# Memory Utilization in Software DSM for Embedded Systems

Nguyen-Thai Nguyen-Phan[1] and Mats Brorsson
Department of Microelectronics and Information Technology.
Royal Institution of Technology
Electrum 229, SE-164 40 Kista, Sweden
Email: thai@imit.kth.se, Mats.Brorsson@imit.kth.se
Fax: +46-8-751 1793

**Abstract:** Software Distributed Shared Memory (S-DSM) systems support parallel programming by implementing a shared memory on top of distributed system. It frees programmer from communication complexities to concentrate to parallel algorithms. However, there is a drawback: current S-DSM systems waist memory on all processors. Therefore it is hard to be implemented to embedded systems, which have small resources e.g. memory. In this project, we modified an implemented S-DSM system named CVM – Coherent Virtual Machine – to support an embedded system and investigate memory consumed, in order to implement an S-DSM system supports embedded systems. The results shows that with a good selection of initial values for an S-DSM system and carefully programmed, S-DSM system can be implemented for embedded system with less memory overhead and high speedup.

**Keywords:** Software distributed shared memory, S-DSM, embedded system, parallel programming**.**

## I. INTRODUCTION

Even though processors for embedded computer systems are becoming more and more powerful there is still a significant performance gap to PC processors and the like [11]. At the same time, we ask that the embedded systems can perform more performance demanding tasks. One prominent example is image processing. In network attached cameras used for surveillance systems—such as the ones produced by Axis Communications [2]—it would save an enormous amount of network bandwidth if intelligent processing, for instance face recognition, could be done in the embedded systems serving the cameras and then only send the important data, e.g. the identity of the person recognized, over the network.

In order to achieve this we need more computational power than normally is present in embedded systems. However, we do not want to put more computational power in the system than needed and it is also desirable that increased performance can be added incrementally and in a modular fashion.

One relatively simple way to increase performance is to use parallelism. Several identical embedded platforms can be joined together to form nodes in a parallel platform that can be programmed as one parallel computer with distributed memory (DM). This has the advantage that we do not need to modify the existing platform to make use of parallelism. The interconnection medium between the individual nodes can be the standard network or preferably a dedicated high-performance network for inter-node communication.

A DM parallel platform is normally programmed with a message-passing programming mode such as MPI [3]. However, it is widely recognized that a shared address space programming model is to prefer since they present a more natural model to the programmer. Unfortunately, these models normally require substantial hardware support. In order to build cost-effective platforms for shared memory programming, S-DSM systems have been developed, e.g. TreadMarks [1] and CVM [7]. In short, an S-DSM system provides the application program with the illusion of a shared memory on a collection of nodes with distributed memory.

In this paper, we describe the adaptation of a well-known S-DSM system, CVM [7], to be used in an embedded computer system environment based on platforms from Axis Communications, and an analysis of the memory usage for a few parallel applications. As far as we are aware, this is the first time a S-DSM system has been used in embedded platforms where the available memory is scarce and it is therefore important to study the memory usage in particular.

Our conclusion of this work is that it is indeed possible to adapt CVM, which was developed for desktop and server type computers, to be used also in embedded systems. The memory usage analysis also shows that it is important to consider where and how much memory is allocated. For instance, through careful selection of the home location for some memory pages we are able to lower the total memory requirement leading to larger data sets for the applications.

In the rest of the paper we first present some background information about S-DSM systems in section II. Section III describes a version of S-DSM system and memory usage in initialization stage. In section IV, results from applying our system in the Axis developer boards are demonstrated and analyzed. Finally, our conclusion is given in the last section.

[1] Presenting author, if accepted

## II. Software Distributed Shared Memory

### A. Overview

A software distributed shared memory (S-DSM) system provides the illusion of coherent memory sharing for machines connected only by a message-passing network, or by a non-cache-coherent memory system. Traditional S-DSM system provides a conceptually appealing programming model for processes that have been spread across a locally distributed cluster for the purpose of parallel speed-up.

