Computación y Sistemas Vol. 4 No. 2 pp. 166 - 177
 ISSN 1405-5546 Impreso en México

Two Techniques for Improvement the Speedup of Nautilus DSM

Mario D. Marino
Computing Engineering Departament - Polytechnic School of the University of Sao Paulo
Av. Prof. Luciano Gualberto 158, trav 3, Cidade Universitaria
Predio de Engenharia de Electricidade, Depto. Computacao
sala C2-24, CEP 05508-900
e-mail: mario@regulus.pcs.usp.br

Article received on February 15, 2000; accepted on August 23, 2000

Abstract

Nautilus is a home-based, page-based, multi-threaded and scope consistency-based DSM system. In this study, two techniques for improve the speedup of Nautilus DSM are investigated: write detection and page aggregation. The write detection technique consists of maintaining the pages writable only on the home nodes and only detecting writes on the cache copies in a page-based DSM. A consequence, this technique generates a less number of page faults and page requests, consequently better speedups can be achieved. The page aggregation technique consists of considering a larger granularity unit than a page, in a page-based DSM system. In order to have a fair and homogeneous comparison, a demo version of TreadMarks and JIAJIA DSM were included in this study. The benchmarks evaluated in this study are EP (from NAS), SOR (from Rice University), LU and Water N-Squared (both from SPLASH-2).

Keywords: distributed, shared, memory, DSM.

1 Introduction

The evolution and the decrement of costs of interconnection technologies and PCs have made the networks of workstations (NOWs) the most used as a parallel computer. Big projects such as Beowulf(Becker and Merkey, 1997) can be mentioned to exemplify this.

The *Distributed Shared Memory* (DSM) paradigm(Li, 1996; Stum and Zhou, 1990), which has been widely discussed for the last 9 years, is an abstraction of shared memory which permits viewing of a network of workstations as a shared memory parallel computer.

Some important DSMs, Munin(Carter, 1993), Quarks(Carter et al., 1995; Swanson et al., 1998), TreadMarks(Keleher, 1995), CVM(Keleher, 1996), JIAJIA(Eskicioglu, 1999; Hu et al., 1998a) and Nautilus(Marino and Campos, 1999a; Marino and Campos, 1999b), are page-based DSM systems. Page-based solutions have achieved good speedups for several benchmarks, but there is still available place for improvements.(Iftode et al., 1999)

In order to give a speedup improvement, in this study two techniques are evaluated for Nautilus DSM System:

- write detection(Hu et al., August 1999a; Hu et al., August 1999b);
- page aggregation(Amza et al., 1999; Keleher, 1999; Keleher, 1998).

write detection is an essential mechanism in multiple-writer protocols to identify writes to shared pages. In order to implement multiple-writer protocol, software DSMs use virtual memory page faults to detect writes to shared pages. Pages are protected at the beginning of an interval to detect writes in it. The

write detection scheme used in Nautilus(Marino and Campos, 1999a; Marino and Campos, 1999b) is based on the scheme proposed by Hu(Hu et al., August 1999a; Hu et al., August 1999b) for home-based DSMs as JIAJIA(Eskicioglu, 1999; Hu et al., 1998a). In home nodes, a write to a shared page is detected and this page will remain to be written by the home node until it is written by another node. Thus, in this interval, the page only is written by its home node and no write detection is necessary, decreasing the number of page faults and the overhead, thus improving its speedup.

In page-based DSM systems, shared memory accesses are detected using virtual memory protection, thus one page is the unit of access detection and can be used as a unit of transfer. Depending on the memory consistency model and the situation, also the diffs[1] are used as an unit of transfer. For example, in homeless lazy release consistency (LRC), such as TreadMarks, if the node has a dirty page, diffs are fetched from several nodes, when an invalid page is accessed. On the other hand, in JIAJIA, pages are fetched from the home nodes when a remote page fault occurs.

The unit of access detection and the unit of transfer can be increased by using a multiple of the hardware page size. In this way, if an aggregation of several pages is done, false sharing is increased. Besides, aggregation can reduce the number of messages exchanged. If a processor accesses several pages successively, a single page fault request and reply can be enough, instead of multiple exchanges, which are usually required. A secondary benefit is the reduction of the number of page-faults. On the other hand, false sharing can increase the amount of data exchanged and the number of messages(Amza et al., 1999).

One of the main goals of this paper is to evaluate the write detection scheme for Nautilus and its influence on Nautilus's speedup. In order to have a reference parameter of speedups, two DSMs are included in this study: TreadMarks(Keleher, 1995) and JIAJIA(Eskicioglu, 1999; Hu et al., 1998a). These two DSMs are well-known by the scientific community as optimal speedups in DSM area.

Other main goals of this paper is to evaluate the page aggregation technique(Amza et al., 1999) in Nautilus DSM system. It will be investigated what is the influence of different page sizes in Nautilus speedup. In this study, two grain sizes are used for Nautilus: 4kB and 8kB.

This study is an original contribution because the study of Amza(Amza et al., 1999) is applied with TreadMarks, a lazy release consistency homeless system, and this technique until the present was not applied in a home-based and scope consistency, multi-threaded and for Unix DSM, which are Nautilus's features. In addition,, this is the first study which combines both techniques, write detection and page aggregation, applied and evaluated on a DSM with Nautilus features.

TreadMarks, a reference of optimal speedups by the scientific community, is included in the comparison in order to have a reference parameter of speedups. Unfortunately, the results from write detection technique applied to TreadMarks DSM will not be showed nor compared here because the version (1.0.3) used in this study is a demo version, therefore, the source is not available.

The speedup results from write detection technique applied to JIAJIA DSM (version 2.1) will not be showed nor compared here because, this technique has showed any meaningful improvement in its speedups, probably due some implementation problem of this technique with JIAJIA DSM.

