# Reducing Communication Overhead and Page Faults in SDSM Platforms

Artemis A. Christopoulou[1] and Eleftherios D. Polychronopoulos[1]

High Performance Information Systems Laboratory
Computer Engineering & Informatics Department
University of Patras, 26500 Rio, GREECE
{aac,edp}@hpclab.ceid.upatras.gr

**Abstract.** In this paper we present a new dynamic, cache coherence protocol for Software Distributed Shared Memory (SDSM) systems that adopt the scope-consistency model[7]. We initially outline our basic protocol, called Reduced Message Protocol (RMP), and then propose two enhancements: the Multiple Home RMP (RMP-MH) and the Lock Migration RMP (RMP-LM). The experimentation we conducted with the proposed protocols, exhibits significant improvements by reducing two of the major latency factors in SDSM platforms: the total communication messages and the overall number of page faults. To demonstrate the efficiency and the effectiveness of the RMP protocols, we used SPLASH as well as synthetic application benchmarks.

**Keywords:** Cache Coherence Protocols, Memory Consistency Models, Software DSM Systems, Clusters, Grids.

## 1 Introduction

The advances of the last two decades in software environments for distributed and cluster computing, along with the improvement in the networking technology, have brought clusters in the proscenium of today's massive multiprocessor systems. When it comes to most enterprize and IT applications, cluster computing of today dominates over tightly coupled multiprocessor systems or the proprietary supercomputer designs of the previous decade. On the other hand, programming clusters of computational nodes, in order to take advantage of parallel execution, is more complex than programming for shared memory systems (the default model supported by multiprocessors). The communication cost of message passing implementations has fallen drastically over the years, but remains far higher than shared memory models, especially in the case of fine grain parallel execution.

Our research builds upon previous approaches which combine the convenience and low cost of clusters with the programming simplicity of shared memory systems, hiding away the distributed architecture via efficient communication libraries. These libraries provide for the transparent integration of message passing in a shared memory model as seen by the applications or the programmer. SDSM

models, have been the subject of significant research in the past two decades and constitute the underlying framework for our research work.

In this paper we present a new dynamic Reduced Message Protocol (RMP) for Software Distributed Shared Memory systems (SDSM) which adopts the scope-consistency model[7]. Our main objective is the improvement of the SDSMs' cache coherence protocol, enabling them to function in Wide Area Networks as well as improving their performance in small to medium-sized clusters. Our ultimate goal is to incorporate the proposed cache coherence protocol in wide area clusters as well as define a computational platform in the Grid for parallel processing, based on new SDSM platforms with advanced features in communication and computation mechanisms[13]. Recently there is active research interest in this area, namely, using SDSMs platforms for investigating and testing new methods for Grids[11][12].

Software DSMs are typically categorized into write-invalidate and write-update, on the basis of the cache coherence protocol used to inform the processors for memory page modifications. In write-update protocols the modifications of a page are sent to the processors and the page copies are updated, while in write-invalidate protocols only write-notices for a modified page are sent and the page copies are invalidated. Several protocols have been proposed which adapt between write-invalidate and write-update. [4], [5] and [6] are some of them. In this paper, we propose a new adaptive cache coherence protocol, which was implemented in the Software DSM JiaJia[1]. Our protocol exploits the characteristics of the Scope consistency model[7] used by Jiajia, in order to improve the system's performance.

The rest of the paper is organized as follows. Section 2 describes the new protocol, section 3 presents the experimental evaluation, section 4 describes related work. Finally, the conclusions of the paper is drawn in Section 5.

## 2 Cache Coherence Protocol

The proposed protocol is based on the JiaJia protocol and specifically in it's write-vector version [1]. The main functions of JiaJia and our protocol have been analyzed in [14]. The protocol's objective is the reduction of page faults inside critical sections and the reduction of the total sent messages. This objective is achieved by piggybacking to the acq-grant message the modifications of the pages, expected to be used by the acquirer. Since the information is sent in existing messages, the number of messages is greatly reduced, although the total amount of transferred bytes remains the same. This provides for a significant benefit, since a great part of communication burden is due to message initialization, a cost greatly reduced when the same bytes are sent in fewer messages. The pages which are piggybacked in the lock grant message are only the pages which have the same home as the lock, since only these pages are available to the processor that sends the lock-grant message. Thus, when a processor acquires a lock, it receives within the acq-grant message updates of modified pages instead of simply write-notices. Consequently, during the next critical section re-

