A Comparison Study of the HLRC-DU Protocol versus a HLRC Hardware Assisted Protocol

Salvador Petit, Julio Sahuquillo, and Ana Pont
Departamento de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia, Cno. de Vera s/n 46071, Valencia (Spain)
{spetit, jsahuqui, apont}@disca.upv.es

Abstract

SVM systems are a cheaper and flexible way to implement the shared memory programming paradigm. Their huge flexibility is due to their software implementation; however, this is also the main responsible of their performance drawbacks with respect to hardware systems.

In this paper we compare a pure software HLRC protocol called the HLRC-DU, versus an improved version of the HLRC protocol that uses hardware support to reduce asynchronous communication. Performances of both protocols are compared over a baseline HLRC protocol. Results show that, by on the half of the benchmarks, our protocol performs better than the hardware approach, even more, in some cases our protocol reaches a speedup higher than 22% with respect to the baseline protocol.

Keywords—Shared Virtual Memory Systems, Memory Consistency Models, Memory Consistency Protocols, Asyncrhonous Communication.

1. Introduction

Li and Hudak introduced the SVM system concept [1] and their implementation details [2]. Four main features define an SVM system: i) nodes share a common virtual memory address space, by using the virtual memory system provided by the supporting OS; ii) the page is the sharing unit; iii) the supporting software (OS, libraries, etc.) takes charge of guaranteeing coherence maintenance of the shared pages; and iv) the parallel workload is independent of the interconnection network and the hardware supporting it. These features make SVM systems especially attractive because they allow the use of shared memory code without modifications in heterogeneous and decoupled networks.

SVM systems are usually composed by several inexpensive nodes (single processors or SMP systems) connected by a commodity network, which makes these systems cheaper than other hardware based alternatives. Since nodes are physically independent this approach offers both good flexibility when maintaining and upgrading the processing nodes of the systems.

Although these advantages, SVM systems suffer performance limitations due to their software implementation as well as the parallel workload behavior: *critical section dilation* [3] and *sharing pattern conversion* [4]. By one hand, most current parallel workloads have been optimized to run on distributed hardware systems (e.g., SMP or supercomputers). On the other hand, SVM systems lack hardware support for a lot of tasks supported by these hardware systems. Therefore, SVM systems experience performance losses because they implement these tasks by using software asynchronous communication [5].

Some SVM incorporate hardware mechanisms to mitigate performances losses due to asynchronous communication. In this paper we propose a new pure software SVM multiple-writer protocol which achieves better performance than a protocol using hardware mechanism to reduce asynchronous communication.

The remainder of this paper is organized as follows. Section 2 discusses the background regarding workload studies aimed to SVM systems and asynchronous communication. Section 3 explains the performance problems that current SVM systems suffer. Section 4 discusses the motivation of this work. Section 5 presents the simulation environment as well as the baseline and HLRC-DU protocols. Section 6 details step by step the design process we followed till obtain a protocol which achieves better performance that some recent protocols proposed in the open literature. Finally, section 7 presents some concluding remarks.

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

2. Background

Previous research has explored the behavior of parallel workloads running on SVM systems [4][3][6]. Iftode *et al.* [4] established a workload taxonomy based on sharing patterns and the granularity of sharing. Jiang *et al.* [3] modified source code based on the described axes to improve the performance of SVM systems. Zhou *et al.* [6] studied the behavior of workloads running on several protocols and systems, establishing a number of rules about the optimal granularity of sharing. In their workload taxonomy, they also studied the frequency of synchronization.

Recent research also has been addressed to reduce the impact of asynchronous communication in SVM systems, which is a key factor in improving their performance and one of the most problematic points in their design. Some mechanisms include hardware support that partially, or totally, avoids this kind of communication [7][5][8][9]. Others try to reduce this communication or hide its latency using software techniques [11][10][12].

Software techniques can avoid asynchronous data requests by updating the correspondent data. The Quarks system [12] uses a pure update protocol, while other systems use a hybrid protocol, which updates when certain conditions occur. The Brazos system [10] uses multicast to update other nodes in the copyset of the page if they have a data request for the same page, as well as to update predicted clients before they leave the barriers. Stets *et al.* [9] measure the performance of a multicast protocol based on a history record.

