



Optimizing Home-Based Software DSM Protocols*

WEIWU HU, WEISONG SHI and ZHIMIN TANG

Institute of Computing Technology, Chinese Academy of Sciences, PR China

Abstract. Software DSMs can be categorized into homeless and home-based systems both have strengths and weaknesses when compared to each other. This paper introduces optimization methods to exploit advantages and offset disadvantages of the home-based protocol in the home-based software DSM JIAJIA. The first optimization reduces the overhead of writes to home pages through a lazy home page write detection scheme. The normal write detection scheme write-protects shared pages at the beginning of a synchronization interval, while the lazy home page write detection delays home page write-protecting until the page is first fetched in the interval so that home pages that are not cached by remote processors do not need to be write-protected. The second optimization avoids fetching the whole page on a page fault through dividing a page into blocks and fetching only those blocks that are dirty with respect to the faulting processor. A write vector table is maintained for each shared page in its home to record for each processor which block(s) has been modified since the processor fetched the page last time. The third optimization adaptively migrates home of a page to the processor most frequently writes to the page to reduce twin and diff overhead. Migration information is piggybacked on barrier messages and no additional communication is required for the migration. Performance evaluation with some well-accepted benchmarks and real applications shows that the above optimization methods can reduce page faults, message amounts, and diffs dramatically and consequently improve performance significantly.

Keywords: home-based software DSM, scope consistency, lazy write detection, reducing message overhead, home migration, performance evaluation

1. Introduction

Software distributed shared memory (DSM) systems can be categorized into homeless and home-based systems both have strengths and weaknesses when compared to each other. Home-based systems differ from homeless ones in that each shared page has a designated home to which all writes are propagated and from which all copies are derived. Home-based systems benefit from the convenience introduced by the home. The home node of a shared page can modify the page directly without trapping diffs for the modification, and a remote node can get the page directly from the home node on a page fault. By contrast, homeless protocols must trap all writes to shared pages and must collect diffs from all concurrent writers of a shared page on a page fault. However, disadvantages of home-based protocols are also obvious. Modifications of shared pages have to be eagerly sent back to their home at the end of an interval. Besides, on a page fault, homeless protocols fetch diffs only while home-based protocols fetch the whole page.

This paper introduces three optimizations to exploit advantages and offset disadvantages of the home-based protocol in the home-based software DSM JIAJIA [8].

The first optimization reduces the overhead of writes to home pages through a lazy home page write detection scheme. Though home-based protocol does not require twin and diff to record writes to home pages, it still needs to detect whether a home page is modified to keep the page co-

herent. The normal method of home page write detection write-protects home pages at the beginning of a synchronization interval so that writes to home pages can be detected through page faults. The lazy home page write detection delays home page write-protecting until the page is first fetched in the interval so that home pages that are not cached by remote processors do not need to be write-protected.

The second optimization is motivated by the idea of fetching diffs only on a page fault in homeless protocols. It avoids fetching the whole page on a page fault through dividing a page into blocks and fetching only those blocks that are dirty with respect to the faulting processor. A write vector table is maintained for each shared page in its home to record for each processor which block(s) has been modified since the processor fetched the page last time.

The third optimization adaptively migrates home of a page to the processor most frequently writes to the page. This optimization helps reduce diff overhead because writes to home pages do not produce twin and diff in home-based protocol. In the home migration scheme, pages that are written by only one processor between two barriers are recognized by the barrier manager and their homes are migrated to the single writing processor. Migration information is piggybacked on barrier messages and no additional communication is required for the migration.

The effect of these three optimization methods is evaluated on a cluster of eight 300 MHz Power PC nodes with ten benchmarks, include Water, Barnes, and LU from SPLASH2 [23], MG and 3DFFT from NAS Parallel Benchmarks [2], SOR, TSP, and ILINK from TreadMarks benchmarks [18], and two real applications EM3D for magnetic

* The work of this paper is supported by the National Climbing Program of China and the National Natural Science Foundation of China (Grant No. 60073018).

field computation and IAP18 for 18-layer climate simulation. Evaluation results show that these optimization methods achieve a speedup of 7% in Water, 23.4% in LU, 52.7% in SOR, 17% in TSP, 15.9% in ILINK, 24.1% in MG, 21.5% in EM3D, and 12.5% in IAP18. Only Barnes and FFT do not benefit significantly from the three optimization methods.

The rest of this paper is organized as follows. The following section 2 briefly introduces the JIAJIA software DSM system. Section 3 illustrates the three optimizations implemented in JIAJIA. Section 4 presents experiment results and analysis. Section 5 cites some related work. The conclusion of this paper is drawn in section 6.

2. The JIAJIA software DSM system

In JIAJIA, each shared page has a designated home node and homes of shared pages are distributed across all nodes. References to home pages hit locally, references to non-home pages cause these pages to be fetched from their home and cached locally. A cached page may be in one of three states: Invalid (INV), Read-Only (RO), and Read-Write (RW). When the number of locally cached pages is larger than the maximum number allowed, some aged cache pages are replaced to their home to make room for new pages. This allows JIAJIA to support shared memory that is larger than physical memory of one machine.

