DSM*Sim*: A DISTRIBUTED SHARED MEMORY SIMULATOR FOR CLUSTERS OF SYMMETRIC MULTI-PROCESSORS

Darshan Thaker Vipin Chaudhary
Institute Of Scientific Computing
Wayne State University
Detroit, MI 48202

Abstract

*Distributed shared memory systems have become popular as a means of utilizing clusters of computers for solving large applications. We have developed a high-performance DSM at Wayne State University. To improve the performance of our DSM, we have developed a memory hierarchy simulator that allows us to compare various techniques very quickly and with much less effort. This paper describes our simulator, DSM*Sim*. We show that the simulator's performance closely matches the real system and demonstrate potential performance gains of up to 60 percent after adding optimization features to the simulator. The simulator also accepts the same code as the Software Distributed Shared Memory.*

1 Introduction

In recent years, single processor based computers have evolved rapidly. Processor speeds are now in excess of 2GHz. Yet, they are not able to solve increasingly large and complex scientific and engineering problems. Another trend is the decline in the number of specialized parallel machines being built to solve such problems. Instead, many vendors of traditional workstations have adopted a design strategy wherein multiple state-of-the-art microprocessors are used to build high performance shared-memory parallel workstations. From a computer architecture point of view, such machines could use tightly coupled processors that access the same shared memory, known as symmetrical multiprocessors (SMPs), or one could cluster multiple computing nodes with a high-performance interconnect. The latter approach has been extended by combining multiple SMPs to provide a scalable computing environment. Examples of this important class of machines include the IBM SP2, SUN Enterprise and SUNFire Series and machines from HP.

These symmetrical multiprocessors (SMPs) are then connected through high speed networks or switches to form a scalable computing cluster. A suitable runtime system should allow parallel programs to exploit fast shared memory when exchanging data within nodes and using the slower network only when necessary. Existing sequential application programs can be automatically converted to run on a single SMP node through the use of parallelizing compilers such as SUIF [1], etc. However, using multiple nodes requires the programmer to either write explicit message passing programs, using libraries like MPI[5] or PVM[6]; or to rewrite the code using a new language with parallel constructs ex. OpenMP[3], HPF and Fortran 90. Message passing programs are cumbersome to write and have to be tuned for each individual architecture to get the best possible performance. Parallel languages work well with code that has regular data access patterns. In both cases the programmer has to be intimately familiar with the application program as well as the

target architecture. The shared memory model is easier to program since the programmer does not have to worry about the data layout and does not have to explicitly send data from one process to another. However, hardware shared memory machines do not scale that well and/or are very expensive to build. Hence, an alternate approach to using these computing clusters is to provide an illusion of logically shared memory over physically distributed memory, known as a Distributed Shared Memory (DSM) or Shared Virtual Memory (SVM). Recent research projects with DSMs have shown good performance, for example IVY [9], TreadMarks [4], Quarks[8], and CVM[10].

At Wayne State University, we developed a DSM called Strings. The Strings system consists of a library that is linked with a shared memory parallel program. Strings is a page based DSM, that uses an update protocol for Release Consistency. The performance and capabilities of Strings are described in detail in previous work[1]. Currently, our effort is to enhance the performance of Strings. This effort includes studying in detail how Strings manages memory and designing different methods to alleviate the areas that act as a drag on efficiency. Towards this end we have developed an execution driven memory hierarchy simulator that closely mimics the behavior of Strings. The simulator allows us to test different approaches and ideas quickly and at a cost far lesser than what we would have incurred if we had modified Strings itself. In the rest of this paper, we review some features of Strings in section 2, describe the simulator and how it works in sections 3 and 4. Section 5 deals with the performance of the simulator and we conclude in section 6.

2 Strings

The Strings distributed shared memory was derived from the publicly available Quarks[8]. It consists of a library that is linked with a shared memory parallel program. The system allows the creation of multiple application threads on a single node, thus increasing the concurrency level on an SMP cluster. Shared memory in the system is implemented by using the UNIX mmap call to map a file to the bottom of the stack segment. The mprotect call is used to control access to the shared memory region. When a thread faults while accessing a page, a page handler is invoked to fetch the contents from the owning node. Strings currently supports sequential consistency using an invalidate protocol, as well as release consistency using an update protocol [5, 14].