Hardware techniques can update data like software techniques as well as serve data requests automatically without processor intervention [7][5]. In [7], a hardware support that serves data requests is proposed, but the processor is already interrupted to perform protocol tasks. In [5] the NI processor of the Myrinet is used to serve pages automatically.

3. Asynchronous Communication

Basically, in all asynchronous communication, a *client* node initiates a request and a *server* node services the request. For example, the client node asynchronously communicates with the server node to read a given page or to lock a given semaphore. This communication involves a context switch in the server node, which implies high service latencies and wastes precious computing time in the server [7].

In SVM systems, the asynchronous communication that mainly impacts on performance is produced by data requests. When a client node tries to access an invalid page, it starts an asynchronous communication with the server to fetch the data. The server answers by submitting the whole page.

Two main effects increase asynchronous communication: *critical section dilation* and *sharing pattern conversion*. These effects are strongly related with the workload behaviour. Therefore, to discuss them we need to introduce first three workload characterization metrics: the *frequency of sharing*, the *granularity of sharing*, and the *sharing pattern*.

Frequency of Sharing: Coherence actions in SVM systems are carried out at synchronization points; therefore, the frequency of the synchronization operations matches the frequency of sharing. The frequency of sharing metric is calculated as the average computation time between synchronization events [6]. We assume there is *fine-grained* synchronization (FGS) if the average computation time is close to the average synchronization time. Otherwise, the synchronization is *coarse-grained* (CGS).

Granularity of Sharing: This characteristic quantifies the mean amount of data transferred when an update occurs. It is computed with regard to the granularity of the system (i.e., the page size, which is typically 4 or 8 Kbytes). The granularity of sharing is classified as *fine-grained* (FG) when only a few words (less than 30%) of the page are shared, *medium-grained* (MG) when at least 30% of the page is shared, and *coarse-grained* (CG) if more than 60% of the page is shared. The granularity of sharing can be further broken down depending on the type of memory operation performed on the shared data (i.e., *granularity of reading* and *granularity of writing).* Both granularities are commonly present in different sizes.

Sharing Pattern: During workload execution data sharing follows a given pattern. This sharing pattern can be stable throughout the workload execution or can dynamically change. According to the number of producers and consumers of data, the sharing pattern for a given instance of data can be classified in one of four categories: i) *1P-1C,* there is only one producer (P) and one consumer (C). This

category includes the case known as migratory sharing, where the consumer becomes the producer of the same data in the future; ii) *1P-MC*, there is just one producer and multiple (M) consumers; iii) *MP-1C*, there are multiple producers and only one consumer; and iv) *MP-MC*, there are both multiple producers and consumers. In addition, we consider the patterns *0P-1C* and *0P-MC*, which refer to one and multiple consumers of the first-loaded data, respectively.

Now we are going discuss the critical section dilation and sharing pattern conversion effects, from the last three metrics point of view.

3.1. Critical Section Dilation

Hardware systems usually support FGS synchronization and workloads are optimized to minimize the communication to computation ratio. In systems supporting this granularity of synchronization, the cost of synchronization events is small with respect to a SVM system, which allows short critical sections to be frequently executed. Critical sections are used to protect shared data or to implement shared task queues.

The time that parallel workloads spend in critical sections increases when running in SVM systems because of two main reasons. By one hand, SVM systems do not support FGS synchronization; thus, the synchronization primitives such as locks or barriers are mapped to a distributed set of software queues. On the other hand, some SVM systems carry out invalidations at synchronization points; thus, increasing the probability that a page fault occurs while executing the critical section. Both situations introduce latency due to software message passing and asynchronous communication with other nodes. In addition, the software management of the SVM protocol adds more latency.

The sum of the mentioned latencies implies that the total time that workloads spend in the critical sections is much higher in SVM systems than in hardware systems. Since critical sections are frequently executed, the contention increases, which also results in lower performance. This effect is so important than some sections of code, which represent a very small percentage of the total execution time in hardware systems, may become performance bottlenecks in SVM systems.

3.2. Sharing Pattern Conversion

The execution of each parallel workload follows a given pattern we will refer to as the *inherent sharing pattern*. This pattern can be static or can change dynamically throughout workload execution. Since data instances are shared at a given granularity, this granule size can be carefully optimized to map efficiently onto specific hardware systems. We will refer to this granularity as *workload granularity*, while we will refer to the sharing unit granularity supported by the system as *system granularity.* In general, the workload granule size is small (less than 64 bytes, FG) but it can change with the problem size, while the granularity of the SVM system is usually CG.