The idea of implementing an S-DSM system is to implement a shared memory layer on top of a message-passing layer. Many memory consistency protocols have been introduced and implemented to achieve this. It has been shown that the home-based lazy release consistency (HLRC) protocol is robust and that it provides good performance for many different types of applications [9]. Therefore, we have chosen HLRC as the memory consistency protocol for our system. In HLRC, as in most other S-DSM protocols, the shared memory is divided into pages with fixed size. However, specific to HLRC is that each page is assigned with a processor called *home* for that page. Memory synchronization takes place at barriers or locks, thus reducing communication and ping-pong effect. At a synchronization point, modifications of shared memory pages since the last synchronization are stored in a data structure, called *diff*, and are sent to the memory in the home node.

### B. Shared address space protocols

In most S-DSM systems that run on Linux OS, accesses to the shared memory are handled by catching a page fault (SEGV) generated when a memory access is made to a page, which is read or write protected.

As introduced before, HLRC was selected as a protocol for our system. We have chosen to base our system on CVM, a flexible and well-documented S-DSM system supporting HLRC [7]. While the protocol is well described in publications [9], in this part we only concentrate on memory consumed in the protocol it self.

In HLRC, shared memory is divided into chunks of pages and each page has a home in one of the processor nodes. The home node keeps track of changes in the content of the page and a list of other nodes, which has a copy of that page. Each non-home node must ask the home node before accessing the page. Since all processors may request any page of the shared memory for reading or writing, they should keep list of properties of all pages, e.g. home address, its current version etc.

When a node—which is not the home—performs the first read access, it will experience a page fault (SEGV) and the S-DSM system will request the page, through messages, from the home. When a write fault occurs, if the node does not already have a copy of the page, it sends a request to the home for a copy of the page. Then it creates another copy of the page, called *twin*, in order to keep track of the modification it will make to the page that later will be communicated to the home node. While the memory for the twin is dynamically allocated, the system also maintains a heap for holding *write notices* and *diffs*. A write notice is a message from the home with information about which pages have been modified and when. This provides information to the HLRC protocol about when it needs to request new information from the home node. A diff is an encoding of the modifications made to a page by that node created from the modified page and the twin.

Because of the different nature of these data structures, the heap is in CVM separated into two heaps: the home-heap for holding write notices and the diff-heap for creating diffs.

The execution of an application is divided into *intervals*. An interval starts at the acquisition of a lock and finishes at the releasing point. A barrier synchronization is from a memory consistency point of view semantically equivalent to a release immediately followed by an acquire-operation. HLRC is a multiple-writer protocol, which means that several nodes may modify the same page simultaneously in order to avoid performance problems. Therefore, new intervals only start at a barrier or a lock request. Barriers and locks are thus the only two synchronization points in this protocol. While barriers synchronize all modified pages to their home, locks only synchronize pages with the owner of the lock at that time. Therefore, like pages, locks also need memory for the lock manager and the next node requesting the lock.

## III. Adapting an S-DSM for embedded systems

For this study, we have built a small cluster system with 4 Etrax developer boards, connected via 100 Mbits/s Ethernet. The memory system for the Etrax board consists of 2 Mbytes Flash ROM and 8 Mbytes RAM. The OS (Linux) is stored in compressed format and saved on the Flash ROM and is uncompressed and installed in RAM during the boot process. When running, a maximum of 3 MByte of RAM is left available for applications on each board. We used the Linux 2.4.5 kernel version and gcc-cris[2] compiler version 2.96.

Etrax itself is a custom-made processor used for communication-oriented applications. The version we are using is clocked with 100 MHz and has a unified 8 KBytes cache memory.

CVM is a research tool for S-DSM systems and contains many different consistency protocol variations. In order to adapt it to an embedded environment we first removed all different protocols but HLRC, modified architecture dependent points to be suitable for the Etrax architecture. By doing that, the CVM library size was reduced from 2 MByte to 0.2 MByte. We also implemented a very simple rsh server and client since these services are not present in the developer board version of Linux. Additionally, we inserted more tracing information in order to track memory usage in applications. The result is a lightweight S-DSM system for the Etrax developer boards that we call *eCVM* for *embedded* CVM.

[2] gcc-cris is a port of gcc for the Etrax processor.