Strings utilizes the concepts of *twins* and *diffs* to allow multiple application threads on the same node. All shared regions are mapped to two different addresses. This makes it possible to write to the secondary region, without changing the protection of the primary region. The copy of the page that is mapped to the secondary region is called a *twin*. The thread then modifies the *twin*. When the pages have to be synchronized, the difference between the original page and its twin is computed. These are called the *diffs*, which are then sent to the remote nodes that share the page.

For further details, we invite you to refer to a previous paper[1] that discuss Strings in greater detail.

3 Simulator for Strings

Previous work has demonstrated the performance capabilities of Strings. Nonetheless, there exist various areas where improvements and performance enhancements can be achieved. Some of these improvements can be in the area of networking, memory management, etc. With regard to memory management there are various approaches that can potentially change and improve the manner in which Strings behaves.

3.1 Motivation

To experiment with certain memory management functions in Strings would be time consuming. And consider that even after the change was accomplished there is no guarantee that this would positively effect the outcome in terms of performance enhancement. It is with this in mind that we decided to build a simulator for Strings. It would be easy to make a modification in the simulator and see the effects it would produce. Provided the performance enhancement was satisfactory, the changes could then be incorporated into Strings. Thus, it would function as a test bed for nascent ideas and design philosophies.

The simulator could also serve as a teaching tool, in that it would allow a student unfamiliar with the complexities of the distributed shared memory system, to understand the underlying principles and reasons behind certain design decisions.

3.2 Simulation system details

The simulator is based on Augmint[7]. Nonetheless, it is portable and can also be used on SUN machines with UltraSparc Architectures (using ABSS[11]). Some of the features of the simulator are -

- Accurately mimics the behavior of Strings
- It works with real applications that interact with the environment
- Applications that work on the simulator will work on Strings.
- Provides 'ease of use' with respect to understanding and modifying the simulation architecture
- Runs on both Intel and UltraSparc architectures.
- Manages the scheduling of events.
- Recognizes memory references.

The subsystems of the simulator are -

- Augmint Doctor - the x86 code augmenter
- Simulation Infrastructure (Augmint) - including event management, task scheduling and thread switching.
- The Distributed Shared Memory model under study.
- Applications - written using ANL m4 macros.

3.3 Augmint

Augmint is a software package on top of which multiprocessor memory hierarchy simulators can be constructed. Augmint consists of a front end memory event generator and a simulation infrastructure which damages the scheduling of events.

3.4 Augmint Doctor

The Doctor takes an x86 assembly source file as input and analyzes it for memory references. Each instruction that performs a memory reference causes additional code to be added before or after that instruction. This code handles details like saving state and setting up the simulation event structure.

3.5 The Simulation Infrastructure

The infrastructure provides the mechanisms for scheduling events and accounting for the passing of time. The important concepts here are: events, tasks, and the task scheduler.

3.6 Simulator Build and Execution Model

DSM*Sim* is an execution based simulator. The simulator build process (Figure 1)creates an executable. The build process is as follows - The source code of the application under study is passed through the macro m4 preprocessor where all the ANL like macros are expanded. (This is another advantage of the simulator, that all the applications that run on the simulator, run on Strings. The same application source code can be used for both the simulator and Strings. The difference is at the m4 preprocessor stage). Following this, the C compiler is run that converts the

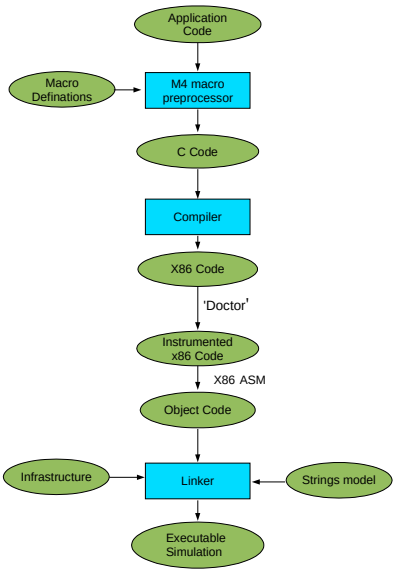

Figure 1. Dsm*Sim* Build Process for the Intel Architecture.