When the workload granularity is smaller than the system granularity, it is probable that a new *induced sharing pattern* appears. The chance of this new pattern arise depends upon the characteristics of the workload and is a function of the disparity between granularity sizes. There are two main effects that can produce sharing pattern conversion: *false sharing* and *fragmentation*. Both can appear simultaneously.

False sharing appears when the *write* granularity of the workload is smaller than the system granularity. In this case, the producer only writes a fraction of the whole sharing unit, so several producers could write data to the same sharing unit. Thus, the inherent sharing pattern with one producer (1P) becomes an induced sharing pattern with multiple producers (MP).

Fragmentation appears when the *read* granularity of the workload is smaller than the system granularity. In this case, the consumer only reads a fraction of the whole sharing unit, so several consumers could read data from the same sharing unit. Thus, the inherent sharing pattern with one consumer (1C) becomes an induced sharing pattern with multiple consumers (MC).

4. Motivation

In a previous work [13], we performed a characterization study of the discussed phenomena on several parallel workloads. We concluded that i) asynchronous communication occurs in bursts, producing critical section dilation; ii) due to the large granularity supported by SVM systems sharing pattern transformation appears, considerably increasing the amount of asynchronous page

requests, which is one metric directly related with performance losses. Therefore, one can deduce that performance will improve by reducing such amount.

From this characterization study, we proposed the HLRC Diff Update (HLRC-DU) protocol [14], which is an improvement over the Home Lazy Release Consistency (HLRC) protocol [15]. The HLRC-DU protocol updates selectively because the key to achieve good performance is to update data without increasing excessively the network traffic.

In this work we will compare the HLRC-DU protocol, which is a pure software protocol with a hardware technique to avoid asynchronous communication from the recent literature [5]. Next sections will describe how our protocol works and study the performance of it with respect to a baseline and the hardware technique.

5. Simulation Environment

To check the impact on performance of write updates we use the LIDE execution driven simulator [16]. The simulator uses a modified version of the GCC v2.6.3 compiler using the -O2 optimization flag. Table 1 lists the problem size used for each benchmark. Every benchmark was executed considering 16 processes.

Table 1. Benchmark problem sizes

Benchmark	Problem Size
Barnes	2K particles
FFT	256K points
LU	512x512 points
LU-CONT	512x512 points
Ocean	130x130 ocean
Radix	1M integers
Water-NSQ	512 molecules
Water SP	512 molecules

The modeled system consists of a single cluster composed of 16 nodes connected through an overclocked 1 Gb/s Ethernet network. The network contention is also modeled. Each node contains a single 1GHz processor.

The load in each node includes both the parallel application plus the operating system overhead introduced by the memory consistency model. Each node has two-cache levels: a direct mapped 64KB L1 cache and a direct mapped 1 MB L2 cache. The latencies of both caches were assumed 1 and 8 cycles, respectively, while when both caches miss (access to main memory) the latency is 20 cycles long. When a page fault or a remote request occurs, the operating system takes 100 ?sec to change the context. Before returning to the parallel application, the system checks if there is any request pending from a remote processor. If so, the system attends those requests, and each one takes 10 µsec.

In SVM systems, updates are performed using *diffs*, as described in [17]. Diff creation and application time grows linearly with the page size (4 cycles per word). As page size is assumed to be 4KB and word size 4B, all protocols take 4096 nanoseconds in either creating or applying the diff. This overhead is not present when copying a single page because the model assumes that a DMA device performs this task.

5.1. The Baseline HLRC Protocol

In the HLRC protocol, each page has associated a *home* node that concentrates all the diffs. For this purpose, when a node writes in a page it supplies the diffs only to the home node. Once supplied, diffs are removed from the writing node. The remaining nodes invalidate the written pages when receive the write notices associated to the diffs. When a node has a page fault the OS asks the home node for an updated page. Due to network delays, the needed diffs may have not already arrived at the home node. In this case, the request is queued until the diffs arrive.

Figure 1 shows how modifications of the writer node (node *A*) on given page arrive at the home node of the page (node *B*) before the invalidated node (node *C*) asks for an up-to-date copy from the home node (node *B*).