The evaluation comparison for write detection and page aggregation is done by applying different benchmarks: EP (from NAS), LU (kernel from SPLASH-2)(Woo et al., 1995), SOR (from Rice University) and Water N-Squared (from SPLASH-2). The environment of the comparison is an eight-PC's network interconnected by a fast-Ethernet shared media. The operating system used in each PC is Linux (2.x). Based on a combination of write detection and page aggregation, four combinations of these techniques can be created. They are: i) traditional virtual memory write detection with 4kB of page size; ii) traditional virtual memory write detection with 8kB of page size; iii) write detection with 4kB of page size; iv) write detection with 8kB of page size.

In section 2 a brief description of Nautilus is given. In section 3, JIAJIA is described. In section 4, TreadMarks is briefly described. In section 5, write detection mechanism for Nautilus is detailed. In section 6, page aggregation technique is explained. In section 7, the environment and the applications are defined. Section 8 presents the results and their analyses. Section 9 concludes this work.

[1] diffs: codification of the modifications suffered by a page during a critical section

2 Nautilus DSM

The main function of the new software DSM Nautilus is to develop a DSM with a simple consistency memory model, in order to provide good speedups, and also another one with a simpler user interface, totally compatible with TreadMarks and JIAJIA. This idea is very similar to the ideas utilized by JIAJIA, mentioned in the studies of Hu(Hu et al., 1998a) and Eskicioglu(Eskicioglu, 1999), but Nautilus makes use of some other techniques, which distinguishes it from JIAJIA. These techniques will be mentioned below. In order to be portable, it was developed as a runtime library like TreadMarks, CVM and JIAJIA, because there is no need to change the operating system kernel(Carter, 1993).

Nautilus is a page-based DSM, as TreadMarks and JIAJIA. In this scheme, pages are replicated through the several nodes of the net, allowing multiple reads and writes(Stum and Zhou, 1990), thus improving speedups. By adopting the multiple writer protocols proposed by Carter(Carter, 1993), false sharing is reduced and good speedups can be achieved. The mechanism of coherence adopted is write invalidation(Stum and Zhou, 1990), because several studies (Carter et al., 1995; Eskicioglu, 1999; Iftode et al., 1999; Keleher, 1995) show that this type of mechanism provides better speedups for general applications. Nautilus, as JIAJIA does, uses scope consistency model, which is implemented through a locked-based protocol(Hu et al, 1998b).

Nautilus is the first multi-threaded DSM system implemented on top of a free Unix platform that uses the scope consistency model because:

1)there are versions of TreadMarks implemented with threads, but it does not use the scope consistency memory model;

2)JIAJIA is a DSM system based on scope consistency, but it is not implemented using threads.

3)CVM(Keleher, 1996) is a multi-threaded DSM system, but it uses lazy release consistency and at the moment, it does not have a Linux based version.

4)Brazos(Speight and Bennet, 1997) is a multi-threaded DSM and uses the scope consistency, but it is implemented on a Windows NT platform.

Let's summarize Nautilus features: i) scope consistency only sending consistency messages to the owner of the pages and invalidating pages in the acquire primitive; ii) multiple writer protocols; iii) multi-threaded DSM: threads to minimize the switch context; iv) no use of SIGIO signals(which notice the arrival of a network message); v) minimization of diffs creation; vi) primitives compatible with TreadMarks, Quarks and JIAJIA; vii) network of PCs and Linux 2.x; viii) UDP protocols.

Nautilus is different from other DSMs in several ways. First, its implementation is multi-threaded, thus it minimizes the context switches overheads, and in addition, does not use SIGIO signals in its implementation. Second, as JIAJIA does, Nautilus manages the shared memory using a home-based scheme, but with a directory structure of all pages instead of only a structure of the relevant pages (cached), used by JIAJIA. Third, a different memory organization from JIAJIA.

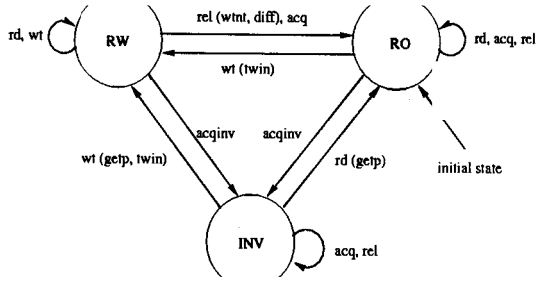
To improve the speedup of the applications submitted, Nautilus uses two techniques: i)multi-threaded implementation; ii) diffs of pages that were written by the owner are not created.

The multi-threaded implementation of Nautilus permits: 1) minimization of context switch; 2) no use of SIGIO signals.

The major part of all DSM systems created until today implemented on top of an Unix platform uses SIGIO signals to activate a handler to take care of the arrival of messages which come from the network. Some examples of DSMs that use the SIGIO signal are TreadMarks and JIAJIA. One of the threads remains blocked trying to read messages from the net. While blocked, it remains asleep, thus non consuming CPU. This technique decreases the overhead of the DSM and allows to give as much CPU time as possible to the user program. Thus, Nautilus is the first scope consistency DSM system of the second generation which does not use the SIGIO signal in its implementation. The use of a multi-threaded implementation permits to eliminate this overhead to take SIGIO signals and activate its respective handler, in all arrivals of messages.