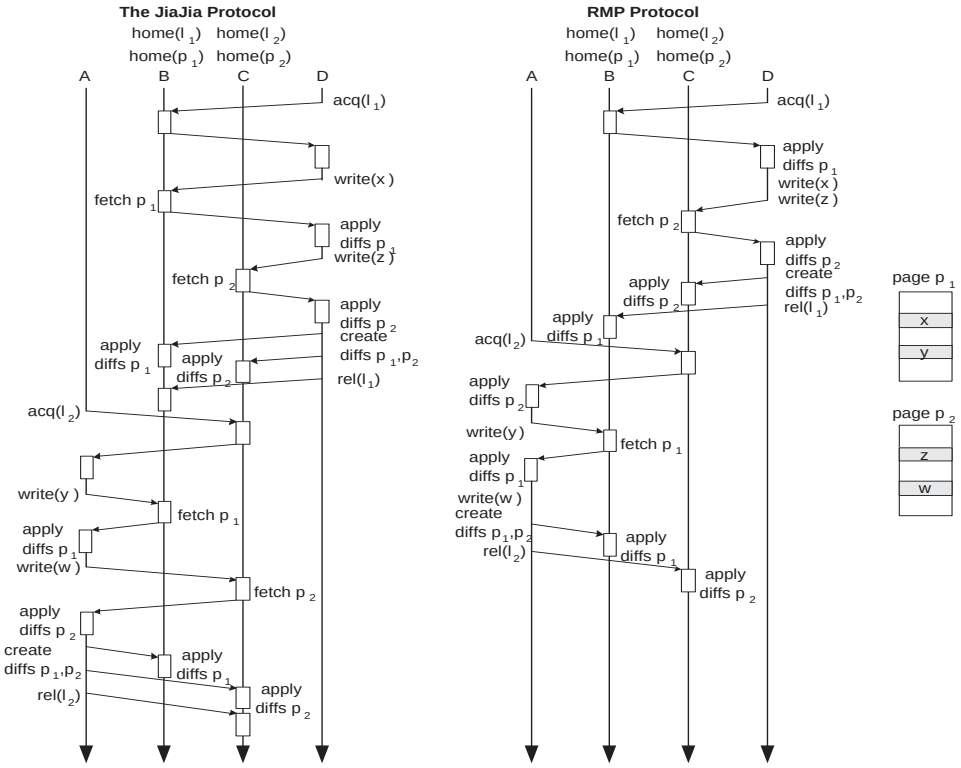


**Fig. 1.** JiaJia - RMP Protocol

quests for the corresponding pages will be facilitated by the local copies instead of resulting in page faults, minimizing the exchange of messages with the page owner. In order to piggyback to a lock grant message as many pages as possible, we perform page migrations as follows. When a locks home processor receives write notices for the lock, it records the pages that were modified. The pages that are frequently modified during critical sections of one lock migrate to the locks home processor.

There is a small chance that a page contains two different variables which are protected by different locks with different homes. In this case the question that comes up is to which processor the page should migrate. Our protocol follows a greedy approach according to which the page migrates to the first processor that asks for it and cannot migrate to any other processor after that. In addition, we have implemented two alternate approaches to the above problem: lock-migration and multi-home pages.

– Multi-home Pages: In this protocol variation(RMP-MH), a page is allowed to have more than one home, so that it can migrate to the homes of multiple locks associated with it. The additional actions taken in this case are that

the diffs of a modified page are sent to all the home processors, and that diffs are also created if a page is modified by one of its home processors.
- Lock Migration: In this protocol variation(RMP-LM), if a page is associated with two different locks, then the page migrates to the home processor of one of the locks, and so does the second lock.

A significant further improvement of our protocol, also presented in this paper, regards the reduction of the diff messages, which are the messages that contain a page's modifications from some node. In the initial JiaJia protocol, a node sent diff messages for the pages modified during the critical section of the lock before issuing a lock release message. We reduce the diff and subsequently the total messages as follows. For the pages that have the same home node as the lock, we do not send extra messages for the page diffs, but piggyback them to the lock release message. The operations of the JiaJia and our RMP and variations protocols, are shown in figure 1.