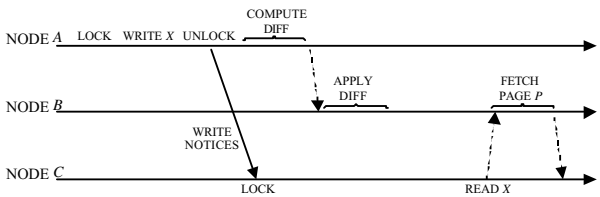

Figure 1. HLRC protocol example

Of course, for a given page, processes running in the home node never have a page fault. Therefore if the home is properly chosen (e.g., by profiling), asynchronous communication could be reduced. In addition, bandwidth consumption is also reduced because writers only update the home. For this

reason, HLRC protocols are the most used in SVM system implementations.

In our implementation, the page homes are selected by means of a module function of the most significant bits of their page addresses. Each write notice contains the identification of the writer process, the timestamp of the write, and the page address. Write notices are only sent to a given process whether it acquires a semaphore or to all processes when they reach a barrier. Once a process releasing a semaphore or a barrier sends the write notices to the acquirer, it immediately sends to the homes those diffs produced by its previous writes in order to keep the homes updated.

As the page homes, each semaphore and barrier has a home node selected by using a module function. The semaphore home node queues the acquire requests and remembers which was the last node releasing the semaphore. This allows it to forward those requests to the last releaser when needed. Then, the releaser will directly send the write notices to the acquirer node without crossing the home.

Nodes that reach a barrier send the write notices to the barrier home and get blocked. When the home has received all the barrier requests it sends to all pending processes the write notices it has received. Finally, nodes invalidate the corresponding pages and release the barrier. Barriers are implemented without using the broadcast capabilities of Ethernet.

5.2. The HLRC-DU Protocol

In contrast with the baseline HLRC the HLRC-DU protocol works as a hybrid protocol that decides which diffs will be updated and which not. The protocol submits those diffs to be updated attached to the write notices. All this information is referred as write updates. When diffs are not updated the protocol proceeds as the baseline submitting only write notices and invalidating the page. If the updates are effective enough the protocol reduces considerably the amount of page faults; i.e., it reduces the amount of asynchronous page requests to the homes.

Figure 2 shows a working example under the same scenario than the baseline protocol. The example shows how modifications of the writer (node *A*) on given page arrive directly to node *C* when entering the semaphore as well as to the home node (node *B*). Later accesses to the page of node *C* (*READ X*) will not result in a page fault so that it will not interrupt to node *C* for an up-to-date page as occurred in the baseline protocol.

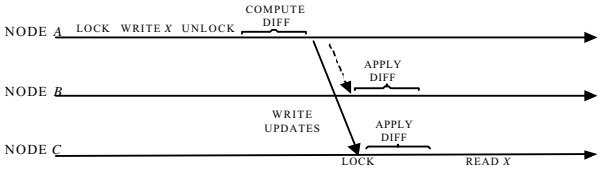

Figure 2. HLRC-DU protocol example

If during the acquisition of a semaphore, a process receives both an update and invalidation for the same page, the page becomes invalid. Therefore, if the node requires the page it proceeds as the baseline protocol. Thus, there is a need to update the home (node *B*) as occurs in the baseline HLRC protocol.

The main decision the protocol does is to select which diffs should be updated. This is a critical decision since update techniques introduces a tradeoff between the benefits on performance that involves the reduction of asynchronous communication and the adverse effects due to the increase of network utilization. We use a threshold value that acts as a mechanism that can open or close the traffic injected in the network due to write updates. The HLRC-DU protocol detects diffs smaller than this threshold size and updates them via write updates. Note that as the threshold approaches zero, the protocol works close to the baseline HLRC; whereas under no threshold restrictions it behaves like a pure update protocol.

6. Experimental Results

To reach as good performance as possible, the HLRC-DU protocol should select to update those diffs having two main restrictions, 1) their overall volume (size times frequency) should be small enough to fit in the network bandwidth without introducing network congestion, 2) their update should be useful enough to avoid part of the subsequent asynchronous communication. This section firstly analyzes the characteristics of the running workload and how they accomplish both restrictions before doing performance evaluation.