### A. Application run-time phases

The runtime of a parallel application in this system can be divided into 3 phases: *initialize*, *run*, *finalize*.

At beginning, an execution command is sent to the OS in the node 0, the master node. The OS loads the executable file to main memory, initializes the environment and points the program counter to the starting address. Then the program starts executing and goes to initialize state. This very first process is called *parent*. The parent executes appropriate instructions to invoke the same application on other machines. After this point, all processors go to the initialize state; execute exactly the same instructions, like the parent. Therefore, they are called parent simulation.

In this step, the shared address space is created using virtual memory mapping; the memory fault handler is established; the communication mechanism is generated. Then, shared memory variables location is located and initialized. Since all processors are running exactly the same code, and eCVM is running in a homogeneous system, all parallel processes have the same status, including the memory variables' virtual addresses and values, except for information on how to recognize the process itself in the system. Finally, connections to others processes are established.

After the initialization, all processes go to running phase and really execute in parallel. Memory consistency is automatically handled by the system.

In the finalize phase, statistics information is collected; memory is released and the application is finished. However, in CVM, and most of the other system, the memory releasing is skipped. The job is passed to OS do the garbage collection.

### B. The effect of system parameters on memory usage

There are four parameters specified in an S-DSM system—using the HLRC protocol—that affect the memory resource. They are:

- The maximum number of shared memory pages that the system can manage: MAX_PAGES.
- The number of pages which the system pre-allocates to use each time: NUM_PAGES
- The maximum number of processors the system can use: MAX_PROCS
- The maximum number of locks to be used: MAX_LOCKS

We used two different sets of parameter to investigate the memory usage according to Table 1.

**Table 1. Different sets of system parameters used.**

| | Set 1 | Set 2 |
|---|---|---|
| MAX_PAGES | 8192 | 512 |
| NUM_PAGES | 200 | 200 |
| MAX_PROCS | 32 | 32 |
| MAX_LOCKS | 4110 | 500 |

The first set takes common parameters using in standard CVM system. The second set was taken considering the maximum free memory that can be used in our system.

As shown in Table 2, CVM takes less than 1MB of memory for management purpose, which is very small for workstations because they have large memory and virtual memory. However, in embedded system like ours, it could be over 10% of the total memory resource and is about 30% of maximum free memory for an application.

**Table 2. Memory needed at initialization of the system.**

| Memory for | Size (KBytes) | | Affected by parameters |
|---|---|---|---|
| | Set 1 | Set 2 | |
| Page management | 49.6 | 49.6 | NUM_PAGES |
| Page copy sets | 32.0 | 2.0 | MAX_PAGES |
| Home management | 80.3 | 12.8 | MAX_PROCS, MAX_PAGES |
| Locks | 722.5 | 87.9 | MAX_LOCKS |
| **Total** | **884.4** | **152.3** | |

We observed that the maximum problem size decreases when the number of processors increases (see Table 3) and it is independent on number of iterations in iterative algorithms. A reason for this is that the main process, process id 0, uses a system command to invoke a shell to execute rsh. It thus allocates memory to initialize the environment and does not return it when done. Besides that, resources for holding connections and preparing the exchange of messages increase when the number of nodes increases.

**Table 3. Maximum problem size vs. number of processors for two different versions of SOR.**

| Number of nodes | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| SOR | 590x590 | 540x540 | 520x520 | 520x520 |
| ID_SOR[3] | 590x590 | 775x775 | 885x885 | 1000x1000 |

## IV. APPLICATION STUDIES

### A. Speedup

Lacking good parallel applications for high-performance embedded systems, we have used five applications from the CVM distribution to evaluate our new system [7]. They are SOR, Water, FFT, QS and TSP. The programs originally come from the SPLASH benchmark suite and have been described in detail elsewhere so we do not explain the algorithms here [6].

Because of the limited memory resource on the developer boards, the applications can only run with relatively small data sets as given by Table 4. Most of applications got speedup of about 3.5 or higher with four processors. The performance speedup will be discussed in more detail in the next section.

[3] ID_SOR is a modified version of SOR, in which the shared memory is initialized by it' home only.