2.1 Lock-based Coherence protocol

Nautilus follows the lock-based protocol proposed by JIAJIA(Hu et al, 1998b), because of its simplicity, thus minimizing the overheads: the pages can be in one of three states: Invalid(INV), Read-Only (RO) and Read-Write(RW). *"Initially all pages are in RO state at all home nodes. Ordinary read and write accesses to a RW page or read access to a RO page, or acquire and release on an INV or RO page do not cause any transi-*



Notes

rd, wt:	read, write
acq, rel:	acquire, release
acqinv:	invalidate the page on acquire
getp:	get the page from its home
wtnt:	send write-notices to the lock
diffs:	send page diffs to home(s)
twin:	create a twin of the page

Figure 1: JIAJIA Coherence Protocol (Hu et al, 1998b)

tion. Like the shared pages, each lock has a home node. On a release operation, the node generate diffs for all modified pages and sends them to their respective homes eagerly. Also, the processor sends a release request to the home node of the lock, along with the write-notices (list of a modified pages) for the associated critical section. Similarly, an acquiring node sends a request to the owner of the lock and waits until it receives a grant message for the lock. Multiple acquire requests for a lock are queued at the locks home processor. When the lock becomes (or is) available, a lock grant message is sent to the first node in the queue, piggybacked with the applicable write-notices. After receiving the lock grant message, the acquiring node invalidates the pages listed in the write-notices and continues with its normal operation(Hu et al, 1998b)."

In summary, the home nodes of the pages always contain a valid page, and the diffs corresponding to the remote cached copies of the pages are sent to the home nodes. A list with the pages to be invalidated in the node is attached to the acquire lock message.

JIAJIA(Eskicioglu, 1999; Hu et al., 1998a) only contains information of the relevant pages, the cached copies of the pages, because it argues that it reduces the space overhead of the system. On the other hand, Nautilus maintains a local directory structure for all pages, since it does not occupy a relevant space and does not increase the overhead of the system. Inversely, this helps increasing the speedup of the system.

In Nautilus, the owner nodes of the pages do not need to send the diffs to other nodes, according to the scope consistency model. So, diffs of pages written by the owner are not created, which is more efficient than the lazy diff creation of TreadMarks. The implementation of the state diagram of Figure 1 is done in Unix with the *mprotect()* primitive, where pages can be in RO, INV or RW states, thus their states can be changed easily.

2.2 Data Distribution and Consistency Related Informations

Nautilus distributes its shared pages across all nodes and each shared pages has a home node. When home nodes access their home pages, no page faults occurs. When remote pages are accessed, page faults occur, and these pages are fetched from their home node and cached locally. Instead of JIAJIA, Nautilus does not have a replacing mechanism of cached pages, since in Linux, they are replaced as memory size increases.

Nautilus uses the scope consistency memory model(Iftode et al., 1996), where the coherence of cached pages is maintained through write-notices[2] kept on the lock (lock-based). As a result from the multiple-writer protocols technique application, diffs are sent to their home nodes. The implementation of Nautilus barrier mechanism is very similar to JIAJIA, because it is believed that with less time consuming and a lower number of messages, it becomes more efficient. Now, the acquire/release mechanism of Nautilus is briefly described. In order to signal the end of the critical section, a release message is sent to the manager. Taking in advantage of the fact of sending this message, the write notices are piggy-backed on the release message. On the acquire, the processor which is doing it sends a lock request to the manager. When granting the lock, the manager piggy-backs write-notices associated with this lock on the grant message. At the acquire, the processor, which is doing it, invalidates all cached pages that are notified as obsolete by the received write-notices. On a barrier, all write notices of all locks are cleared.

[2]write-notices: indication of which pages were modified during the critical section

3 JIAJIA

JIAJIA (Eskicioglu, 1999; Hu et al., 1998a) is another important DSM system that uses scope consistency, which can be interpreted as an intermediary consistency model between release consistency and lazy release consistency or also be interpreted as a kind of implementation of release consistency. So, diffs are transmitted in each critical section to maintain the consistency. Thus, the consistency model used by JIAJIA is the scope consistency model, only sending consistency messages to the owner of the pages and invalidating pages in the acquire primitive.

To summarize the JIAJIA features: i) scope consistency(Iftode et al., 1996) home based, minimizing the number of consistency messages through the net; ii)multiple writer techniques; iii) primitives compatible with TreadMarks; iv)UDP protocols, minimizing network protocols overhead; v) data distribution can be chosen by the user (over the network nodes).

The main objective of JIAJIA(Eskicioglu, 1999; Hu et al., 1998a) is to make the minimization of the overheads of diff creation and storage as simple as possible, thus minimizing the number of consistency messages through the net. The most interesting feature of JIAJIA is its simple ideas: home based, so the diffs are transmitted only to the owner of the pages and not to several nodes, minimizing the number of messages through the net; the user knowing the behavior of his program, chooses a data distribution which is more appropriated, which allows for better speedups.

4 TreadMarks

The consistency model used by TreadMarks is the lazy release consistency(Keleher, 1995), so the propagation of the modifications which occurred during a critical section are delayed until the next acquire. By using multiple writer protocols and the lazy release consistency model, the speedups of TreadMarks are well-known, making it one of the most used DSM systems.

To summarize TreadMarks features: i) lazy release consistency and its variations (Keleher, 1995), minimizing the number of consistency messages through the net; ii) multiple writer techniques of Munin (Carter, 1993); iii) primitives compatible with m4; iv) IBM SP2, Sun Sparc, PCs; v) AIX, Solaris, free Unix (Linux 2.x); vi) UDP protocols to minimize network protocols overhead.

The speedups of TreadMarks made it the main DSM used by the scientific community as a reference of optimal speedups. Thus, it makes sense to compare it with other DSMs in order to have an accurate evaluation of its performance.