C code into assembler code. The Augmint Doctor is then used to augment the assembler code. This is then lined with the simulation libraries. The final result is an executable that contains the application code, the distributed shared memory system under study, and the simulation infrastructure. When running, this executable consists of several user level threads that act as simulated processors and a simulation thread that acts as the scheduler.

4 Implementation Details

When developing DSM*Sim*, the first goal was to replicate the behavior of Strings as accurately as possible. When running a simulation executable, the programmer passes the simulator arguments and the application arguments. When the simulation binary is executed, the first thing that is done is that all the shared memory is allocated. The simulator uses hash tables to keep track of the pages that reside on each node. Thus every node in the simulation has a hash table, the entries of which are the pages that reside on that node. The base address of the page is used when inserting into the tables. Every page is owned by one node. Page ownership is migratory at first fault.

4.1 Memory References and Synchronization

Every memory reference has to be analyzed. An access may either be a read or a write and may either be to a shared memory location or to one that is not shared. If it is not an access to a shared region, execution continues. However, if it is a read or write to a shared region, the simulator looks up the page associated with the reference. Depending on the page permissions for that node, a hit or a fault is determined to have occurred. When it is a page fault, a clean copy of the page is retrieved. The consistency model in the simulator was the release consistency model using the update protocol. The simulator provides locks, barriers, and condition variables. In the update protocol, diffs are sent to all pages in the copy-set when a node releases a lock. When a barrier is acquired, all dirty pages are flushed. This technical report describes in greater detail the implementation of the simulator and how it can be used.

5 Performance Analysis

We used applications from the Splash-2 [2] Benchmark Suite. FFT performs a transform of n complex data points and requires three all-to-all interprocessor communication phases for a matrix transpose. The data access is regular. LU-c and LU-n perform factorization of a dense matrix. The non-contiguous version has a single producer and multiple consumers. It suffers from considerable fragmentation and false sharing. The contiguous version uses an array of blocks to improve spatial locality. Radix performs an integer radix sort and suffers from a high-degree of false sharing at page granularity during a permutation phase. The matrix multiplication program uses a block-wise distribution of the resultant

matrix. Each application thread computes a block of contiguous values, hence there is no false sharing.

