp.115-132, January 1994.

[12] D. E. Knuth, *The Art of Computer Programming Vol.1 : Fundamental Algorithms 2nd Edition*, pp.435-454, Addison-Wesley, 1969.

[13] K. Li, "Shared virtual memory on loosely coupled multiprocessors," *PhD thesis, Yale University*, September 1986.

[14] D. J. Scales, K. Gharachorloo and C. A. Thekkath, "Shasta : a low overhead, software-only approach for supporting fine-grain shared memory," *Proceedings of ASPLOS VII*, pp.174-185, October 1996.

[15] D. J. Scales and K. Gharachorloo, "Toward Transparent and Efficient Software Distributed Shared Memory," *Proceedings of the 16th ACM Symposium on Operating System Principles*, Oct ober 1997.

[16] I. Schoinas, B. Falsafi, A. R. Lebeck, S. K. Reinhardt, J. R. Larus and D. A. Wood, "Fine-grain access control for distributed shared memory," *Proceedings of ASPLOS VI*, pp.297-306, October 1994.

[17] P. Stenstrom, M. Brorsson and L. Sandberg, "An adaptive cache coherence protocol optimized for migratory sharing," *Proceedings of the 20th ISCA*, pp.109-118, May 1993.

[18] M. Talluri, M. D. Hill and Y. A. Khalidi, "A new page table for 64-bit address spaces," *Proceedings of the 15th ACM SOSP*, pp.184-200, December 1995.

[19] J. Torrellas, M. S. Lam and J. L. Hennessy, "Shared data placement optimizations to reduce multiprocessor cache miss rates," *Proceedings of the 3rd International Conference on Parallel Processing*, pp.266-270, 1990.

[20] J. E. Veenstra and R. J. Fowler, "MINT Tutorial and User Manual," *Technical Report 452 (Revised Ed.), University of Rochester*, June 1994.

[21] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, "The SPLASH-2 programs : characterization and methodological considerations," *Proceedings of the 22nd ISCA*, pp.24-36, May 1995.

Sangbum Lee received the B.S. and M.S. degrees in computer science from

Seoul National University, Seoul, Korea, in 1984 and 1986, respectively. Since 1988 he has been a technical staff in the Korea Securities Computer Corporation(KOSCOM). He is currently working towards the Ph.D. degree in computer science at the Korea Advanced Institute of Science and Technology(KAIST) since 1994 in the educational program of KOSCOM. His research interests include operating systems and computer architectures.

Inbum Jung received the B.S. degrees in electronics engineering from Korea University, in 1985, and the M.S. degree in information & communication engineering from KAIST, in 1994. He is currently working towards the Ph.D degree in computer science from KAIST. From 1984 to 1994, he was with the Computer System Division of Samsung Electronics Co., Ltd., Korea. His research interests include operating systems, computer architectures, parallel processing and multimedia systems.

Joonwon Lee received the B.S. degree from Seoul National University, in 1983 and the M.S. and Ph.D. degrees from the College of Computing, Georgia Institute of Technology, in 1990 and 1991, respectively. From 1983 to 1986, he was with Yugong Ltd., and from 1991 to 1992, he was with IBM research centers where he was involved in developing a scalable shared memory multiprocessor. He is currently an associated professor at KAIST. He has initiated several research projects in the area of operating systems and he was a recipient of Windows NT source code. His research interests include operating systems, computer architectures and parallel processing.