## 3 Experiments

Our main protocols *RMP* as well as its two variations, *RMP-MH* and *RMP-LM* have been implemented and compared against the initial protocol of JiaJia *JiaJia* and it's write vector version *JiaJiaWV*.

Experiments were carried out on two different systems, a 4-processor SMP and a 4-node cluster. The SMP consisted of four processors 2 of them having 512KB and the other 2 having 1024KB cache and 512MB total main memory. Each node of the cluster had two Intel Pentium III processors with 256KB per-processor cache and 256MB per-node main memory. The nodes were interconnected with a 1000Mbps Ethernet network. In all systems, the operating system used was Linux and the application binaries were created with the gcc compiler.

Our protocols were evaluated using four applications, Water and Raytrace from the SPLASH suite, the TSP problem from the JIAJIA SDSM distribution and one of our synthetic benchmarks.

Water simulates forces between different molecules. It uses an array of data structures, each corresponding to a molecule. The array is statically divided into equal parts, each of which is assigned to a processor. Processors use locks to protect the update of force values relating to the molecules. Barriers are used to ensure that all processes perform calculations corresponding to the same time step, as well as to guarantee global memory consistency at the beginning of each step.

Raytrace renders a three-dimensional scene using ray-tracing. A hierarchical uniform grid(similar to an octree) is used to represent the scene, and early ray termination and antialiasing are implemented. A ray is traced through each pixel in the image plane, and reflects in unpredictable ways of the objects it strikes. Each contact generates multiple rays, and the recursion results in a ray tree per pixel. The image plain is partitioned among processors in contiguous blocks of pixel groups, and distributed task queues, and the primitives that describe the scene. In this application the data access patterns are highly unpredictable.

| Application | Variable | JiaJia | JiaJiaWV | RMP | RMP-LM | RMP-MH |
|---|---|---|---|---|---|---|
| Synthetic | Total Messages | 109458 | 109458 | 5474 | 5474 | 5474 |
| | Messages in bytes | 399210264 | 21139224 | 20552784 | 20552784 | 20552784 |
| | Getp Requests | 48120 | 48120 | 700 | 700 | 700 |
| | Diff Messages | 4800 | 4800 | 42 | 42 | 42 |
| TSP | Total Messages | 11399 | 11341 | 4212 | 2527 | 3921 |
| | Messages in bytes | 28525052 | 5422928 | 5956708 | 5904940 | 5771676 |
| | Getp Requests | 3394 | 3355 | 730 | 312 | 273 |
| | Diff Messages | 1503 | 1510 | 513 | 97 | 838 |
| Water | Total Messages | 1986 | 2000 | 1916 | 1921 | 2058 |
| | Messages in bytes | 4186660 | 2402076 | 2453024 | 2460712 | 2613048 |
| | Getp Requests | 426 | 433 | 380 | 388 | 233 |
| | Diff Messages | 189 | 189 | 193 | 201 | 406 |
| Raytrace | Total Messages | 22567 | 22414 | 12329 | 12382 | 12280 |
| | Messages in bytes | 33803804 | 12008220 | 11785268 | 11787940 | 11782188 |
| | Getp Requests | 3993 | 4003 | 1199 | 1199 | 1200 |
| | Diff Messages | 3827 | 3784 | 1529 | 1536 | 1523 |

**Table 1.** Protocols' communication variables

TSP solves the travelling salesman problem using a branch and bound algorithm. The major shared data structures of TSP include a pool of partially evaluated tours, a priority queue containing pointers to tours in the pool, a stack of pointers to unused tour elements in the pool, and the current shortest path. Processors evaluate the partial paths successively and alternately until the shortest path is found. Locks are used to ensure exclusive accesses to shared objects.

The last application is a synthetic application which we used to stress the proposed protocol and evaluate its maximum performance. In this application there are four locks each of which protects forty variables in forty different memory pages. For each lock, each processor modifies the forty variables belonging to the lock and this procedure is repeated one hundred times.

| Programming Model | Synthetic | TSP | Water | Raytrace |
|---|---|---|---|---|
| Posix Threads | 0,04 | 14,96 | 2,92 | 16 |
| JiaJia | 18,8 | 12,26 | 5,56 | 63 |

**Table 2.** Execution Time in SMP

In the SMP each application was programmed and executed twice, once using Posix Threads and once using the SDSM JiaJia. These experiments were made in order to show the overhead of an SDSM and the communication cost among its nodes. In table 2 we see the total execution time for each application for each case.