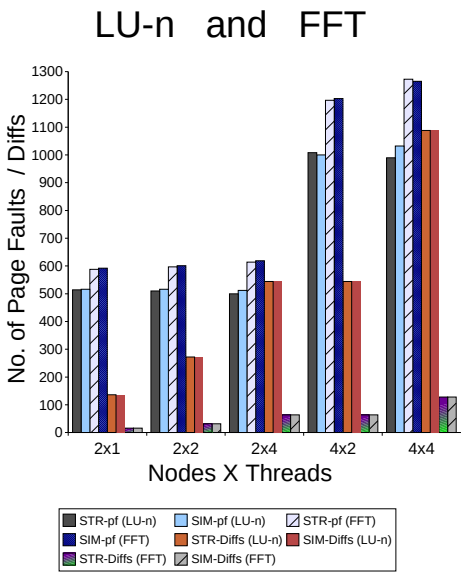

Figure 2. LU-n and FFT: Strings and Dsm*Sim*

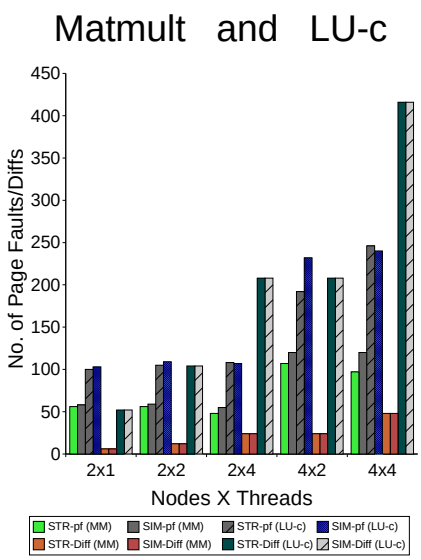

Figure 3. MatMult and LU-c: Strings and Dsm*Sim*

As stated earlier, the simulator can run on the Intel and the Sparc platforms. In most cases, a single CPU (600MHz Pentium) with 192MB RAM was used for the runs. Strings on the other hand was run on a cluster of SUN Enterprise 3500s each with 4 UltraSparc II processors at 330MHz and with 512MB of RAM. The following programs were utilized for our experiments - LU-c, FFT, Radix, LU-n and Matrix Multiplication. We judged our results by how closely the number of page faults in DSM*Sim* matched that of the real system. Since every memory access was trapped by DSM*Sim*, this gave us the ability to collect great amounts of data that gave details of the behavior of the different applications. We could collect details on the number of hits, faults, types of accesses, etc., for each page, each thread, and each node. In addition we could calculate the number of *diffs* that would be generated. It was also possible to see when, as a result of the update protocol, a thread releasing a lock would send *diffs* to another thread that was no longer interested in them. The biggest advantage of gathering such vast amounts of data is that it gives the application developer an idea of how to partition the data, to boost performance.

Figures 2 and 3 show the results comparing DSM*Sim* and Strings. It can be observed that most of the runs resulted in a negligible difference between Strings and DSM*Sim*. The runs shown in the figure are 128x128 matrix multiplication; FFT with 65536 complex doubles, LU decomposition on a 512x512 matrix with a 16x16 block size. We show the number of page faults in the simulator(SIM-PF) and in Strings(STR-PF), the locks and barriers for the simulator (SIM-L and SIM-B) and for

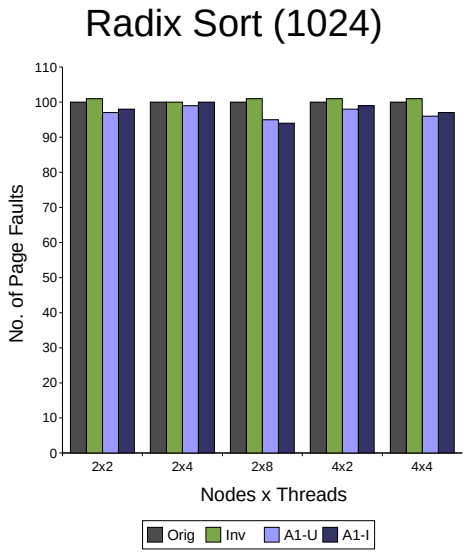

Figure 4. Perf. Enhanacements: Radix

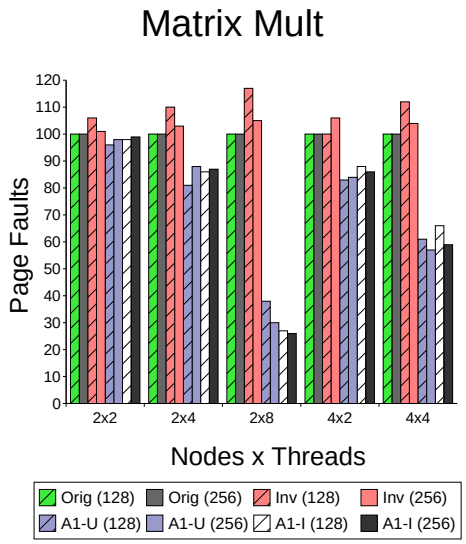

Figure 5. Perf. Enhancements: Matmult

Strings (STR-L and STR-B). In the legend, STR is for strings and SIM is for DSM*Sim*. The average difference in number of page faults is 1.2 percent, with the worst case being a difference of twelve percent. The difference in the page fault reading stems from the fact that DSM*Sim* runs as a single process, whereas Strings spawns multiple processes that in turn spawn threads on multiple machines. As a result, in Strings, if two threads on the same node access the same page (causing a fault) at the same time, this is regarded as just one fault. However, in DSM*Sim*, this would be considered two faults. From our results we can deduce that DSM*Sim* accurately mimics Strings.