**Table 4. Shared memory usage and execution time of applications**

| Application | Shared memory used (pages) | Execution time for 1 processor (seconds) |
|---|---|---|
| SOR (500x500x5) | 247 | 82.5 |
| Water (125 molecules) | 11 | 1245.1 |
| FFT (16x16x16) | 26 | 49.5 |
| QS (150000 elements) | 386 | 27.8 |
| TSP (19 cities) | 99 | 735.6 |

Figure 1 shows the relative speedup of our applications. All applications except QS exhibit a relatively good speedup. One reason for this good speedup is that the Etrax processor does not have hardware support for floating point arithmetic. Therefore, even with a small problem size, the computation-to-communication ratio (CCR) is still quite large; the shared memory exchange between processors is small (see Table 4) causing low overhead from the underlying system. This is confirmed in QS, which was running with large problem size but the CCR is low and the shared memory is accessed heavily.

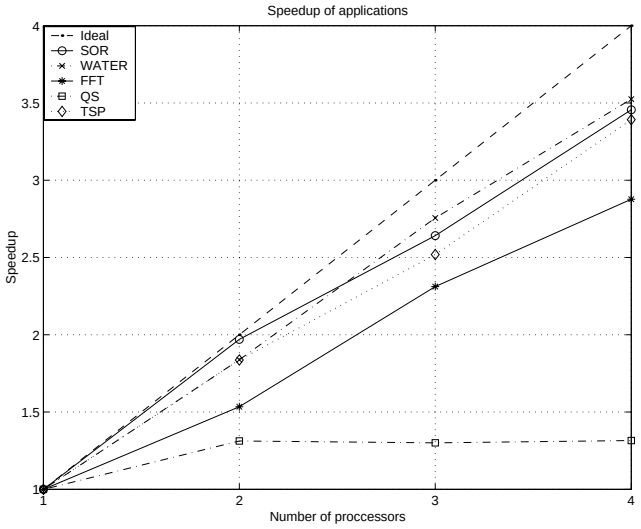


**Figure 1. Speedup of benchmark applications.**

### *B. FFT*

Figure 2 shows a typical diagram of the memory usage for the four nodes in eCVM after the initialization stage. From now on, with *memory* without further explanation we mean the local memory used in the system only. The shared memory space used in the application is not shown in the figures.

The four diagrams in the figure show the amount of memory allocated in eCVM for the four different processors. We can see that node 0 always uses 32 KByte more memory than the other does after the initialization stage (time 0). The graphs for nodes 1-3 are show a characteristic with spikes of high memory usage. A spike indicates a synchronization point—i.e., a new consistency protocol interval—, in which memory is allocated for creating diffs that are sent to home. Then that memory and the corresponding twins are deallocated. The increase of memory usage between spikes shows that a write fault on a remote page occurred and a twin was made. There is also a slight steady increase of allocated memory after each synchronization point. This small amount of memory is used to store the time stamp of pages (the version of page) to keep track of how pages are modified. However, we would only need to keep the last time stamp but since this memory leakage is quite small, we have chosen to ignore it for now.

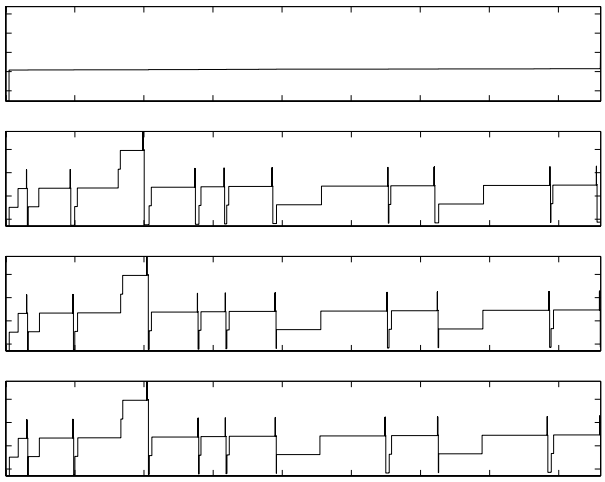

**Figure 2. Memory usage vs. time in FFT (16x16x16)**

### *C. SOR*

In SOR, the memory used for matrix elements is allocated in shared memory. However, communication only occurs across the boundary rows between bands and it is for read only.