JIAJIA implements the scope memory consistency [10] model. Multiple writer technique is employed to reduce false sharing. In JIAJIA, the coherence of cache pages is maintained through write-notices kept on the lock.

On a release, the releaser performs a comparison of all cache pages written in the current critical section with their twins to produce diffs. These diffs are then sent to homes of associated pages. After all diffs have been applied to home pages, a message is sent to the associated lock manager to release the lock. Write-notices of the critical section are piggybacked on the release message to notify modified pages in the critical section. On an acquire, the acquiring processor sends a lock acquiring request to the lock manager. The requesting processor is then stalled until it is granted the lock. When granting the lock, the lock manager piggybacks write-notices associated with this lock on the granting message. After the acquiring processor receives this granting message, it invalidates all cache pages that are notified as obsolete by the associated write-notices. A barrier can be viewed as a combination of an unlock and a lock. Arriving at a barrier ends an old “critical section”, while leaving a barrier begins a new one. On a barrier, all write-notices of all locks are cleared.

On a read miss, the fault page is fetched from the home and mapped in RO state in the local memory. On a write miss, if the written page is not present or is in INV state in the local memory, it is fetched from the home and mapped locally in RW state. If the written page is in RO state in the local memory, the state is turned into RW. A write-notice is recorded about this page and a twin of this page is created.

On replacement of a cache page, the replaced page is written back to its home if it is in RW state in the cache.

The above description of the lock-based cache coherence protocol does not include any optimization. We made the following preliminary optimizations to the above basic protocol [8].

2.1. Single-writer detection

In the basic protocol, a cached page is invalidated on an acquire if there is a write-notice in the acquired lock indicating that this page has been modified. However, if the modification is made by the acquiring processor itself, and the page has not been modified by any other processors, then the invalidation is unnecessary since the modification has already been visible to the acquiring processor. With this optimization, a processor can retain the access right to pages modified by itself on passing an acquire or barrier. Besides, if a page is modified only by its home node, and there is no cached copy of the page, then it is unnecessary to send the associated write-notice to the lock manager on a release or barrier.

2.2. Incarnation number technique

The purpose of the incarnation number technique [3,10] is to eliminate unnecessary invalidation on locks. With this optimization, each lock is associated with an incarnation number which is incremented when the lock is transferred. A processor records the current incarnation number of a lock on an acquire of the lock. When the processor acquires the lock again, it tells the lock manager its currently incarnation number of the lock. With this knowledge, the lock manager knows which write-notices have been sent to the acquiring processor on previous lock grants and excludes them from write-notices sent back to the acquiring processor this time.

3. Protocol optimization

3.1. Lazy home page write detection

Though home-based protocol does not require twin and diff to record writes to home pages, it still needs to detect whether a home page is modified to keep the page coherent. The normal method of home page write detection write-protects home pages at the beginning of a synchronization interval so that writes to home pages can be detected through page faults. Write protecting and trapping entail additional runtime overhead on the protocol. Our previous experiment with JIAJIA shows that, for regular applications with large shared data set and good data distribution, most write detection to home pages is superfluous because many pages are accessed only by the home processor. To remove unnecessary write detection overheads, JIAJIA borrows the idea of lazy diff creation in TreadMarks and implements a lazy write detection scheme that can keep a home page writable across intervals.

Since the purpose of detecting writes to a home page is to produce write notice about this page so that cache copies of this page can be invalidated according to the coherence protocol, it is unnecessary to detect writes to home pages which are not cached. To remove unnecessary write detection, a *read notice* is associated with each home page to record whether the page is exclusive in the home or has remote copies. An exclusive home page is not write-protected at the beginning of an interval. Instead, the write protection is delayed until the exclusive home page turns into non-exclusive on receiving of a remote access request.

Unfortunately, the home processor of a page cannot know exactly whether the page is exclusive or not because it is not informed when a read-only copy of the page is replaced or invalidated by a remote processor. However, it does know that the page is exclusive in some cases. A useful case is that a home page will always be exclusive on leaving a barrier if it is modified by its home processor just before the barrier, because according to the protocol, all remote copies of a page are invalidated on a barrier if the page is modified by its home host in the last barrier interval.

In summary, the lazy home page write detection method makes following modifications to the original write detection method. (1) At the beginning of a synchronization interval, non-exclusive home pages are write protected while exclusive home pages retain write permission. (2) On receiving a remote get page request, the home page state is checked. If the read notice field of the page is “0” (i.e., the page is exclusive in the home), then the home page is write-protected and the read-notice field of the page is set to “1” before the page is returned to the requesting processor. Otherwise, the page is returned to the requesting processor as normal. (3) On a barrier, the read-notice field of a home page is set to “0” if the page is modified by the home host in the last barrier interval.