From the accompanying figures 2 and 3, it can be observed that most of the runs resulted in a negligible difference between the Strings and the simulator. The runs shown in the figure are 128x128 matrix for matrix multiplication. For FFT 65536 complex doubles, for Lu 512x512 matrix and a 16x16 block size. The messages measured are protocol messages. Messages that are sent as a result of UDP based networking in Strings are not considered in the simulator. Thus we could deduce that the simulator was behaving very much like Strings would. The next challenge was to add features to the simulator that would yield an increase in performance.

The following features were added to the simulator. We changed the relationship between the threads and the nodes. Since the ownership of the pages is based on the nodes, this changed the manner in which the threads were accessing the pages. We included an invalidate protocol. Another change was the manner in which owners are assigned to pages. In Strings, pages are initially assigned to node zero, thereafter on first fault the ownership changes hands. We kept this feature and also assigned pages in a round robin fashion without changing ownership at first fault. The user can select either of the new options or the original system behavior by selecting the appropriate command line arguments at runtime. Fig-

ures 4 and 5 show the effect of the different features that were added. On the 'y axis' we plot the number of page faults(normalized). The original version uses a update protocol and release consistency. The second column is an invalidate protocol and the last two columns are both the update(A1-U) and invalidate(A1-I) protocols with a different scheme of assigning data. It can be observed that the most effective feature added was the first one. For matrix multiplication there is a significant improvement when we have more threads per node. The same trend is seen in radix, although the improvements are not so significant. It can be seen that the invalidate protocol when compared to the update does not always give an improvement in the number of faults.

6 Conclusions and Future Work

We developed a distributed shared memory simulator, DSM*Sim* that effectively matches the behavior of Strings. It also provides a portable platform for conducting research in memory consistency models and protocols. In addition, the simulator can be used as a teaching tool to understand the complexities of a DSM. DSM*Sim* is portable across both Sparc and Intel platforms. Various applications can easily be ported for the simulator. It is also very modular and allows a researcher to easily add features and change its behavior. Its ability to run on a single CPU machine is invaluable as one does not need expensive SMP machines to study DSM characteristics.

Our future work includes changes that we have observed as advantageous, into Strings. We plan on adding features related to memory consistency models and coherency protocols and comparing their characteristics. In addition, features like data visualization will be added in the future to effectively study and understand the data that is generated by the simulator.

References

[1] S. Roy and V. Chaudhary, "*Strings:* A High-Performance Distributed Shared Memory for Symmetrical Multiprocessor Clusters," in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, (Chicago, IL), pp. 90 –97, July 1998.

[2] S. C. Woo, M. Ohara, E. Torri, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of the International Symposium on Computer Architecture*, pp. 24–36, June 1995.

[3] The OpenMP Forum, OpenMP: Simple, Portable, Scalable SMP Programming, http://www.openmp.org/.

[4] P. Keleher, A. Cox, S. Dwarkadas and W. Zwaenepoel, TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proc. of the Winter 1994 USENIX Conference*, 1994.

[5] Message Passing Interface (MPI) Forum, MPI: A message-passing interface standard, *International Journal of Supercomputing Applications*, 8(3/4), 1994.

[6] V. S. Sunderam, "PVM: A framework for parallel distributed computing," in *Concurrency: Practice and Experience*, vol. 2(4), pp. 315–339, Dec. 1990.

[7] A.-T. Nguyen, M. Michael, A. Sharma, and J. Torrellas. The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures.*In Proceedings of the 1996 IEEE International Conference on Computer Design (ICCD)*, October 1996

[8] D. Khandekar. Quarks: Distributed shared memory as a basic building block for complex parallel and distributed systems. *Technical Report, University of Utah*, March 1996.

[9] K. Li, "IVY: A Shared Virtual Memory System for Parallel Computing," in *Proceedings of the 1988 International Conference on Parallel Processing*, August 1988, pp. 94-101.

[10] H. Han, C.-W. Tseng, and P. Keleher. Eliminating barrier synchronization for compiler-parallelized codes on software DSMs. In *International Journal of Parallel Programming*, October 1998

[11] D. Sunada, D. Glasco, M. Flynn. ABSS v2.0: a SPARC simulator in *The proceedings of the 8th Workshop on Synthesis And System Integration of Mixed Technologies*, 1998