The results show that all the applications besides TSP take more than double time to execute using JiaJia compared to Posix threads. This clearly indicates that the overhead of an SDSM is significant, and therefore there is great need to make it as efficient as possible.

The main way to improve the SDSM's efficiency was by reducing the total amount of messages sent among the SDSM nodes. Our experiments prove that the cost in time of sending a message is given by the following type:

$$t(size) = t_{init} + t_{send} * size \tag{1}$$

, which means that the cost of sending a message is analogous to its size plus an initialization cost. In the SMP system it was measured that $t_{init}$ is 228,8007 and $t_{send}$ is 0,0235, while in the cluster $t_{init}$ is 121,1614 and $t_{send}$ is 0,0337. All time values are measured in microseconds.

Since the initialization cost of sending a message is that big, it is expected that if we send the same amount of bytes in a smaller number of messages, we can achieve a performance improvement. Our new protocol was designed to achieve this goal.

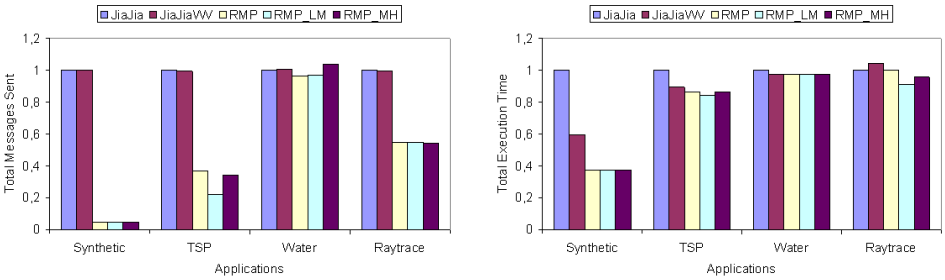


**Fig. 2.** Messages sent

In Figure 2 we can see for each application and for each protocol the total number of messages and it's total execution time, while in table 1 we can more details about see the messages' reduction.

All the results shown have been normalized with the *JiaJia* results.

Checking the number of sent messages, we see a great reduction in our synthetic application. In this application forty pages are modified during a critical section and these pages must be sent to the other nodes before they are accessed by them. In our protocol modifications of these pages are piggybacked to the lock release and lock grant messages and as a consequence the total number of sent messages is greatly reduced. Tsp uses locks as its synchronization method. Every key used protects a lot of pages and since modifications of the pages are sent with the lock release and lock grant messages, the total number of sent messages of the application is significantly reduced. In Water both locks and barriers are used for the synchronization, but the most page modifications occur in critical

sections enclosed by barriers. Consequently the number of total sent messages is little affected by our protocol. Contrary to Water, Raytrace uses only locks for the synchronization which produces a reduction in the sent messages as in the other two applications.

The reduction of sent messages in our protocols lead also to a reduction of the execution time. In our synthetic application the reduction reaches the 63% compared to the initial protocol of JiaJia and the 37% compared to JiaJia's write vector version. In Raytrace the reduction is 9% and 13%, in TSP 16% and 6% while Water shows a little reduction of the total execution time.

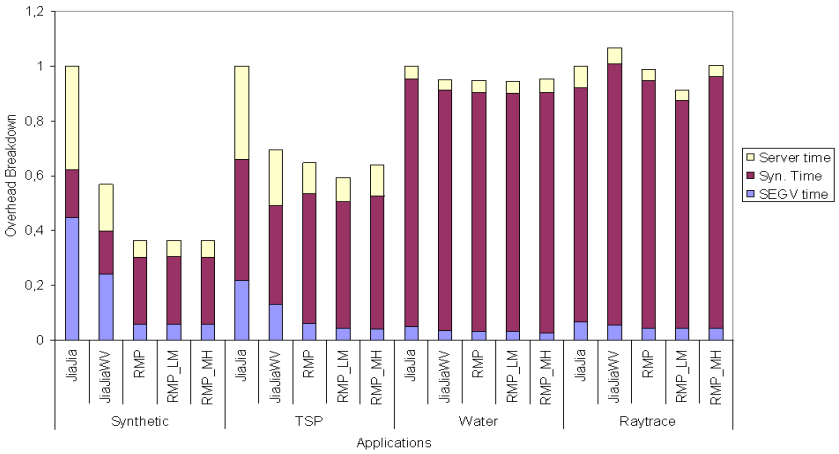


**Fig. 3.** Overhead breakdown

Another performance metric for the comparison of the protocols is their overhead, which consists of the synchronization time, the SEGV time and the server time. The synchronization time is the time spent for barriers, locks and unlocks, the SEGV time is the time spent due to page faults and the server time is the time a processor spends to serve other processors requests(i.e. lock or page requests). The protocol overhead breakdown is shown figure 3.