3.2. Reducing message overhead

In JIAJIA and in other home-based software DSMs, the whole page is fetched from its home on a page fault and fetching remote pages constitutes the major message overhead. Though it is believed that the cost of sending additional data on a message is not significant and typically much less than sending additional messages, our experiments with Ethernet show that the message transmission time is not sensitive to the message size only when the message is small (less than 1 KB). For larger messages such as a page, the message transmission time scales linearly with the message size. Therefore, the message size of getting a remote page should be minimized to reduce message overhead.

To reduce superfluous message transmission on a page fault, JIAJIA divides each page into blocks and allows a page faulting processor to fetch only those blocks that have been modified since it fetched the page last time. To do this, JIAJIA maintains a write vector table for each shared page in its home to record for each processor which block(s) has been modified since the processor fetched the page last time.

A write vector table has a write vector for each processor in the system. Each write vector contains B bits where B is the number of blocks in a page. The setting of the b th bit of the n th write vector represents that the b th block of the page is dirty with respect to the n th processor. In the current implementation, B is set to 32 and a write vector is represented by a 32-bit word.

When a processor fetches a page from its home, the corresponding write vector is checked bit-by-bit. A “1” in the b th bit of the write vector indicates that the b th block of the page contains new information for the faulting processor and should be fetched. After all dirty blocks of the missing page has been sent back to the faulting processor, the write vector corresponds to the faulting processor is cleared, indicating that now the processor has the up-to-date copy of the page.

All write vectors of the write vector table are updated when a writing to a page is detected. For writes from remote processors, the write vector table is updated according to the diff information when a diff is received and applied. For writes from the home processor, a mechanism similar to the twin and diff technique for cache pages is employed to determine modified blocks. When a processor first writes a home page in an interval, a twin is created for the home page. At the end of the interval, the modified home page is compared with its twin to determine which blocks are modified by the home processor during this interval and the write vector table is updated accordingly.

To reduce time and memory overhead of the above write vector technique, an optimization similar to the lazy diffing technique [12] is made to remove unnecessary comparison between a home page and its twin. In the optimization, a twin is not made for an exclusive home page when it is modified by its home processor. Instead, the twin creating is delayed until the exclusive home page is cached by remote processors. Similarly, the comparison between a home page and its twin is delayed (from the end of an interval) to the time when the page is first fetched by remote processors in next interval.

3.3. Home migration

It has been well recognized that the performance of home-based software DSMs is sensitive to the distribution of home pages. A good distribution of home pages should keep the home of each shared page at the processor that most frequently writes to it so that the processor can write directly to the home and least diff is produced and applied.

A problem that arises in home migration is that there must be some mechanism for all hosts to locate the new home of a migrated page. The most straightforward solution is to broadcast the new home of the migrated page to all hosts. Broadcast, however, is very expensive. The probowner method [17] provides a message saving alternative but introduces a more complex protocol in which a page faulting processor may need more than one round trip to get the faulting page. To reduce message overhead, the decision of migrating the home of a page is made at the barrier man-

ager and the page migration information is piggybacked on the barrier grant message in JIAJIA.

To further reduce migration overheads, JIAJIA migrates home of a page to a new host on a barrier only if the new home host is the single writer of the page during last barrier interval. Migrating the home of a page to the single-writing host avoids the page propagation from the old home host to the new home host because the new home host already has the up-to-date copy of the page. JIAJIA recognizes the single writer of a page in the barrier manager and informs all hosts to migrate the home of the page to the single writing processor on the barrier grant message.

With the above analysis, the home migration algorithm of JIAJIA can be described as follows.

On arriving at a barrier, the barrier arriving host sends write-notices of the associated barrier interval to the barrier manager. Each write-notice is the page-aligned address of a page modified since last barrier. The least significant bit of the write-notice is tagged as “1” if the page is valid in the cache (a modified cached page may be replaced to its home to make room for other pages in JIAJIA).

On receiving a barrier request, the barrier manager records write-notices carried on the request. Each write-notice is associated with a `modifier` field to record the host which modifies the page. If a page is modified by more than two hosts, then the `modifier` field is tagged as MULTIPLE.

After all barrier requests are received, the barrier manager broadcasts all write-notices (together with the `modifier` field for each write notice) to all hosts.

On receiving a barrier grant message, the processor checks each received write-notice and associated `modifier` field. If the write-notice and the associated `modifier` field indicate that the page is modified during last barrier interval by multiple hosts or by a single host other than itself, then the associated page is invalidated if it is cached. If the write-notice and the associated `modifier` field indicate that the page is modified by only one host and the cache of the host has a valid copy of the page (the least significant bit of the write-notice is “1”), then home of the page is migrated to the modifying host.

4. Performance evaluation

The evaluation is done in eight nodes of the Dawning-2000 parallel machine developed by the National Center of Intelligent Computing Systems. Each node has a 300 MHz PowerPC 604 processor and 256 MB memory. These nodes are connected through a 100 Mbps switched Ethernet.