The efficiency of TreadMarks is mainly derived from its lazy release consistency model. The major drawback of adopting this model is the need of large amounts ofmemory to store the diffs all over the user's application execution. Thus, the size of the benchmarks used to evaluate the speedups of the DSM system can be compromised if there is not enough memory to execute the program or if the operating system does swap. If it cannot use enough size to run the benchmarks, the computation versus synchronization becomes unfavorable for using a DSM system.

5 Write detection

Nautilus has several common features with JIAJIA, i.e., by observing the item 3, where JIAJIA features were mentioned, from feature i) to iv) Nautilus is similar to JIAJIA, which permits Nautilus to adopt the JIAJIA's write detection scheme.

As other DSMs like TreadMarks(Keleher, 1995) and JIAJIA(Eskicioglu, 1999; Hu et al., 1998a) do, Nautilus uses virtual memory page faults to detect writes to shared pages. Shared pages are protected at the beginning of an interval (several critical sections). When the first write to a shared page occurs, a SIGSEGV signal is delivered, and in this moment the page can be written without protection. At the end of the critical section, as JIAJIA does, Nautilus send the write-notices related about the shared pages.

Several studies (Amza et al., 1999; Bershad et al., 1993; Carter, 1993; Eskicioglu, 1999; Hu et al., 1998a; Iftode et al., 1999; Keleher, 1995; Keleher, 1999) show that the detection of writes to shared pages presents significant overheads. Other studies show that applications with large shared data set and good data distribution, the writes hit in the home. Thus, it is possible to conclude that for applications with large shared data set and good data distribution, if the write detection would be eliminated from the home node, a great overhead can be decreased. And going further, it is not necessary to become the page from the read-write state to read-only state(Hu et al, 1998b) if only

the home node writes to this page. Concluding, if the home page is written in some interval, several *mprotect-s()* and SIGSEGV handlers calls are saved, improving the DSM's speedup. If the home page is not written by its home in the interval, some unnecessary invalidations of remote cached pages can occur, thus more remote accesses.

The study of Amza(Amza et al., 1999) showed that in many applications, single-writer constitutes the dominant part of the sharing behavior and shared pages are normally written by the home node (owner) for a certain interval. Nautilus implements its write detection scheme which recognizes automatically a single write to a shared page by its home node, presuming that the page will continue to be written by its home node until the page is written by remote nodes.

6 Page Aggregation

In terms of implementation, following the other DSMs directions, in Nautilus there is a handler responsible for request a page from a remote node when a segmentation fault occurs. Following the Figure 1 diagram, when a page is accessed and it is in the INV state a SIGSEVG signal is generated and the respective handler, as it was said before, requests the page from the home node. When the page arrives the primitive *mprotect()* changes the state from INV to RO.

When the page is written, another SIGSEGV signal is generated and the primitive *mprotect()* changes the state of the page from RO to RW, as can be seen in Figure 1. As is shown in this figure, after the generation of the diffs, also with the *mprotect()* primitive, pages go to RO state again. And, when the write-notices arrive, indicating the pages are modified by other nodes, pages go to INV state (again with the use of *mprotect()* primitive). The primitive *mprotect()* permits to consider a granularity multiple of a page, thus giving the same permission for a region multiple of a page. Thus, this fact gives the condition to modify more than one page at the same time, which is named page aggregation technique.

The study (Amza et al., 1999) says that if aggregation is done, false sharing is increased and aggregation reduces the number of messages exchanged. In addition, the processor accesses several pages successively, a single page fault request and reply can be enough, instead of multiple exchanges of requests and replies, which are usually required. The study (Amza et al., 1999) also shows that there is a reduction of the number of page-faults, but false sharing can increase the amount of data exchanged and the number of messages.

By changing the page size default (4kB) to, for example, 8kB using *mprotect()* primitive in Nautilus, it is possible to evaluate the effects of the incremented size in page fault reduction in the speedups.

7 Experimental Platform and Applications

The results reported here are collected on an eight-PC's network. Each node (PC) is equipped with a K6 - 233 MHz (AMD)processor, 64 MB of memory and a fast Ethernet card (100 Mbits/s) . The nodes are interconnected with a hub. In order to measure the speedups, the network above was completely isolated from any other external networks. Each PC runs Linux Red Hat 6.0. The experiments are executed with no other user process.

Related to page aggregation, two sizes are considered for page size: 4kB, which is the default (memory hardware) and 8kB, which is multiple of 4kB. And, related to write detection, a version with this technique will be compared to a version without this technique.

Based on the combination of write detection and page aggregation techniques, as it has already been mentioned, four versions of Nautilus DSM will be evaluated and compared to TreadMarks and JIAJIA. They are with Nautilus using: 1. traditional virtual memory write detection with 4kB of page size; 2. traditional virtual memory write detection with 8kB of page size; 3. write detection with 4kB of page size; 4. write detection with 8kB of page size.

The test suite includes some programs: EP (from NAS), LU (from SPLASH-2(Woo et al., 1995)), SOR (from Rice University) and Water N-Squared (from SPLASH-2). SPLASH-2 is a collection of parallel applications implemented to evaluate and design shared memory multiprocessors.

The Embarrassingly Parallel (EP) program from the NAS benchmark suite generates pairs of Gaussian random deviates with a scheme that is well suited for parallel computation and tabulates the number of pairs successively. The only communication and synchro-

nization in this program is summing up a ten-integer list in a critical section at the end of program (Hu et al., 1998a). The parameter used in EP program is M=2^{24}.

The LU kernel from SPLASH-2 factors a dense matrix into the product of a lower triangular and upper triangular matrix. The NxN matrix is divided into an nxn array of bxb blocks (N = n*b) to exploit temporal locality on submatrix elements. The matrix is factored as an array of blocks, allowing blocks' to be allocated contiguously and entirely in the local memory of processors that own them. LU is a kernel from SPLASH2 benchmarks that has a rate computation/communication $O(N^3)/O(N^2)$, which increases with the problem size N. The nodes frequently synchronize in each step of computation and none of the phases are fully parallelized(Hu et al., 1998a).