In general, we can see that the SEGV as well as the server time is reduced, while the synchronization time is increased. This happens because there are less page faults which naturally reduces the SEGV time. Subsequently a processor has less get page requests to serve and the server time is also reduced. On the other hand, at a lock request, a processor receives larger messages, since in the lock grant messages, modifications of some pages are piggybacked and as a result the synchronization time is increased.

If we compare the three new protocols, we see very little variations. Actually, the cases in which one of the variations needs to be taken are few. In table 3 we can see in detail for each application how many pages and how many locks

| Applications | Variable | RMP | RMP-LM | RMP-MH |
|---|---|---|---|---|
| Synthetic | MIpages | 120 | 120 | 120 |
| | MOpages | 120 | 120 | 120 |
| | Mlocks | - | 0 | - |
| TSP | MIpages | 35 | 35 | 36 |
| | MOpages | 35 | 35 | 35 |
| | Mlocks | - | 1 | - |
| Water | MIpages | 7 | 6 | 12 |
| | MOpages | 7 | 6 | 6 |
| | Mlocks | - | 5 | - |
| Raytrace | MIpages | 0 | 0 | 0 |
| | MOpages | 0 | 0 | 0 |
| | Mlocks | - | 0 | - |

**Table 3.** Protocols' migration variables

migrate. MIpages is the number of pages that migrated to a node, while MOpages is the number of pages that migrated from a node. These numbers are not equal in RMP-MH, and for this reason they are given separately. Last, Mlocks is the number of the migrated locks, which of course has a value only in the case of RMP-LM protocol.

By the results, we see that in our synthetic application and in Raytrace the three protocols show the same behavior, since no page includes variables protected by different locks. In TSP only one such page exist, and in RMP-MH this pages obtains two homes, while in RMP-LM one of the locks migrates to the page's home. In Water there are quite a few pages with variables protected by different locks. In RMP-MH six pages obtain two homes, while in RMP-LM, five locks to migrate. In this application we conclude that the best of the three variations is the RMP-LM. It manages to include more pages in the lock operations without any extra burden as in RMP-MH. For this reason, RMP-LM has a larger speedup compared to the JiaJia protocol. Although the benefits of the variations are not quite clear, we believe that their benefits will be greater when applications are run in a different system when nodes are interconnected with a slower network, as in Grids.

## 4 RelatedWork

Since the introduction of Ivy [10], the first Software DSM, many techniques have been proposed to improve SDSM performance. Here we will focus on adaptive techniques between write invalidate, write update, and prefetching techniques.

In [3], the proposed protocol tries to predict in various ways for each lock it's next acquirer(s). At a lock release, diffs of pages modified during the last critical section are sent to the processors that belong to the set of the locks next acquirers.

A dynamic adaptation between write invalidation and write update is described in [6]. Initially, for one page, the protocol switches from write invalidate to write update if the most collaborating processes are in need of that page and the page faults exceeds an experimental threshold. In [4] the pages are categorized in migratory, producer/consumer and falsely-shared. Adaptation is based on the category in which each page belongs. Migratory and producer/consumer pages are managed in a single-writer mode and may be updated, while falsely-shared pages are managed in multiple-writer mode and under invalidated protocol.

Three adaptive techniques are proposed in [5]: adaptation between single and multiple writer, dynamic page aggregation and adaptation between write invalidation and write update. The adaptive protocol between the write invalidate and write update, updates the pages that the processor is expected to access and invalidates the others but there is a limit so that no more than eight pages can be updated. For barrier based applications, each processor p records for a particular page from which processors it has received page requests and sends updates to these processor and invalidates to the others. For lock based applications, the pages that are protected by one lock are recorded and updates are sent for these pages while invalidates for the others.

Finally, in [9] a prefetching technique is proposed, in which the data is invalidated after a repetitive synchronization pattern and is prefetched at proper times.