The benchmarks include Water, Barnes, and LU from SPLASH2 [23], MG and 3DFFT from NAS Parallel Benchmarks [2], SOR, TSP, and ILINK from TreadMarks benchmarks [18], and two real applications EM3D and IAP18 of JIAJIA [9]. EM3D is the parallel implementation of finite difference time domain algorithm to compute the resonant frequency of a waveguide loaded cavity. The three electronic field components, three magnetic field components,

Table 1
Characteristics and sequential run time of benchmarks.

Appl.	Size	Mem. (MB)	Barrs	Locks	Seq. time (s)
Water	1728 mole.	0.5	70	1040	255.28
Barnes	16384 bodies	1.6	28	64	279.76
LU	4096 × 4096	128	256	0	605.24
SOR	4096 × 4096	64	200	0	267.86
ILINK	LGMD-1-2-3	12	740	0	487.50
TSP	-f20 -r15	0.8	0	1167	129.61
MG	256 × 256 × 256	443	592	0	295.12 ^a
3DFFT	128 × 128 × 128	96	12	0	70.89
EM3D	120 × 60 × 416	160	200	0	664.00
IAP18	144 × 91 × 18	20	5400	0	2508.00

^a Estimated from 128 × 128 × 128 MG sequential time (36.89 s). Sequential time of 256 × 256 × 256 MG is unavailable due to memory size limitation.

and eight coefficient components are allocated in shared space. The electronic and magnetic field components are updated alternatively in each iteration. Barriers are used for synchronization. Only 100 iterations are run in our test, while the real run requires 12000 iterations. IAP18 is an 18-layer climate model to simulate the global climate of the earth. It is ported to JIAJIA from SGI parallel program with DOACROSS directives. Barriers are used for synchronization. Only the climate of one day is simulated in our evaluation, while the real run normally simulates the climate of months or years. Table 1 gives characteristics and sequential run time of these benchmarks.

To evaluate the effect of the three optimizations, each benchmark is run under four configurations of JIAJIA: the normal JIAJIA (JIA), JIAJIA with the lazy write detection optimization (JIA_l), JIAJIA with both the lazy write detection and write vector optimizations (JIA_{lw}), and JIAJIA with all of the three optimizations (JIA_{lwm}). In each benchmark, shared pages are initially distributed across processors in the way that best performance is got for JIA. Besides, LU and SOR are also run with shared pages initially distributed page-by-page across processors to quantify the influence of home page distribution on the performance. We use “LU_p” and “SOR_p” to represent LU and SOR with the page-by-page initial home page distribution.

Figure 1 shows parallel execution time results of JIA, JIA_l, JIA_{lw}, and JIA_{lwm} for all benchmarks. For each benchmark, the parallel execution time of JIA rides on the top of the corresponding bar, and the parallel execution time of JIA_l, JIA_{lw}, and JIA_{lwm} are represented as percentages of the parallel execution time of JIA. In figure 1, the execution time of each parallel run is broken down into four parts: page fault (SIGSEGV service) time, synchronization time, remote request (SIGIO) service time, and computation time. The first three parts of time are collected at runtime as the overhead of the execution, and the computation time is calculated as the difference of the total execution time and the total overhead.

Table 2 gives some statistics of the parallel execution. Message amount, remote get page request count, diff count, and home page SIGSEGV count (the number of page faults