SOR from Rice University solves partial differential equations (Laplace equations) with an Over-Relaxation method. There are two arrays, black and red array allocated in shared memory. Each element from the red array is computed as an aritmethic mean from the black array and each element from the black array is computed as an aritmethic mean from the red array. Communication occurs across the boundary rows on a barrier. The SOR from Rice University solves Laplace partial equations. For a number of iterations it has two barriers each iteration and communication occurs across boundary rows on a barrier. The communication does not increase with the number of processors and the relation communication/computation reduces as the size of problem increases (Hu et al., 1998a).

Water is an N-body molecular simulation program that evaluates forces and potentials in a system of water molecules in the liquid state using a brute force method with a cutoff radius. Water simulates the state of the molecules in steps. Both intra- and inter-molecular potentials are computed in each step. The most computation- and communication-intensive part of the program is the inter-molecular force computation phase, where each processor computes and updates the forces between each of its molecules and each of the n/2 following molecules in a wrap-around fashion(Eskicioglu, 1999).

application	EP	LU	Water	SOR
t(1)	121.80	350.90	2983.00	29.10
t(8).Tmk	15.62	54.45	403.20	8.66
t(8).JIA	15.60	65.44	429.54	18.22
t(8).NautV4k	15.65	54.32	426.40	7.66
t(8).NautWD4k	15.67	49.60	422.07	4.37
t(8).NautV8k	15.66	55.52	426.69	6.54
t(8).NautWD8k	15.67	49.28	427.55	4.14
Sp.Tmk	7.80	6.44	7.39	3.36
Sp.JIA	7.81	5.36	6.94	1.60
Sp.NautV4k	7.78	6.46	7.00	3.80
Sp.NautWD4k	7.77	7.07	7.07	6.66
Sp.NautV8k	7.77	6.45	6.99	4.45
Sp.NautWD8k	7.78	7.12	6.98	7.03
SG.JIA	1	5882	106	1177
SG.NautV4k	2	7980	10210	12425
SG.NautWD4k	2	440	824	927
SG.NautV8k	1	5029	855	7912
SG.NautWD8k	1	245	671	458
gp.JIA	1	3110	1924	855
gp.NautV4k	1	1528	505	118
gp.NautWD4k	1	340	448	74
gp.NautV8k	1	1232	440	72
gp.NautWD8k	1	195	331	33

Table 1: Table comparing TreadMarks, JIAJIA and Nautilus (virtual and write detection, with grain sizes of 4kB and 8kB)

8 Result Analysis

Before presenting the results and their analyses, it is necessary to emphasize that the execution time for number of nodes=1 in all evaluated benchmarks is obtained from the sequential version of the benchmarks without any DSM primitive. So, the primitive used to allocate memory to obtain the sequential time (number of nodes=1) is **malloc()**, default primitive of C programming.

In order to have an accurate, homogeneous and fair comparison, the same programs are executed using TreadMarks (version 1.0.3), JIAJIA (version 2.1) and Nautilus (version 0.0.1).

There are some constraints with TreadMarks version (1.0.3) used:

i)the applications were executed and the speedups measured using *Nautilus* running on up to 8 nodes;

ii)bigger input sizes: the shared memory size is limited in this version;

iii)the source code was not available: only time and speedups can be obtained from this version, thus it was not possible to obtain number of page faults and SIGSEGV signals.

The data input size N used in the LU and SOR evaluation is 1792x1792; the number of iterations for the SOR benchmark is 10 . The number of steps used in Water is 25 and the number of molecules is 1728. For EP, M=2^{24}.

Table 1 shows some features and results of the benchmarks: sequential time (t(1)), eight-processor parallel run time(8), speedup (Sp), remote get page request counts (gp) and number of local SIGSEGV of Nautilus(SG). The sequential time t(1) was obtained from the sequential program without no DSM primitives and malloc() primitive. In order to evaluate the write detection speedup, remote get page request counts and the number of local SIGSEGVs of Nautilus are taken. For Table 1, Tmk means TreadMarks, JIA means JIAJIA, NautV means Nautilus with the traditional virtual memory write detection and NautWD means Nautilus with the write detection. NautWD assumes that a page will only be written by its home in the future barrier interval, keeping its home page writable if only the home writes to it. When 4k is mentioned, it means that Nautilus is using grain size of 4kB and 8k means Nautilus using 8kB of grain size. Thus, for Nautilus using: i) Nautilus versions with 8kB of page size: Naut8k; ii) Nautilus versions with 4kB: Naut4k; iii)traditional virtual memory write detection with 4kB of page size, the notation is NautV4k; iv)traditional virtual memory write detection with 8kB of page size: NautV8k; v) traditional virtual memory write detection versions: NautV; vi) write detection with 4kB of page size: NautWD4k; vii) write detection with 8kB of page size: NautWD8k; viii) write detection versions: NautWD.

Except for EP benchmark, some general conclusions can be taken from Table 1: the number of SIGSEGVs and the number of page requests decreased when both mechanisms, write detection and page aggregation, were employed. It can be seen lower remote page accesses (gp rows) in NautWD than in NautV, because of the correct home page write assumption. It was said before that if a home page is assumed to be written by its home next barrier interval and if the home does not write it, the assumption causes unnecessary invalidation of remote cached pages and consequently some other page requests. This justifies the decrement of the number of SIGSEGVs and page requests. Related to aggregation, it was said before that if the page size increases, the number of page requests and SIGSEGVs decreases, which justifies the observed behavior.