Figure 3 shows that many twins and diffs are generated regularly when executing SOR. This is because the matrix is allocated continuously in the shared memory space; therefore, it may happen that a processor owns a row over a page boundary and the next page is assigned to another processor. It also may happen that a boundary row spans over two pages although its size was less than a page size. This causes more memory access faults than expected.

Although the overhead for creating twins and to generate diffs is low [10], it is simple to insert some pads before and after a boundary row to make sure that it spans over one or more entire pages. We observed that this effectively removed the overhead of creating twins and generating diffs. It also reduced the number of page faults.

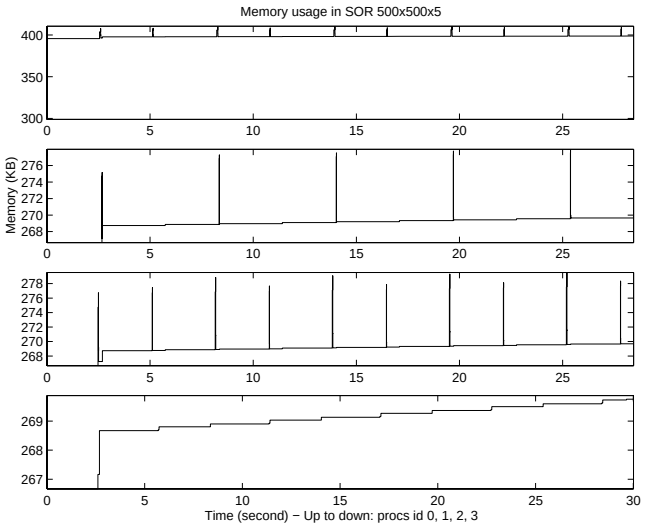


**Figure 3. Memory usage in eCVM for SOR before inserting pads.**

In both cases, with and without inserting pads, although the matrix was divided into balanced parts for each of processors, there was still load imbalance (see Figure 4). Processors 0 and 2 had to wait longer time than processors 1 and 3 at barriers.

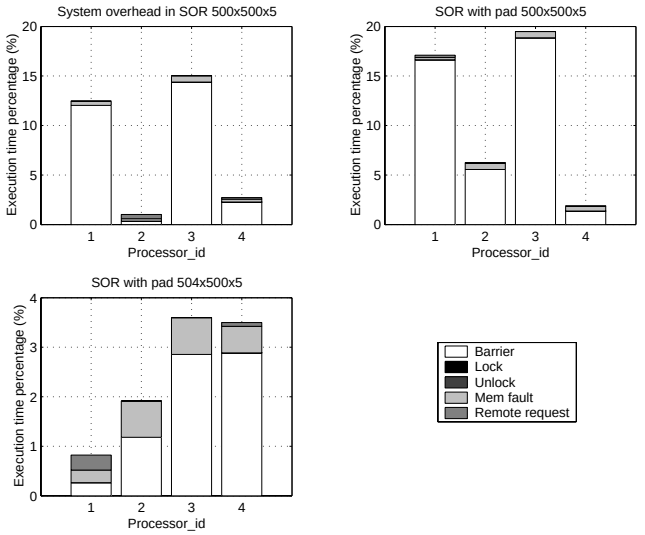


**Figure 4. System overhead time specification in SOR.**

In each iteration, the computation on each processor is divided into two parts separated by two barriers. The first part computes all black elements; the second computes all red elements. If the number of rows is not divisible with eight, when running on four processors, then processors 0 and 2 start with a black element and the processors 1 and 3 start with a red element. Although the total number of red and black elements assigned to each processor is equal, the number of red elements is different for processors 0 and 2 compared with 1 and 3, and so does the number of black elements. This caused a load imbalance between two parts on a processor (see Table 5). We call it local load imbalance.