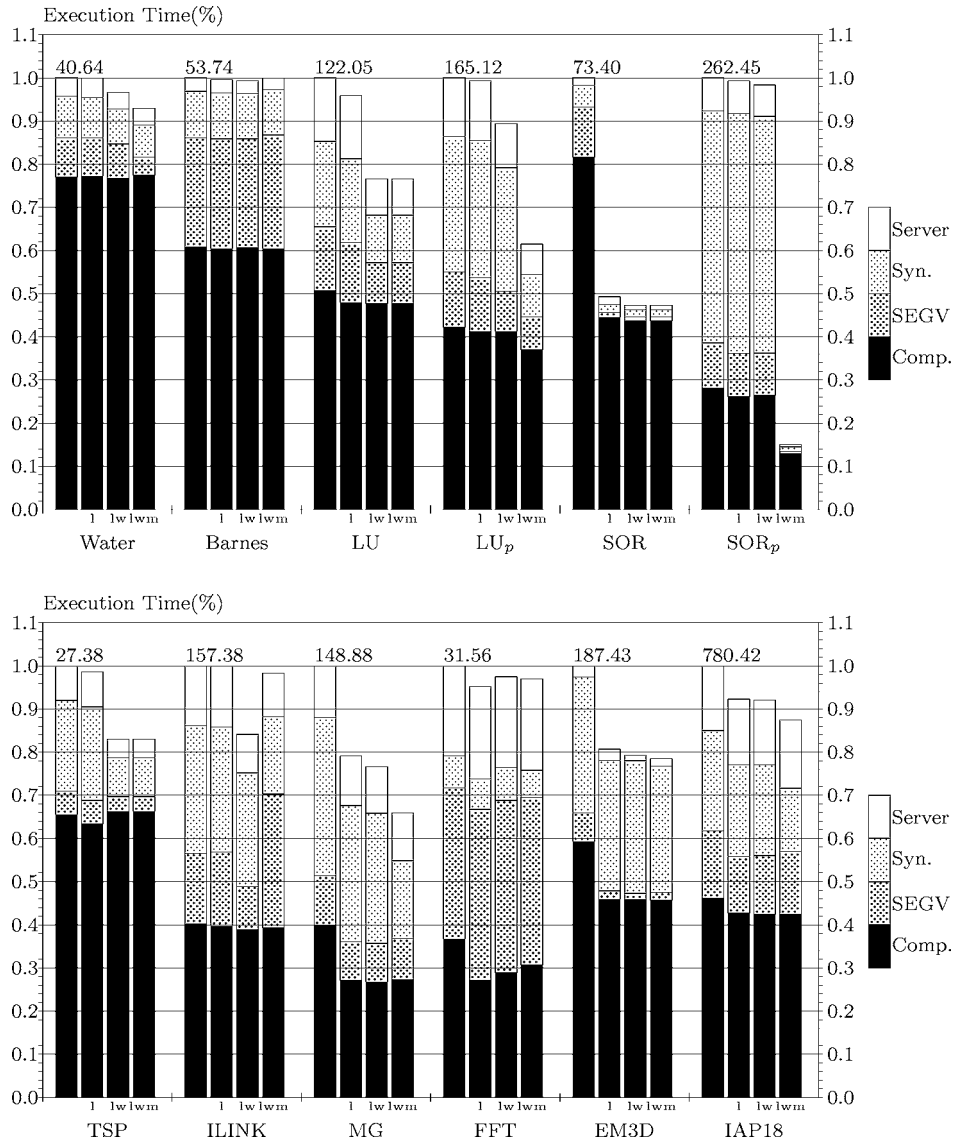


Figure 1. Parallel execution time breakdown.

caused by writing to read-only home pages) are listed in table 2 for each run of each benchmark.

4.1. Effect of lazy home page write detection

It can be seen from figure 1 that, the lazy home page write detection method significantly improves performance in LU, SOR, MG, EM3D, and IAP18.

As has been stated, the difference between JIA and JIA₁ is that JIA write-protects all home pages at the beginning of a synchronization interval, while JIA₁ keeps an exclusive home page writable until it is first accessed by remote processors in the interval. In this way, JIA₁ saves the overhead of `mprotect()` calls and page faults for home pages that are exclusively accessed by the home host itself.

Statistics in table 2 show that JIA and JIA₁ is very close in message amounts, the number of remote accesses, and the number of diffs in all applications. However, JIA₁ dramati-

cally reduces home page faults in matrix-based benchmarks LU, SOR, LU_p, SOR_p, MG, EM3D, and IAP18.

A comparison of the execution time breakdown between JIA and JIA₁ in figure 1 reveals that, as a result of the reduction of `mprotect()` calls and SIGSEGV signals in JIA₁, the synchronization and/or SIGSEGV overheads are reduced significantly in JIA₁ for LU, SOR, MG, EM3D, and IAP18. Figure 1 also shows that a great part of performance gains of JIA₁ over JIA is caused by the reduction of computation time in JIA₁. This fact implies that frequent delivery of SIGSEGV signals and calls of SIGSEGV handler have great impact on processor pipeline, processor cache hit rate, and TLB hit rate, etc.

4.2. Effect of reducing message overhead

As a method of reducing message overhead, JIA_{lw} differs from JIA₁ in that JIA₁ fetches the whole page on a page fault while JIA_{lw} only fetches updated blocks of the page. Sta-

Table 2
Runtime statistics of parallel execution.

Appl.	Msg. amt. (MB)				Get pages			
	JIA	JIA ₁	JIA _{1w}	JIA _{1wm}	JIA	JIA ₁	JIA _{1w}	JIA _{1wm}
Water	40	40	30	29	3,678	3,678	3,680	3,774
Barnes	75	75	72	72	9,692	9,692	9,692	9,692
LU	396	396	101	101	47,228	47,228	47,228	47,228
LU _p	470	469	261	118	43,451	43,331	43,331	49,303
SOR	23	23	2	2	2,800	2,800	2,800	2,800
SOR _p	533	533	51	3	2,400	2,400	2,400	2,789
ILINK	479	479	179	217	54,577	54,777	54,577	60,354
TSP	43	43	7	7	5,201	5,203	5,090	5,090
MG	653	653	609	410	17,924	17,924	17,924	23,444
3DFFT	149	149	149	149	17,976	17,976	17,976	17,976
EM3D	89	89	9	10	10,472	10,469	10,469	13,949
IAP18	2,890	2,890	2,740	2,644	269,583	269,589	269,589	300,312

