An Evaluation of the Speedup of Nautilus DSM System

Mario Donato Marino, Geraldo Lino de Campos and Líria Matsumoto Sato[‡]

Computer Engineering Department- Polytechnic School of University of São Paulo

Abstract

In this paper, a new DSM system called Nautilus is described and evaluated with four different benchmarks running on a 8 PC's network. The evaluation experiments show that Nautilus provides good speedups, which are comparable to other DSMs like JIAJIA and TreadMarks. The benchmarks evaluated in this study are: LU (kernel from SPLASH-2) and SOR (from Rice University).

key words: distributed shared memory, DSM

1 Introduction

The evolution and the decreasement of costs of interconnection technologies and PCs has become the networks of workstations (NOWs) the most used as a parallel computer. Big projects such as Beowulf[11] can be mentioned to exemplify this.

The *Distributed Shared Memory* (DSM) paradigm[8], which has been largely discussed for the last 9 years, is an abstraction of shared memory which permits to view a network of workstations as a shared memory parallel computer. By moving or replicating data[8], shared memory uniform accesses are done by the different nodes, implementing in this way the DSM main aim. These movimentation and/or replication of data, guarantee its consistency, allowing programs done for phisically shared memory machines to be easily ported and developed[1], since to develop message passing programs is more difficult than to develop shared memory programs.

In the last years the DSM area has great diffused, with the development of a large number of consistency models and DSM systems. Carter [1] has classified the DSM evolution in two generations:

1) a big number of consistency messages and the adoption of the sequential consistency model; one can exemplify this with the Ivy [6];

2) a drastic reduction of the number of consistency messages by the adoption of the release consistency model, applying techniques to reduce the false sharing; several examples can be exemplified: Munin[2], Quarks[7], TreadMarks[3], CVM[10], JIAJIA[4] and Nautilus[5].

Commonly, DSM comparisons base on simulations, rather than confronting execution results, for example, two different DSM systems over a computer network. Also, because to compare executions is more accurate and more correct than to compare simulations of the DSMs.

There are a few previous papers [3] [9] comparing different DSM systems; however, most of then evaluate DSM systems using networks with different operating systems, and different interconnection systems.

The main goal of this paper is to present and to evaluate a new DSM System called Nautilus in an accurate way, confronting it with two of the most important DSM systems, which have largely been used by several research groups in the scientific community: TreadMarks and JIAJIA. This confront is done in a same PC's network, so with the machines and a free operating system. Once they are evaluated under the same conditions, the results of this comparison would tend towards accurate, homogeneous and fair comparisons. In addition, TreadMarks and JIAJIA speedups are compared.

The comparison of Nautilus speedups with TreadMarks and JIAJIA speedups is done by applying two different benchmarks: LU (kernel from SPLASH-2)[15] and SOR (from Rice University). The environment of the comparison is a 8PC's network interconnected by a fast-ethernet shared media. The operating system used in each PC is Linux (2.x).

2 Nautilus DSM

The first motivation in working in the DSM area, is to acquire experience in it, in order to create a laboratory of experimenting different DSM systems, thus evaluating and confronting them.

After this, to gain practice experience in DSM area and techniques, the motivation of a new DSM creation, called Nautilus was born. Thus, the main motivation of the new software DSM Nautilus is to develop a DSM with a simple consistency memory model, in order to

*{mario,geraldo,liria}@regulus.pcs.usp.br ; Address: Avenida Professor Luciano Gualberto 158, trav3, Cidade Universitária, São Paulo, Brazil, CEP: 05508-900

†paper number: 302-207

provide good speedups, and also a with a simpler user interface, totally compatible with TreadMarks and JIAJIA. This idea is very similar with the ideas utilized by JIAJIA, mentioned in the studies of Hu[4] and Eskicioglu[12], but Nautilus makes use of some other techniques, which distinguishes it from JIAJIA. These techniques will be mentioned bellow. In order to be portable, it was developed as a runtime library as TreadMarks, CVM and JIAJIA, because there is no need to change the operating system kernel[2].

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[8], thus improving speedups. By adopting the multiple writer protocols proposed by Carter[2], false sharing is reduced and good speedups can be achieved. The mechanism of coherence adopted is write invalidation[8], because several studies [2, 3, 4, 12] 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[13]. It is believed that this model is a relaxed view of release consistency, thus it is simple, so it produces a little overhead. If well applied, the release consistency model yet can produce good speedups[1].

Nautilus is different from other DSMs in several ways. First, its implementation is multithreaded, 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, as it uses the scope consistency model, which can be viewed as a relaxed view of release consistency model, it has a simple implementation, which produces lower overheads. Fourth, a different memory organization from JIAJIA.

2.1 Scope Consistency

Following the DSM classification proposed by Carter[1], as it was said before (item 1), DSMs which belong to the first generation have a big number of consistency messages. This high number of consistency messages fills the net up, decreasing so much the speedups.