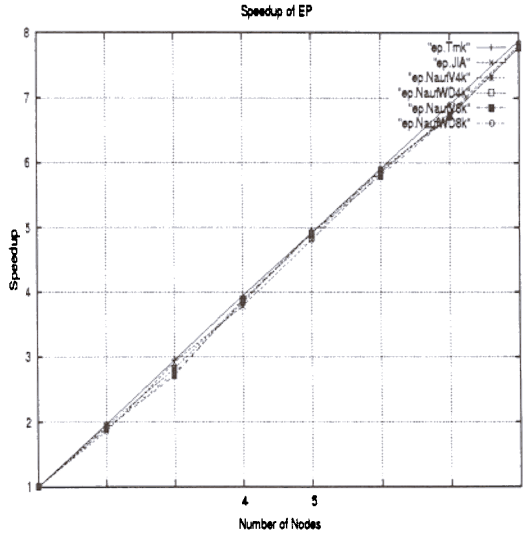


Figure 2: speedups of EP: M=2^{24}

Now, for each benchmark, the behavior of the techniques is analyzed.

8.1 EP

EP (NAS) has a small amount of communication and a small number of messages transmitted through the net, as can be observed from Table 1. By looking at Figure 2, all Nautilus's versions, JIAJIA and TreadMarks have similar speedups, only differing about 1-3%. Due to the small number of messages generated by this applicative, it was possible neither to notice the difference when the write detection nor the page aggregation techniques, when both were applied.

8.2

By looking at Figure 3, the speedups of LU can be seen. It can be noticed from this figure that mainly the write detection technique improved Nautilus's speedup and the page aggregation technique practically did not

improve Nautilus speedup.

Analyzing the write detection technique in LU , matrices are distributed across processors in a way that each processor writes to its home part of the matrices in the computing.(Becker and Merkey, 1997) Since the computation of an iteration is synchronized with barriers and passing a barrier causes all shared pages to be write-protected in traditional virtual memory, page faults occur for writing all home pages in an iteration. The method does not write protect shared pages on a barrier, and writing to home pages of a processor can process without any SIGSEGV.

From Table 1, for eight nodes, it can be noticed that for Naut4k (page size of 4kB), the write method improved the speedup up to 9.78% and for Naut8k (page size of 8kB), the method improved the speedup of Nautilus up to 0.7%. The increasing of the speedups can be justified by observing the number of SIGSEGVS from Table 1 an order of magnitude lower for the technique versions compared to traditional versions. The number of page requests (gp rows) were reduced an order of magnitude when this method was applied.

Analyzing the page aggregation technique, for eight nodes, for NautV, this method decreased the speedup up to 0.1% and, for NautWD, this method improved the speedup up to 0.7%. The justification of the decrease of speedup is increasing of false sharing. The number of SIGSEGVs was reduced by 36.98% for NautV and by 44.32% for NautWD; in addition the number of page requests was reduced by 19.37% for NautV and by 42.64% for NautWD.

With both methods applied, write detection and page aggregation, the speedup of Nautilus was improved up to 10.21%, for eight nodes.

Comparing TreadMarks generically with NautV4k, both have similar behavior, some times TreadMarks outperforming NautV4k (six nodes for example) and some times NautV4k outperforming TreadMarks (for five and seven nodes). This alternate behavior is due to choice of data distribution and the needing of diff storage of TreadMarks, which can cause in some cases the swapping of the operating system(Marino and Campos, 1999b). With the adoption of the mechanisms (write detection and aggregation), Nautilus outperforms TreadMarks up to 10.56%.

Comparing with JIAJIA, for eight nodes, TreadMarks outperforms it by 20.15%, and Nautilus with both techniques outperforms it by 32.83%. Better data

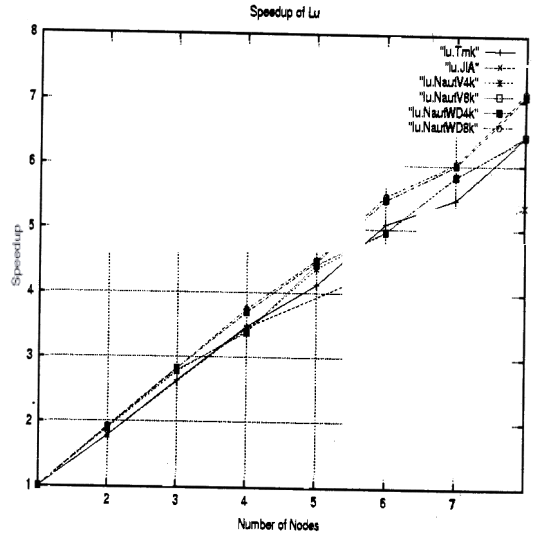


Figure 3: speedups of LU: N=1792

distribution (choice of the page owners) improves data locality and gives a lower cold start up time to distribute shared data are factors which also contributes to the better speedups of TreadMarks and Nautilus over JIAJIA. In addition, the elimination of SIGIO signals minimizes the overheads of Nautilus over the other DSMs. In terms of number of SIGSEGVS (from Table 1), JIAJIA has 26.29% lower than NautV4k, but comparing to NautV8k, the later has 14.51% more speedup than the former; for the versions, Nautilus is one order of magnitude lower than JIAJIA. In terms of page requests, NautV has lower number of them (50.87% and 60.38%) and for NautWD, it is one order of magnitude lower than JIAJIA.

8.3 Water

By looking at Figure 4, the speedups of Water can be seen. It can be noticed from this figure that mainly the write detection technique improved Nautilus's speedup and the page aggregation technique did not improve Nautilus speedup.