Appl.	Diffs				Home SEGVs			
	JIA	JIA ₁	JIA _{1w}	JIA _{1wm}	JIA	JIA ₁	JIA _{1w}	JIA _{1wm}
Water	2,370	2,370	2,370	1,883	340	339	339	819
Barnes	7,575	7,575	7,575	7,572	848	848	848	849
LU	0	0	0	0	135,963	16,709	16,709	16,709
LU _p	118,967	118,967	118,967	1,799	16,996	2,130	2,101	15,135
SOR	0	0	0	0	818,800	2,786	2,786	2,786
SOR _p	716,600	716,600	716,600	7,166	102,200	199	2,773	2,786
ILINK	27,467	27,467	27,467	10,801	5,136	3,352	3,352	9,931
TSP	3,414	3,439	3,350	3,350	294	278	364	364
MG	27,960	27,960	27,960	5,862	368,760	21,265	21,266	27,365
3DFFT	56	56	56	56	45,064	14,344	14,344	14,344
EM3D	9,900	9,900	9,900	3,729	905,100	10,256	10,256	13,721
IAP18	137,280	137,280	137,280	94,970	2,248,940	249,831	249,831	280,553

tistics in table 2 show that JIA_{1w} is successful in reducing message amounts in most of the tested benchmarks. Consequently, figure 1 shows that JIA_{1w} obtains significant performance improvement over JIA₁ in LU, LU_p, TSP, and ILINK. Performance gains of JIA_{1w} over JIA₁ can also be observed in Water, SOR, MG, and EM3D.

Since the get page request is issued at the SIGSEGV handler and is replied at the SIGIO handler, the reduction of message size on replying get page requests helps to reduce the page fault time and remote request service time, as can be obviously seen from figure 1 in Water, LU, LU_p, SOR, TSP, ILINK, and EM3D. In TSP where lock is the mere synchronization method, lock waiting time dominates synchronization overhead because page faults dilate the critical sections. Therefore reduction of data transfer time in JIA_{1w} causes the reduction of synchronization time.

Besides, the fact that JIA_{1w} and JIA₁ have similar remote request service overhead in Barnes, MG, 3DFFT, and IAP18 indicates that the time overhead of the write vector technique is not significant because most time overhead (comparing home pages with their twins) of the technique happens on serving the page fetching request.

4.3. Effect of home migration

It can be seen from figure 1 that JIA_{1wm} outperforms JIA_{1w} in Water, LU_p, SOR_p, MG, EM3D, and IAP18, while JIA_{1w} performs better than JIA_{1wm} in ILINK. Table 3 presents the number of migrated pages in JIA_{1wm}.

The performance benefits of JIA_{1wm} over JIA_{1w} comes from the “home effect” of home-based software DSMs: no diffs generation and application is required for writes to home pages. Table 2 shows that JIA_{1wm} generates much less diffs in SOR_p, LU_p, ILINK, MG, EM3D, Water, and IAP18 than JIA_{1w}. As a result, JIA_{1wm} reduces message amounts in Water, LU_p, SOR_p, MG, and IAP18 compared to JIA_{1w}. The dramatic reduction of diffs in JIA_{1wm} implies that, though the large granularity of coherence unit in software DSMs increases the probability of false sharing, single-writing is still an important sharing behavior in many applications. This is in accordance with some previous studies [1,13].

In JIAJIA, diffs are generated at synchronization point and are applied to the home by the home host in the SIGIO handler. It can be observed from figure 1 that JIA_{1wm} introduces less synchronization time and/or server time overhead than JIA_{1w} in Water, LU_p, SOR_p, MG, EM3D, and IAP18. In LU_p and SOR_p, the computation time and SIGSEGV time are also reduced with home migration, implying that the heavy diffs overhead has impact on processor pipeline, processor cache hit rate, and TLB hit rate, etc. With the home migration scheme, the parallel execution time of LU_p (101.63 s) and SOR_p (39.23 s) is close to that of LU (93.44 s) and SOR (34.75 s).

Table 2 also shows that, JIA_{1wm} causes a little more remote page accesses than JIA_{1w} for all applications with migrated pages. Further analysis reveals that this is because when a page is written by a non-home processor, sending

Table 3
The number of migrated pages in JIA_{lwm}.

Water	Barnes	LU	LU _p	SOR	SOR _p	ILINK	TSP	MG	3DFFT	EM3D	IAP18
21	0	0	1799	0	7166	2071	0	1440	0	130	6

diff back to the home has a pre-sending effect, while home migration makes most single writes hit in home and removes the pre-sending effect. In JIAJIA, if a page is singly written by a non-home host during a barrier interval, then both the writing processor and the home processor have the valid version of the page after the barrier; if a page is singly written by its home host, then only the home host has the valid version of the page after the barrier. Table 2 also shows that the extra remote get pages of JIA_{lwm} over JIA_{lw} make the message amounts of JIA_{lwm} to be more than that of JIA_{lw} in ILINK and EM3D.

ILINK is the only benchmark in which home migration deteriorates performance though JIA_{lwm} generates only 40% diffs of that generated by JIA_{lw}. Analysis of statistics of each processor gives the reason for the anomaly in ILINK. In ILINK, home migration makes home of some frequently used pages migrated to host 0 and host 0 becomes a bottleneck in serving remote get page requests. The server time of host 0 is about 14 s in JIA_{lw}, and is more than 68 s in JIA_{lwm}. The SIGSEGV time is around 16 s for each of other hosts in JIA_{lw}, and is around 55 s for each of other hosts in JIA_{lwm}.