DSMs which belong to the second generation by adopting the relaxed memory models, reduce dramatically the number of consistency messages, delaying the emition of them until the synchronization points. In the second generation, several consistency models appeared. The most importants are the Lazy Release Consistency(LRC), entry Consistency (EC) and Scope Consistency(ScC).

LRC, which is a more relaxed form of release consistency[3, 10](RC), *"guarantees that the shared data are the most recent at any acquire operation"*.[12] As RC does, LRC allows multiple writer writers, by the creation of the twins and diffs[1] [2], thus improving the concurrence, so the speedups. The propagation of the modifications (diffs) done in a critical section is delayed until the next acquire[3], "*even if only a few or none of these data are used later.*"[12] For example, as TreadMarks does, when an invalidate protocol is adopted, the diffs of the pages are only sent if the node does a lock and access these pages.

In the EC model[9], each shared data is associated with a lock, in such way that only the data associated with the current lock is guaranteed to be the most recent. So, this association implies that the programmer needs to modify its programs, which can be a difficult task depending on the data structures contained in it.

The ScC model[14] is proposed to bridge between entry release consistency and release consistency. Instead of associate locks with shared data, locks are associated with critical sections. As it was said in the Eskicioglu[12] study: "*Any modification made during a critical section is guaranteed to be seen by other processors that later enter the same section on the same scope. General rules for the ScC can be summarized as follows: i) an ordinary memory access (read or write) is allowed to happen with respect to any other processor only after all previous acquires are carried out; ii) a release is allowed to happen with respect to any other processor only after all previous ordinary memory accesses to the region (protected by the same lock) are performed; iii) synchronization accesses are sequentially consistent.*"

Most of the applications written for LRC and RC can run under ScC without any modification or with a little modification. Thus, Nautilus can run the same programs developed to TreadMarks and JIAJIA, only changing the names of the primitives.

Concluding, Nautilus's coherency protocol implements the ScC model because of its advantages and simplicity.

2.2 Lock-based Coherence protocol

Nautilus follows the lock-based protocol proposed by JIAJIA[12], because of its simplicity, thus minimizing the overheads. Figure **??**(from [12]) summarizes the state transitions.

Based on this protocol, the pages can be in one of three states: Invalid(INV), Read-Only (RO) and Read-Write(RW). The following text, from [12]), with the word "processor(s)" changed to "node(s)" explains the transition state diagram : "*Multiple reads are allowed,*

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

thus several nodes have cached copies of a page concurrently. By resuming the protocol: 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 transition. 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. Thus, as it was said previously, this model can be considered a release consistency mode. *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.*

Resuming, 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[3] only contains information of the relevant pages, the cached copies of the pages, because it argues that it *reduces the space overhead the system*[4]. On the other hand, Nautilus maintains a local directory structure for all pages, since it does not occupy a relevant space and does not increases the overhead of the system. Differently, this helps increasing the speedup of the system.

Following JIAJIA [3, 12] idea, 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, what it's believed to be more efficient than the lazy diff creation of TreadMarks.

2.3 Nautilus's Techniques

Nautilus is the first multithreaded 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 scope consistency memory model;
2. JIAJIA is a DSM system based on scope consistency, but it is not implemented using threads.
3. CVM[10] is a multithreaded DSM system, but uses lazy release consistency and as of now, it does not have a Linux based version.

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) multithreaded 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; viii) operating under Linux 2.x; ix) UDP protocols.

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

The multithreaded 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 slept, thus non consuming CPU. This technique decreases the overhead of the DSM and allows to give as 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 SIGIO signal in its implementation. The use of a multithreaded implementation permits to eliminate this overhead to take SIGIO signals and activate its respective handler, in all arrivals of messages.

On the same way that TreadMarks and JIAJIA do, also Nautilus is worried about network protocols. So, it also uses UDP protocol to minimize overheads.

Nautilus also cares about compatibility of primitives. Its primitives are simple and totally compatible with TreadMarks, JIAJIA and Quarks; as a result there is not any need of code rearrangements. One example of this compatibility is that in this study, LU and SOR are converted from JIAJIA and SOR from TreadMarks, basically changing the name of the primitives.

As TreadMarks and JIAJIA do, Nautilus also is worried about synchronization messages. To minimize the number of messages, the synchronization messages would carry consistency information, minimizing the emission of the last ones.

3 Experimental Evaluation Platform and Applications

The results reported here are collected on a 8 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 5.0. The experiments are executed with no other user process.

The test suite includes four programs: LU (from SPLASH-2[15]), and SOR (from Rice University). SPLASH-2 is a collection of parallel applications implemented to evaluate and design shared memory multiprocessors.