6.1. Protocol Sensitiveness

To check the first restriction we measure the diff

size distribution per processor produced by the write notice when they are sent. Looking at Table 2 one can see that most diffs are relatively small since, on the average, by about the 58% contains less than 128 words. Thus, in principle, one could think that this value would be a good value; however, as showed later this would saturate the network in many cases. Note that what the algorithm pursues by using write updates is to avoid page requests to the home node. In this sense, Table 2 represents the maximum number of requests that can be avoided.

Table 2. Distribution of diff sizes

Diff Size Range (Words)	Benchmark								Cumulative percentage
	Barnes	FFT	LU	LU-CONT	Ocean	Radix	Water-NSQ	Water-SP	
]0,1]	3366	1	0	0	12	7	0	3	2%
]1,2]	93	0	0	0	665	7	161	164	3%
]2,4]	14525	0	0	0	21	21	74	3	14%
]4,8]	839	0	32	32	1608	38	95	16	16%
]8,16]	746	0	0	0	1128	99	148	256	18%
]16,32]	1322	0	9215	0	1016	139	9065	2	33%
]32,64]	261	0	0	0	4570	10869	77	560	45%
]64,128]	647	0	5600	0	3345	7886	28	17	58%
]128,256]	303	0	23488	0	3689	2037	70	78	79%
]256,512]	40	0	3002	1585	2166	1373	36	980	86%
]512,1024]	0	3120	0	4912	8760	113	704	0	99%

The threshold value imposes a trade-off between bus utilization and the reduction of asynchronous communication. This means that both restrictions are strongly dependent on the threshold value. Therefore, to explore both restrictions, we use six different threshold values ranging from 16 to 512 words, besides of no threshold restrictions.

Table 3. Percentage of saved home page requests varying the threshold size

Benchmark	Requests	Threshold size (words)						
		16	32	64	128	256	512	?
Barnes	11539	70%	76%	76%	78%	85%	87%	87%
FFT	26616	0%	0%	0%	0%	0%	0%	30%
LU	39507	0%	0%	2%	14%	69%	82%	82%
LU-CONT	2880	0%	0%	0%	0%	0%	0%	24%
Ocean	35231	7%	7%	9%	16%	32%	45%	90%
Radix	22944	0%	0%	0%	6%	6%	6%	7%
Water-NSQ	4083	5%	40%	41%	41%	42%	44%	74%
Water-SP	7266	6%	6%	13%	13%	13%	72%	72%

Table 3 shows the results, that is, the percentage of home page requests saved by write updates by varying the threshold value. Results from the last column are lower than 100% due to the initial home page requests.

In general, larger write updates avoid larger amount of page requests among the benchmarks. This is because larger diffs present less false sharing, i.e. they leave fewer places in the page that could be invalidated. Although larger diffs hugely reduce the number of page requests, the network traffic will increase too. Consequently, we must trade off this reduction with network traffic (bus utilization) to improve the system performance.

6.2. Performance Results

Figure 3 shows the network utilization in the baseline HLRC model for each benchmark used while varying the threshold size. While in most cases, under a large threshold value, the injection of write updates considerably increases the network traffic; e.g., FFT, LU, LU-CONT, Ocean and Radix, it is remarkable that no benchmark increases the network traffic with respect to the baseline protocol when using a small threshold value.

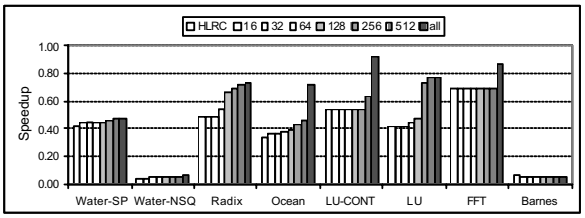

Figure 3. Network utilization varying the threshold size

It is unclear from Figure 3 when the network becomes a performance bottleneck. To check the impact of network utilization on performance we measure the speedup of the proposed protocol over the baseline one, varying the threshold size.