**Table 5. Average time spent on each part of each processor in one iteration.**

| | Processor time (ms) | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 500, black | 2782 | 2841 | 2838 | 2829 |
| 500, red | 2749 | 3398 | 2781 | 3223 |
| 504, black | 2815 | 2828 | 2846 | 2797 |
| 504, red | 2772 | 2783 | 2802 | 2786 |

By using a number of rows divisible by 8, e.g. m=504, this local load imbalance was isolated and we thus got a speedup of 3.9 on 4 processors instead of 3.5 (see Figure 5).

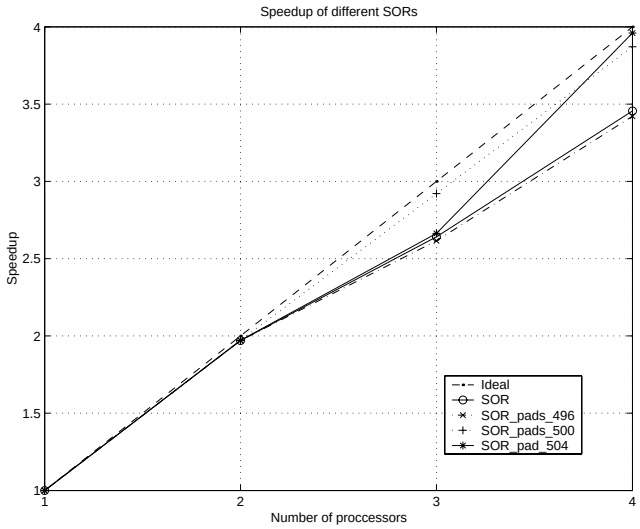


**Figure 5. Speedup of a: SOR m = 500, b: SOR padded m=500, c: SOR padded m=504**

### *D. QS and TSP*

The quick sort (QS) and traveling salesman problem (TSP) algorithms are well described in [5]. They parallel versions use the same algorithms. However, recursion is replaced by a shared pool of jobs in which all processors can take out a job in turns and put new jobs. Load balancing is implied in this sort of algorithms, because the one, which served heavier jobs, was served less number of jobs than others. Locks are used to protect and synchronize the shared memory among processors instead of barriers.

When running on 4 processors, the overhead of S-DSM system took average of 61% of the execution time of QS and about 8% in that of TSP. We gained a speedup of 1.3 with 2 processors and the same speedup with 4 processors in QS. This can be explained by the workload of QS was small (see Table 4).

As shown in Figure 6, the memory used for making twins and diffs was as much as double the amount of shared memory allocated. Intuitively, one might think that distributing pages among processors may reduce the local memory usage because it will reduce the number of diffs and twins. However, this did not help. The overhead of the S-DSM system even increased. One reason is that although the number of diffs and twins was reduced, the number of exchanged messages was increased and the overhead of

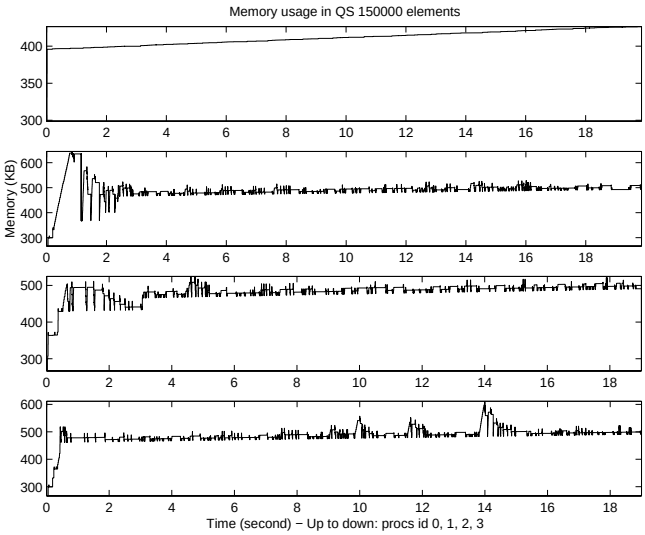


**Figure 6. Memory usage in QS, 150000 elements**

sending messages is well-known from other studies on S-DSM systems [8].