"*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 then. LU is a kernel from SPLASH-2 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*".[4]

"*SOR from Rice University solves partial differential equations (Laplace equations) with a Over-Relaxation method. There are two arrays, black and red array allocated in shared memory. Each element from red array is computed as an aritmethic mean from black array and each element from black array is computed as an aritmethic mean from 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*". [4]

In order to show the compatibility of Nautilus with TreadMarks and JIAJIA primitives, the program SOR was taken from TreadMarks and and the program LU was taken from JIAJIA distribution, only changing the names of its primitives, without to rearrange any code.

4 Result Analysis

Before presenting the results and their analysis, 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) and JIAJIA (version 2.0). Only a **few results** were obtained with the current version of **JIAJIA** (supposing some implementation problem). JIAJIA only execute correctly programs with **even** number of nodes.

Due to the limitation of the 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** for both benchmarks LU and SOR, **were not possible** to be evaluated.

Each application is executed with two different data input size to evaluate the effect of problem size on the speedups.

The data input size N used in the LU evaluation are **N = 1024** and **N = 1792**. The data input size of red and black matrix used in SOR evaluation are **1024x1024** and **1792x1792**. The number of iterations for the SOR benchmark is **10** .

4.1 LU

Looking at the figures 1 and 2, the speedups of the three DSMs can be seen under two different input size: N=1024 and N=1792.

The increasement of N improves the locality of LU, when it is submitted to both DSMs, thus improving its speedups, as it can be seen, for example, with 7 nodes, switching from N=1024 to N=1792, when the speedups change from less than 5.5 to more than 5.5, for TreadMarks.

With a number of nodes less than 3 nodes, for the two different input size, the speedups of the three DSM systems are very similar. With four nodes, for N=1024 and N=1792, JIAJIA has the best speedup. Taking advantage of its data distribution and the lower number of diffs sent, for N=1024, TreadMarks is faster than Nautilus and JIAJIA and, as a consequence, has better speedups. But, for N=1792, Nautilus is faster than TreadMarks and JIAJIA. This behavior occurs because with this N value, due to lazy release consistency model adopted by TreadMarks, too much diffs are stored, causing the swapping of the operating system.

In terms of percentage, for N=1024, TreadMarks outperforms Nautilus about 2% until 5 nodes and 10.5% with 7 nodes and Nautilus is faster than JIAJIA up to 4%. For N=1792, Nautilus outperforms Tread-