From Table 1, for eight nodes, it can be noticed that for NautV4k the write method improved the speedup up to 1.00% and for NautV8k, the method decreased the speedup of Nautilus by 0.15%, this because of the false sharing effect. By observing the number of SIGSEGVS from Table 1, NautWD4k version has an

order of magnitude lower, and NautWD8k has 21.52% lower SIGSEGVs . In the same way, the number of page requests (gp rows) were reduced: 11.28% for 4kB's page size and 24.77% for 8kB's page size.

Analyzing the page aggregation technique, for eight nodes, for NautV, this method decreased the speedup by 0.15% and, for NautWD versions, decreased the speedup by 1.30%, both speedup reductions justified because of the increasement of the false sharing effect. With this technique, the number of SIGSEGVs was reduced an order of magnitude for NautV and by 18.57% for NautWD; in addition the number of page requests was reduced by 12.87% for NautV and by 26.11% for NautWD.

Considering the two techniques, the write detection and page aggregation, the method proportioned better speedup for Nautilus, up to 1.00%, for eight nodes. The problem with this application is its high synchronization, which is the dominant feature. Also, the false sharing effect increases with the increasement of the page size, which contributes to decrease the speedup when the aggregation technique is applied.

Confronting TreadMarks generically with NautV4k, TreadMarks outperforms it up to 5.7%, due to its better data distribution and the semaphore implementation of Nautilus is until in developing. Confronting with JIAJIA, for eight nodes, TreadMarks outperforms it up to 6.48% and NautV4k outperforms it up to 1.87%. In terms of number of SIGSEGVs, JIAJIA has an order of magnitude lower than NautV4k, and 70-80% lower for the other versions, mainly due its better data distribution than Nautilus.

8.4 SOR

By looking at Figure 5, the speedups of SOR can be seen. It can be noticed from this figure that mainly the write detection technique improved Nautilus's speedup and the page aggregation technique practically did not improve Nautilus speedup.

In SOR, as the same way in LU, matrices are distributed across processors in a way that each processor writes to its home part of the matrices in the computing.(Becker and Merkey, 1997) Since the computation of an iteration is synchronized with barriers and passing a barrier causes all shared pages to be write-protected in traditional virtual memory, page faults occur for writing all home pages in an iteration. The method does not write protect shared pages on a barrier, and writing to home pages of a process without any SIGSEGV.

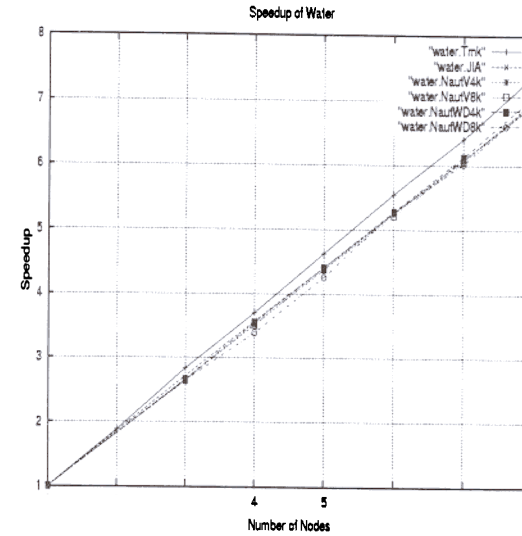


Figure 4: speedups of Water: **1728 molecules** and **25 steps**

From Table 1, for eight nodes, it can be noticed that for Naut4k, the write method improved the speedup up to 75.26% and for Naut8k, up to 57.98%. The increasement of the speedups can be justified by observing the number of SIGSEGVS from Table 1, an order of magnitude lower for the NautWD versions compared to NautV versions. The number of page requests were reduced too.

Analyzing the page aggregation technique, for eight nodes, for NautV version, this method improved the speedup up to 17.11% and, for NautWD version, up to 5.56%. In this technique, the number of SIGSEGVs was reduced by 36.32% for NautV version and by 50.59% for NautWD version; also the number of page requests was reduced by 38.98% for NautV and by 55.41% for the NautWD.

With both methods applied, write detection and page aggregation, the speedup of Nautilus was improved up to 85.00%, for eight nodes.

Confronting TreadMarks generically with NautV4k, it outperforms TreadMarks up to 13.10%. This happens because of the better data distribution (choice of the page owners) adopted by itself improving the matrix data locality (minimizing the number of messages through the net) and resulting in a lower cold start up

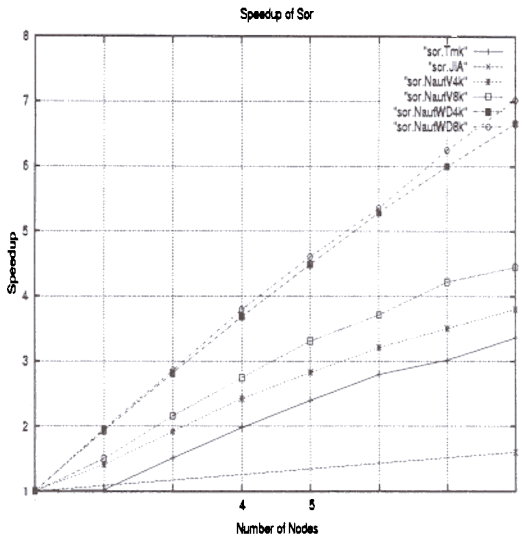


Figure 5: speedups of SOR: 1792x1792

time to distribute shared data. In addition, the avoidance of SIGIO signals and the multi-threading help to improve the speedup of SOR. In addition, with all techniques proposed and applied in this paper for Nautilus, it outperforms TreadMarks up to 109.22%.