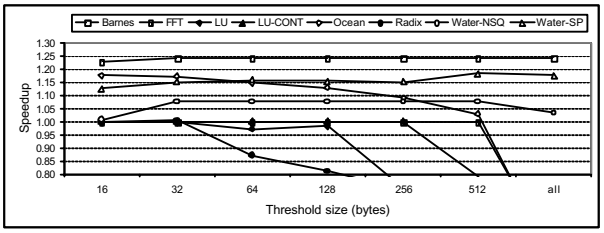

Figure 4. Speedup relative to the baseline protocol varying the threshold size

Figure 4 shows the results. As one can see, network becomes a bottleneck for utilization higher than 50%. For example, Ocean with no threshold restrictions, and Radix for threshold values higher than 64B. Note that in the case of FFT, the network already was a bottleneck for the baseline protocol. For lower utilization values the speedup mainly depends on the percentage of asynchronous communication saved by the HLRC-DU protocol.

Experiments show that the threshold values that achieve the best speedup depend on the workload used but, in general, they range from 16 to 64 words.

6.3. Performance versus Hardware Techniques

In this section we compare the HLRC-DU performance versus a relevant hardware approach found in the literature [5].

In general, hardware techniques for improving performances of HLRC protocols use specific hardware, or dedicated processors, for avoiding asynchronous communication at the node serving the page. In this way, pages are served automatically and the home node is uninterrupted. We modeled this feature in the simulator by assuming that in these kinds of systems the page is served in zero time. Figure 5 presents the speedup of HLRC-DU using different threshold sizes (16B, 32B, and 64B) and the baseline protocol with the hardware that automatically server pages without asynchronously interrupting the processor.

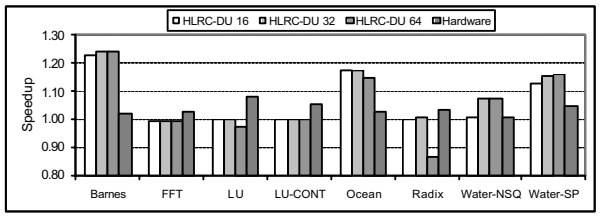

Figure 5. Speedup of the HLRC-DU protocol using a threshold size of 32 words relative to the baseline protocol

Results show that specific hardware performs better than HLRC-DU only in those cases where HLRC-DU does not obtain benefits relative to the baseline protocol. In all other cases, HLRC-DU performs better than hardware. This occurs because write updates save two interrupts, one at the node accessing the page and the other at the home node. In contrast, the hardware only saves one interrupt at the home node. Although pages are served in zero time, in four of the eight workloads considered (Barnes, Ocean, Water-NSQ and Water-SP) the additional saved interrupt improves the performance.

Note that the compared hardware and the HLRC-DU protocol are compatible. In fact, they are complementary approaches. Table 4 shows the results of combining both approaches. In some workloads like Barnes, Ocean and Water-SP, the HLRC-DU protocol and hardware sum up performances. In others, the advantage of just one approach remains. There are only two workloads where HLRC-DU impacts negatively together with the hardware (FFT and LU-CONT), but in those two cases the negative impact is less than 2%.

Table 4. Speedup relative to the baseline protocol

Benchmark	HLRC-DU 32	Hardware	HLRC-DU 32 + Hardware
Barnes	1.24	1.02	1.25
FFT	1.00	1.03	1.02
LU	1.00	1.08	1.08
LU-CONT	1.00	1.06	1.04
Ocean	1.17	1.03	1.21
Radix	1.01	1.04	1.04
Water-NSQ	1.08	1.01	1.08
Water-SP	1.15	1.05	1.19

7. Conclusions

The overhead associated with the software management of SVM systems introduces extra latencies that can adversely impact on system performance. One way to mitigate this overhead is to design more efficient SVM consistency protocols.

A previous characterization work [13] induced us to design the HLRC-DU protocol, which is based on the HLRC protocol. In the proposed protocol the writer node sends write updates instead of write notices when it detects diffs smaller than an experimental threshold, which is used to avoid the network becomes a bottleneck as well as to reduce asynchronous communication.

The percentage of home page requests our protocol saves is really high, whatever the threshold used. When considering the optimal threshold (32B is the one which achieves the best performance), results show that in some cases, like Barnes and

Water-NSQ, this percentage surpasses the 40%. This reduction is the reason why our protocol, in some cases, reaches a speedup even higher than the 20% over the HLRC baseline protocol.

We also compare the HLRC-DU to a hardware approach found in the literature [5]. Results show that, by on the half of the benchmarks, our protocol performs better than the hardware approach. To remark that performance results (speedup) are much closer when the hardware reaches better performance. We also modelled a variant of our protocol including the hardware mechanism (since they are complementary). We found that, on the average, this variant performs by about 11% better than the baseline protocol.