## 5 Conclusion

In this paper we introduced the RMP cache coherence protocol for SDSM systems. This protocol in some cases adopts write-invalidate and in some cases write-update method. The difference with previous adaptive protocols is that in the new protocol the updates of the pages is done without sending any extra messages, but rather by piggybacking the information in the existing lock grant messages. Since the updates are successful, we achieve sending in general the same amount of bytes but in significantly less number of messages. We must also note that the performance improvements are even bigger because the messages are sent in the beginning of a critical section, while in other previous protocols, such as in JiaJia, there would be many more page faults and consequently, messages sent during critical sections. Apart from page requests, we further reduce messages owed to page diffs, which are piggybacked to lock release messages.

In order to send as many pages as possible in grant lock messages, we associate a lock with the pages modified during a critical section of the lock and if a page is associated with only one lock, it migrates to the lock's home. This happens in all three proposed protocols. However, in order to include pages that contain variables protected by different locks, we implemented two additional variations of RMP. In the first variation, we allowed a page to have more than one homes, and in the second, we permit, if necessary, a lock to migrate. The second variation achieves better results than the first since it has no extra burden like sending modifications of pages to it's multiple homes. On the other hand,

it sends updates of more pages and achieves even less page faults. The significant reduction of messages in all three proposed protocols, results respectively in major reduction in the applications total execution time.

## References


1. W. Shi, PhD thesis, Institute of Computing Technology, Chinese Academy of Sciences, 1999.
2. W. Hu, W. Shi and Z. Tang, *Optimizing Home-based Software DSM Protocols*, Cluster Computing: The Journal of Networks, Software and Applications, Baltzer Science Publishers, 2001.
3. C. B. Seidel, R. Bianchini, and C. L. Amorim, *The Affinity Entry Consistency Protocol*, Proceedings of the 1997 International Conference on Parallel Processing, August 1997.
4. L. Whately, R. Pinto, M. Rangarajan, L. Iftode, R. Bianchini, and C. L. Amorim, *Adaptive Techniques for Home-Based Software DSMs*, Proceedings of the 13th Symposium on Computer Architecture and High-Performance Computing, September 2001.
5. C. Amza, A.L. Cox, S. Dwarkadas, K. Rajamani, and W. Zwaenepoel, *Adaptive Protocols for Software Distributed Shared Memory*, Proceedings of the IEEE, Special Issue on Distributed Shared Memory, vol.87, no.3, pages 467-475, March 1999.
6. M. Ng and W. Wong, *Adaptive Schemes for Home-based DSM Systems*, Proceedings of the 1st Workshop on Software Distributed Shared Memory, pages. 13-20. June 1999
7. L. Iftode, J. P. Singh, and K. Li, *Scope consistency: A bridge between release consistency and entry consistency*, Proceedings of the 8th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'96), pages 277-287, June 1996.
8. H. C. Yun, S. K. Lee, J. Lee, and S. Maeng, *An Efficient Lock Protocol for Home-based Lazy Release Consistency*, Proceedings of Cluster Computing and the Grid, 2001.
9. S. K. Lee, H. C. Yun, J. Lee, and S. Maeng, *Adaptive Prefetching Technique for Shared Virtual Memory*, Proceedings of 3rd International Workshop on Software Distributed Shared Memory System, Brisbane Australia, May 2001.
10. K. Li, *A shared virtual memory system for parallel computing*, Proceedings of the 1988 International Conference on Parallel Processing (ICPP88), pages 94101, 1988.
11. Louis Rilling and Christine Morin, *A Practical Transparent Data Sharing Service for the Grid*, Proceedings Fifth International Workshop on Distributed Shared Memory (DSM 2005), Cardiff, UK, May 2005.
12. G. Antoniu, L. Bouge, and M. Jan, *JuxMem: Weaving together the P2P and DSM paradigms to enable a Grid Datasharing Service*, Kluwer Journal of Supercomputing, 2004.
13. G. Tournabitis, E. Polychronopoulos, *Multithreaded Home-based Lazy Release Consistency for Clusters of SMPs*, Technical Report, HPCLAB-TR-250206, February 2006.
14. A. Christopoulou, E. Polychronopoulos, *A Dynamic Lock Protocol for Scope-Consistency in Software DSM Systems*, Technical Report, HPCLAB-TR-100106, January 2006.