5. Related work

Many software DSM systems have been built since Kai Li proposed the concept of shared virtual memory and implemented the first software DSM system Ivy [17]. Some frequently cited software DSMs include Midway [3], Munin [4], TreadMarks [12], CVM [13], Cashmere-2L [22], Quarks [15], and Brazos [21]. Two important technical bases for the recent prosperity of software DSMs are the multiple writer protocol which is first implemented in Munin and the lazy implementation of release consistency which is proposed in TreadMarks.

In his Ph.D. thesis [10], Liviu Iftode proposed the concept of home-based software DSM and systematically analyzed the tradeoffs between home-based and homeless software DSMs. The concept of scope consistency was also proposed in [10]. The work of this paper is partly inspired by this thesis which pointed out the main disadvantages of home-based protocols compared to homeless ones and suggested that home migration in home-based software DSMs may be an interesting future work. However, the three optimization methods proposed in this paper are not proposed in [10].

Both the lazy home page write detection and the write vector optimization methods adopt some idea from the lazy diffing technique implemented in TreadMarks to reduce overhead about home pages in home-based systems.

Recently, [5] and [19] also implement home migration schemes in JIAJIA and in another home-based software DSM system Orion. However, their home migration

schemes migrate home of pages on remote references and maintain a probowner directory to record the new home, while our home migration scheme migrates home on barriers and piggybacks the migration information on the barrier messages to reduce message overhead.

6. Conclusion and future work

This paper studies tradeoffs between home-based and homeless software DSM protocols and introduces three optimizations to exploit advantages and offset disadvantages of the home-based protocol in the home-based software DSM JIAJIA. Performance evaluation shows that the three optimization methods can significantly improve performance in most of the tested benchmarks.

The lazy home page write detection method contributes much to the performance improvement of LU, SOR, MG, EM3D, and IAP18 through dramatic reduction of write faults on read-only home pages. Evaluation result also implies that though the reduction of `mprotect()` calls and page faults decrease synchronization and SIGSEGV overhead correspondingly, the major contribution to the execution time reduction is the reduction of processor pipeline starvation, cache pollution, and TLB pollution as a result of page fault decrement.

The write vector optimization reduces message amounts through dividing a page into blocks and fetching only updated blocks on a page fault. Water, LU, TSP, ILINK, LU_p, SOR, MG, and EM3D benefit more or less from this optimization. The main reason for the performance improvement is that the reduction of message amounts reduces the SIGSEGV and server overhead accordingly.

Benchmarks which benefit from the home migration optimization include Water, LU_p, SOR_p, MG, EM3D, and IAP18. Evaluation results show that single writing is an important sharing behavior in applications and the home migration scheme is effective in migrating home of pages to their single writer. Diffs are reduced dramatically. As negative side effects, home migration removes the effect of data pre-sending when a non-home writer sends diffs of a modified page to its home, and home migration makes host 0 the bottleneck and degrades the performance in ILINK.

Our recent works about JIAJIA include further improving the lock-based cache coherence protocol of JIAJIA, optimizing JIAJIA for cluster of SMPs, supporting JIAJIA with faster communication mechanism, implementing fault tolerance such as checkpoint mechanism in JIAJIA, and porting more real applications on JIAJIA. Further information about JIAJIA is available at www.ict.ac.cn/chpc/dsm.

Acknowledgements

We would like to thank Liviu Iftode of Rutgers University for his encouragement on the work of this paper. Acknowledgements also go to National Research Center of Intelligent Computer Systems for the allocation of computer time to run our experiments.