## V. DISCUSSION AND CONCLUSION

One way of reducing the duplication of the initialization step is to implement a "fork-like" function. As we all know, fork creates a new process and copies the environment and parent page tables to create a unique task structure for the child. Then the "parent simulation" is unnecessary, the child processes will only access memory pages it really needs. It thus reduces the memory consumed in those processes for shared memory. However, with this solution, the parent process still suffers from the initialization step. It accesses all shared memory pages during the initialization; therefore lack of memory is still a problem.

A large number of shared pages requested in QS were because of its upper bound parameters:
MAX_SIZE = 768K elements which is about 384 pages. The amount of memory really used was 150000 elements, which corresponds about 74 pages. However, by a property of Linux, a memory page is only allocated at the first reference, therefore this large amount of pages did not affect the amount of shared memory. The only negative impact it had was the increase of local memory for managing pages. This cost 49 Kbytes extra. Thus, a carefully selection of the initial parameters for each architecture in an S-DSM system will reduce large amount of memory usage, especial in embedded systems.

The local memory used for making twins and generating diffs may as large as double the shared memory used. Distribution of home pages among all processors may reduce this memory but it may increase the memory used for producing messages at synchronization points. This was confirmed in figures of memory used in TSP and TSP_D (home pages distributed over processors).

For producer/consumer problems, like the boundary rows in SOR, dynamic mapping of shared memory will reduce a large amount of memory for creating twin and making diffs in the consumer. Clearly, every time a page fault occurs in the consumer, it fetches the page from home (producer) into its local memory only. Whenever the producer updates its data, it should announce that to the consumer. This is an adaptation of the invalidation protocol.

We have shown in this study that it is possible to achieve high speedup with a software distributed shared memory system in a cluster of embedded processing nodes. However, lack of memory resources is a major problem in S-DSM systems on such platforms. However, we have only studied with a maximum of four processors. A study with 16 or more processors for scalability would give better useful information.

Due to the difficulty of finding and parallelizing applications to run on embedded system, all benchmark applications were taken from scientific computing world. They are rarely run in this kind of systems. A study on classifications of common applications run on embedded system and their parallelized versions would be of interest.

## ACKNOWLEDGMENT

This research has been funded by the Swedish Foundation of Strategic Research. Devices have been graciously provided by Axis Communications [2].

## REFERENCES

[1] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu and W. Zwaenepoel. *TreadMarks: Shared Memory Computing on Networks of Workstations*, IEEE Computer, Vol. 29, no. 2, pp. 18-28, February 1996.

[2] Axis Communications, http://www.axis.com

[3] J. W. Chung, B.H.Seong, K. H. Park, D. Park. Moving Home-Based Lazy Release Consistency for Shared Virtual Memory Systems, in *proceedings of the 1999 International Conference on Parallel Processing*, Septembet 1999.

[4] L. Clarke, I. Glendinning and R. Hempel, *The MPI Message Passing Interface Standard*, The MPI Forum, March 1994

[5] T.H.Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[6] A. Gupta, J. Hennessy, C. Holt. *Stanford Parallel Applications for Shared Memory*. http://www-flash.stanford.edu/apps/SPLASH/

[7] P. Keleher, *CVM: The Coherent Virtual Machine*, Tech. Report, Department of Computer Science, University of Maryland, July 1997.

[8] E. W. Parsons, M. Brorsson and K. C. Sevcik, *Predicting the Performance of Distributed Virtual Shared Memory Applications*, IBM Systems Journal, Volume 36, No. 4, 1997, pp. 527-549.

[9] R. Samanta, A. Bilas, L. Iftode, and J. P. Singh. *HomeBased SVM Protocols for SMP Clusters: Design and Performance*. In Proc. of the 4th IEEE Symp. on High-Performance Computer Architecture, pages 113--124, February 1998.

[10] W. Shi, W. Hu, Z. Tang. *Where does the time goes in S-DSM systems: Experiences with JIAJIA*, Journal of Computer Science and Technology, Vol 14, No.3, pp.193-205, May 1999

[11] C. D. Snyder, *Faster & Wider Performance Parts*, Microprocessor report, 2/19/02-03.