For this benchmark, it seems that JIAJIA probably has an implementation problem for this benchmark, so it is not considered for speedup analysis. For number of SIGSEGVs, JIAJIA outperforms NautV4k and NautV8k by one order of magnitude and for the NautWD versions, they outperform JIAJIA by 21.24% and 61.09%. For the number of page requests, JIAJIA is 86.20% higher than NautV4k and has an order of magnitude higher page requests than other Nautilus's versions.

9 Conclusion

The contribution of this study is an evaluation of two techniques, write detection and page aggregation, on a DSM with Nautilus features. In order to have a fair and homogeneous comparison, two well known DSMs are used: TreadMarks and JIAJIA. In addition, these three DSMs are compared with respect to speedups, number of page requests and number SIGSEGVs.

The study shows that for LU application, both proposed techniques have improved Nautilus speedup up to 10.21%. For SOR application, which have shared data and single-writer behavior, the write detection technique can improve its speedup. For SOR, a speedup increasement of 85.00% for Nautilus was obtained. For other applications with high synchronization like Water, both techniques do not contribute to increase the speedup. The number of SIGSEGVs and the number of request page faults have reduced by one order of magnitude in several cases, mainly when the write detection technique was applied.

In future works, other benchmarks will be evaluated in this comparison. Also, an improved version of JIAJIA will be evaluated and a complete version of TreadMarks to measure the number of page faults and SIGSEGVs.

References

Amza C., Cox A. L., Dwarkadas S., Jin L. J., Rajamani K., Zwaenepoel W., Adaptive Protocols for Software Distributed Shared Memory, Proceedings of IEEE, Special Issue on Distributed Shared Memory, pp. 467-475, March 1999.

Becker D., Merkey P.; *Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs*, Proceedings, IEEE Aerospace, 1997.

Bershad B. N. , Zekauskas M. J. , SawDon W. A. , *The Midway Distributed Shared Memory System* , COMPCOM 1993.

Carter J. B., Khandekar D., Kamb L., *Distributed Shared Memory: Where We are and Where we Should Headed*, Computer Systems Laboratory, University of Utah, 1995.

Carter J. B., *Efficient Distributed Shared Memory Based on Multi-protocol Release Consistency*, PHD Thesis, Rice University, Houston, Texas, September, 1993.

Eskicioglu, M.S., Marsland T.A., Hu W, Shi W.; *Evaluation of the JIAJIA DSM System on High Performance Computer Architectures*, Proceeding of the Hawai'i International Conference on System Sciences, Maui, Hawaii, January, 1999.

Hu W., Shi W., Tang Z., *JIAJIA: An SVM System Based on a new Cache Coherence Protocol,* technical report no. 980001, Center of High Performance Computing, Institute of Computing Technology, Chinese Academy of Sciences, January, 1998.

Hu W., Shi W., Tang Z.; *A lock-based cache coherence protocol for scope consistency*, Journal of Computer Science and Technology, 13(2):97-109, March, 1998.

Hu W., Shi W., Tang Z., *Adaptive write detection in Home-based Software DSMs*, to appear in Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, August 1999, Redondo Beach, CA.

Hu W., Shi W., Tang Z., *Write detection in Home-based Software DSMs*, to appear in Proceedings of Euro-Par'99, August 31-September 2, Tolouse, France.

Iftode L., Singh J.P., Li K; *Scope Consistency: A bridge between release consistency and entry consistency.* Proceedings of the 8th ACM Annual Symposium on Parallel Algorithms and Architectures (SPAA'96), pp. 277-287, June, 1996.

Iftode L., Singh J. P.; *Shared Virtual Memory: Progress and Challenges*; Proceedings of the IEEE, Vol 87, No. 3, March 1999, 1999.

Keleher P., *Lazy Release Consistency for Distributed Shared Memory*, PHD Thesis, University of Rochester, Texas, Houston, January 1995.

Keleher P., *The Relative Importance of Concurrent Writers and Weak Consistency Models*, in Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS-16), pp. 91-98, May 1996.

Keleher P., *Update Protocols and Cluster-based Shared Memory,* In Computer Communications, 22(11), pp. 1045-1055, July 1999.

Keleher P., Update Protocols and Iterative Scientific Applications, In The 12th International Parallel Processing Symposium, March 1998.

Li K, *Shared Virtual Memory on Loosely Coupled Multiprocessors*, PHD Thesis, Yale University, 1986.

Marino M. D., Campos G. L., Sato L. M.; *An Evaluation of the Speedup of Nautilus DSM System*, IASTED PDCS99, pp 250-255, Boston, USA, November, 1999.

Marino M. D.; Campos G. L.; *A Preliminary DSM Speedup Comparison: JIAJIA x Nautilus*, to be published at HPCS99.

Speight E., Bennett J. K., *Brazos: A third generation DSM system*, In Proceedings of the 1997 USENIX Windows/NT Workshop, pp. 95-106, August, 1997.

Stum M. , Zhou S., *Algorithms Implementing Distributed Shared Memory*, University of Toronto, IEEE Computer v.23 , n.5 , pp.54-64, May 1990.

Swanson M., Stoller L., Carter J., *Making Distributed Shared Memory Simple, Yet Efficient*, Computer Systems Laboratory, University of Utah, technical report , 1998.

Woo S., Ohara M., Torrie E., Singh J.P., Gupta A.; *The SPLASH-2 programs: Characterization and methodological considerations. In Proceedings of the 22th Annual Symposium on Computer* Architecture, pages 24-36, June, 1995.

***Mario D. Marino**, graduate: Electrical Engineering at Escola Politecnica da Universidade de Sao Paulo, December, 1992. Master of Science: Computing Engineering Department at Escola Politecnica da Universidade de Sao Paulo, October, 1996. PHD: Computing Engineering Department at Escola Politecnica da Universidade de Sao Paulo, January, 2001.*