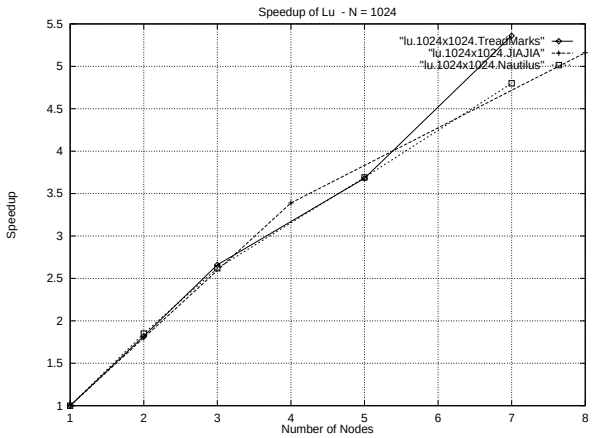

Figure 1: speedups of LU: N=1024

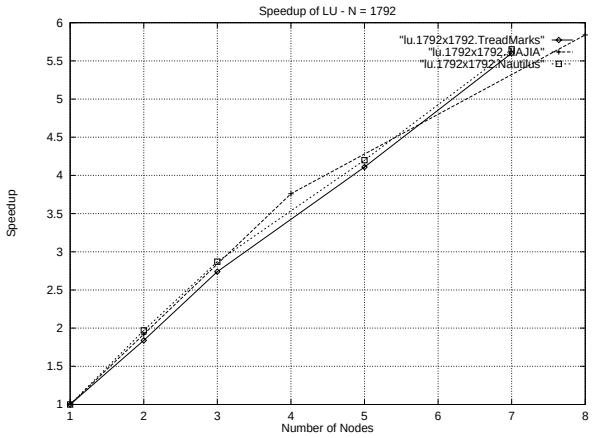

Figure 2: speedups of LU: N=1792

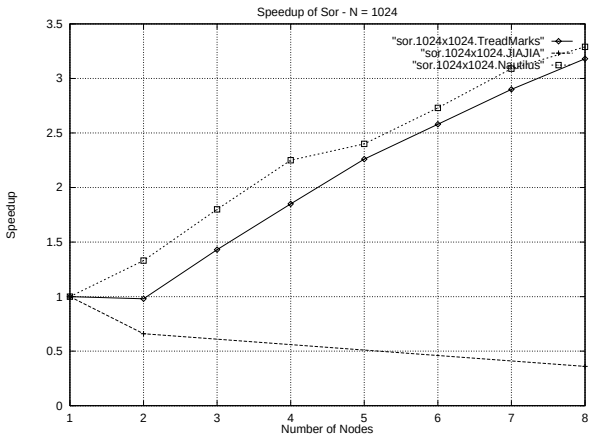

Figure 3: speedups of SOR: N=1024

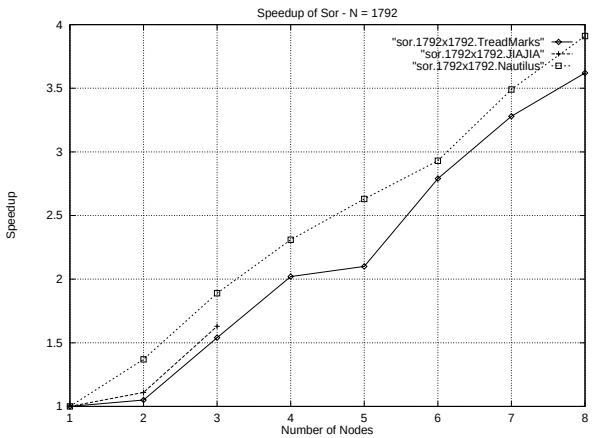

Figure 4: speedups of SOR: N=1792

Marks about 2 until 6.60% (2 nodes) and TreadMarks outperforms JIAJIA around 7%.

Summarizing, the good data distribution used by TreadMarks, the lower number of diffs sent due to its consistency model, minimizes the number of messages through the net and gives it better speedups than JIAJIA and Nautilus. Although JIAJIA and Nautilus use the same memory consistency model, the use of multithreading and the avoidance of SIGIO signals makes Nautilus faster than JIAJIA. So, the two release consistency models, lazy release consistency model (adopted by TreadMarks) and the scope consistency model (adopted by Nautilus and JIAJIA) have a good and similar behavior, providing good speedups.

4.2 SOR

Looking at the figures 3 and 4, the speedups of the three DSMs can be seen under different input size: N=1024 and N=1792.

The speedups of JIAJIA as can be seen in the figures 3 and 4, and , are very unusual . Therefore any related speedups are not considered for SOR analysis.

For both N=1024 and N=1792, and for 2 to 8 nodes Nautilus outperforms TreadMarks. For N=1024 the speedup difference reaches up to 26.31%. And, for N=1792, the speedup difference reaches up to 23.36%. The excellent speedup of Nautilus happens because of the better data distribution (choice of the page owners) addopted by itself improving the matrix data locality (minimizing the number of messages through the net) and giving a lower cold start up time to distribute shared data. Also the avoidance of SIGIO signals and the multithreading help to improve the SOR speedup.

Changing N from 1024 to 1792, the locality of the SOR improves, also improving the speedup.

Justifying the best speedups of Nautilus: with a low number of processors, the speedup difference between Nautilus and TreadMarks is higher, because of the higher cost of lazy release consistency to maintain the directory on a page fault. It is needless to alloc twins and diffs when a page is written in its owner(node), which decreases the overhead and also the data distribution (choice of the page owners), multithreading and the avoidance of SIGIO signals, improve the Nautilus speedup.

5 Conclusion

In this paper a new DSM called Nautilus was presented and described. Its features were justified and compared with other well known DSM systems: TreadMarks and JIAJIA. In addition, Nautilus's speedup was confronted and compared with TreadMarks and JIAJIA DSMs, on the same environment, with the same computers, network and operating system.

For the LU applicative, TreadMarks has the best speedups for N=1024 , reaching until 10% more speedup than Nautilus. For N=1792, Nautilus has the best speedup, until 6.6% faster than TreadMarks. For the SOR, Nautilus has the best speedup also, reaching until 28.87% more speedup than TreadMarks.

As shown, Nautilus has good speedups, comparable to other well-known DSMs, surpassing some of them depending on the application. The use of multithreading, the avoidance of SIGIO signals and a good data distribution (choice of the page owners) also help to improve Nautilus speedup. Other data distribution are undergoing study, in order to further improve Nautilus speedup. The lazy release consistency model (TreadMarks) and the scope consistency model (JIAJIA and Nautilus) presented good speedups for all the benchmarks evaluated in this paper.

Since only a few results were obtained with the current version of JIAJIA, it will be possible to compare the speedups of an improved version of this DSM with TreadMarks and Nautilus.

By evaluating other problem size: one of the problems of this study is the Demo version of TreadMarks that was evaluated. This version, that has the same efficiency of a normal version of the software, has two problems limitations: the size of the shared memory and a limited number of nodes. Another improved version is being acquired and due its memory release consistency model and its need to store diffs, more memory is being provided to evaluate it better.

References

[1] 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.

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

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

[4] 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.

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

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

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

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

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.