Acknowledgments

This work has been partially supported by the Generalitat Valenciana under grant GV04B/487.

References

[1] K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *Proceedings of the 5th Annual Symposium on Principles of Distributed Computing*, August 1986.

[2] K. Li, "IVY: A Shared Virtual Memory System for Parallel Computing," *Proceedings of the 1988 International Conference on Parallel Processing*, August 1988.

[3] D. Jiang, H. Shan, and J. P. Singh, "Application Restructuring and Performance Portability across Shared Virtual Memory and Hardware-Coherent Multiprocessors," *Proceedings of the 6th Symposium on Principles and Practice of Parallel Programming*, June 1997.

[4] L. Iftode, J. P. Singh, and K. Li, "Understanding Application Performance on Shared Virtual Memory," *Proceedings of the 23rd Annual Symposium on Computer Architecture*, May 1996.

[5] A. Bilas, C. Liao, and J. P. Singh, "Using Network Interface Support to Avoid Asynchronous Protocol Processing in Shared Virtual Memory Systems," *Proceedings of the 26th Annual International Symposium on Computer Architecture*, May 1999.

[6] Y. Zhou, L. Iftode, J. P. Singh, K. Li, B. R. Toonen, L. Schoinas, M. D. Hill, and D. A. Wood, "Relaxed Consistency and Coherence Granurality in DSM Systems: A Performance Evaluation," *Proceedings of 6th Symposium on Principles and Practice of Parallel Programming*, June 1997.

[7] R. Bianchini, L. I. Kontothanassis, R. Pinto, M. De Maria, M. Abud, and C. L. Amorim, "Hiding Communication Latency and Coherence Overhead in Software DSMs," *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[8] M. A. Blumrich, R. D. Alpert, A. Bilas, Y. Chen, D. W. Clark, S. Damianakis, C. Dubnicki, E. W. Felten, L. Iftode, K. Li, M. Martonosi, and R. A. Shillner, "Design Choices in the SHRIMP System: An Empirical Study," in *Proceedings of the 25th Annual Symposium on Computer Architecture*, June 1998.

[9] R. Stets, S. Dwarkadas, L. Komothanassis, U. Rencuzogullari, and M. L. Scott, "The Effect of Network Total Order, Broadcast and Remote-Write Capability on Network-Based Shared Memory Computing," *Proceedings of the 6th Symposium on High-Performance Computer Architecture*, January 2000.

[10] E. Speight and J. Bennett, "Using Multicast and Multithreading to Reduce Communication in Software DSM Systems," *Proceedings of the 4th Symposium on High-Performance Computer Architecture*", February 1998.

[11] A. Bilas and J. P. Singh, "The Effects of Communication Parameters on End Performance of Shared Virtual Memory Clusters," *Proceedings of the Supercomputing '97 Conference*, November 1997.

[12] A. M. Swanson, L. Stoller, and J.B. Carter, "Making Distributed Shared Memory Simple, Yet Efficient," *Proceedings of the 3rd International Workshop on High-Level Parallel Programming Models and Supportive Environments*, March 1998.

[13] S. Petit, J. Sahuquillo, A. Pont, and D. Kaeli, "Characterization the Dynamic Behavior of Workload Execution in SVM Systems," *Proceedings of the 16th Ssymposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, October 2004.

[14] S. Petit, J. Sahuquillo, and A. Pont, "About the Sensitivity of the HLRC-DU Protocol to the Written Area Size and Page Size," *Proceedings of the 2001 IEEE International Symposium on Performance Analysis of Systems and Software*, November 2001.

[15] Y. Zhou, L. Iftode, and K. Li, "Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Virtual Memory Systems," *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation*, October 1996.

[16] S. Petit, J. A. Gil, J. Sahuquillo, and A. Pont, LIDE: A Simulation Environment for Shared Virtual Memory Systems, September 2000 issue of the ACM Computer News, Vol. 28, No. 4.

[17] S. Petit, Efficient Home-Based Protocols for Reducing Asynchronous Communication in Shared Virtual Memory Systems, Ph.D. Thesis, Universidad Politécnica de Valencia, February 2003.