References

- [1] C. Amza, A. Cox, S. Dwarkadas and W. Zwaenepoel, Software DSM protocols that adapt between single writer and multiple writer, in: *Proc. of the 1997 Int. Symp. on High Performance Computer Architecture* (1997).
- [2] D. Bailey, J. Barton, T. Lasinski and H. Simon, The NAS Parallel Benchmarks, Technical Report 103863, NASA (July 1993).
- [3] B. Bershad, M. Zekauskas and W. Sawdon, The midway distributed shared memory system, in: *Proc. of the 38th IEEE Int. CompCon Conf.* (February 1993) pp. 528–537.
- [4] J. Carter, J. Bennet and W. Zwaenepoel, Implementation and performance of Munin, in: *Proc. of the 13th Symp. on Operating Systems Principles* (October 1991) pp. 152–164.
- [5] B.W. Cheung, C. Wang and K. Hwang, A migration-home protocol for implementing scope consistency model on a cluster of workstations, in: *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications* (June 1999).
- [6] A. Cox, E. Lara, C. Hu and W. Zwaenepoel, A performance comparison of homeless and home-based lazy release consistency protocols in software distributed shared memory, in: *Proc. of the 1999 Int. Symp. on High Performance Computer Architecture* (January 1999).
- [7] S. Dwarkadas, A. Schaffer, R. Cottingham, A. Cox, R. Keleher and W. Zwaenepoel, Parallelization of general linkage analysis problems, *Human Heredity* 44 (1994) 127–141.
- [8] W. Hu, W. Shi and Z. Tang, Reducing system overhead in home-based software DSMs, in: *Proc. of 13th Int. Parallel Processing Symp.* (April 1999) pp. 167–173, available at www.ict.ac.cn/chpc/hww.
- [9] W. Hu, F. Zhang, L. Ren, W. Shi and Z. Tang, Running real applications on software DSMs, in: *Proc. of 4th Int. Conf. on High Performance Computing in Asia-Pacific Region* (May 2000) pp. 148–153, available at www.ict.ac.cn/chpc/hww.
- [10] L. Iftode, Home-based shared virtual memory, Ph.D. thesis, Princeton University (August 1998).
- [11] L. Iftode, M. Blumrich, C. Dubnicki, D. Oppenheimer, J. Singh and K. Li, Shared virtual memory with automatic update support, in: *Proc. of the 1999 Int. Conf. on Supercomputing* (June 1999) pp. 175–183.
- [12] P. Keleher, S. Dwarkadas, A. Cox and W. Zwaenepoel, TreadMarks distributed shared memory on standard workstations and operating systems, in: *Proc. of the 1994 Winter Usenix Conf.* (January 1994) pp. 115–131.
- [13] P. Keleher, The relative importance of concurrent writers and weak consistency models, in: *Proc. of the 16th Int. Conf. on Distributed Computing Systems* (May 1996) pp. 91–98.
- [14] P. Keleher, Symmetry and performance in consistency protocols, in: *Proc. of the 1999 Int. Conf. on Supercomputing* (June 1999) pp. 43–50.
- [15] D. Khandekar, Quarks: Distributed shared memory as a building block for complex parallel and distributed systems, Master's thesis, Department of Computer Science, The University of Utah (March 1996).
- [16] G. Lathrop, J. Lalouel, C. Jurier and J. Ott, Strategies for multilocus analysis in humans, *PNAS* 81 (1994) 3443–3446.
- [17] K. Li, IVY: A shared virtual memory system for parallel computing, in: *Proc. of the 1988 Int. Conf. on Parallel Processing*, Vol. 2 (August 1988) pp. 94–101.
- [18] H. Lu, S. Dwarkadas, A. Cox and W. Zwaenepoel, Quantifying the performance differences between PVM and TreadMarks, *Journal of Parallel and Distributed Computing* 43(2) (June 1997) 65–78.
- [19] M. Ng and W. Wong, Adaptive schemes for home-based DSM systems, in: *Proc. of the 1st Workshop on Software Distributed Shared Memory* (June 1999) pp. 13–20.
- [20] A. Schaffer, S. Gupta, K. Shriram and R. Cottingham, Avoiding re-computation in genetic linkage analysis, *Human Heredity* 44 (1994) 225–237.
- [21] W. Speight and J. Bennett, Brazos: A third generation DSM system, in: *Proc. of the 1997 USENIX Windows/NT Workshop* (December 1997).
- [22] R. Stets et al., Cashmere-2L: Software coherent shared memory on a clustered remote-write network, in: *Proc. of the 1997 Symp. on Operating Systems Principles* (October 1997).
- [23] S. Woo, M. Ohara, E. Torrie, J. Singh and A. Gupta, The SPLASH-2 programs: Characterization and methodological considerations, in: *Proc. of ISCA'95* (1995) pp. 24–36.
- [24] Y. Zhou, L. Iftode and K. Li, Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems, in: *Proc. of the 2nd USENIX Symp. on Operating System Design and Implementation*, Seattle (October 1996).



Weiwu Hu received his B.S. degree from the University of Science and Technology of China in 1991 and his Ph.D. degree from the Institute of Computing Technology, the Chinese Academy of Sciences in 1996, both in computer science. He is currently a professor in the Institute of Computing Technology. His research interests include high performance computer architecture, parallel processing and VLSI design.
E-mail: hww@ict.ac.cn



Weisong Shi received his B.S. degree from Xidian University in 1995 and his Ph.D. degree from the Institute of Computing Technology, the Chinese Academy of Sciences in 2000. He is currently an associate research scientist in Parallel and Distributed Systems Group of Department of Computer Science, Courant Institute of Mathematical Sciences, New York University. His research interests include parallel and distributed computing systems, transparent distribution middleware for general purpose applications, mobile computing and ubiquitous computing.
E-mail: weisong@cs.nyu.edu



Zhimin Tang received his B.S. degree from the Nanjing University in 1985 and his Ph.D. degree from the Institute of Computing Technology, the Chinese Academy of Sciences in 1990, both in computer science. He is currently a professor in the Institute of Computing Technology. His research interests include high performance computer architecture, MPP systems, and digital signal processing.
E-mail: tang@ict.ac.cn