Distributed Shared Memory: To Relax or Not to Relax?

Vadim Iosevich and Assaf Schuster

Technion–Israel Institute of Technology
Computer Science Dept.
{vadim_ds,assaf}@cs.technion.ac.il

Abstract. Choosing a memory consistency model is one of the main decisions in designing a Distributed Shared Memory (DSM) system. While Sequential Consistency (SC) provides a simple and intuitive programming model, relaxed consistency models allow memory accesses to be parallelized, improving runtime performance. In this article we compare the performance of two multithreaded memory coherence protocols. The first protocol implements Home-based Lazy Release Consistency (HLRC) memory semantics and the second one implements SC semantics using a MultiView (MV) memory mapping technique. This technique enables fine-grain access to shared memory while using the virtual memory hardware to track memory accesses. We perform an "apple-to-apple" comparison on the same testbed environment and benchmark suite, and investigate the effectiveness and scalability of both these protocols.

1 Introduction

A *Distributed Shared Memory* (DSM) system provides the distributed application with an abstraction of the shared address space in such a way that all data stored in this space is shared between all nodes in the cluster. Generally, each node uses its local virtual memory as a cache of the shared memory, often identifying the presence of data in the local cache by utilizing the virtual memory hardware. If the data is located on a remote node, the DSM system is responsible for fetching it, while maintaining the correctness of the shared memory. This concept was first proposed by Li and implemented in the first software DSM system, named IVY [1]. In order to keep the cache in a coherent state, the shared data is grained to atomic segments, like lines in a real cache. These segments are called *coherency units*.

A formal specification of how memory operations appear to execute to the programmer is called a *memory consistency model*. Since the introduction of Lamport's now-canonical sequential consistency (SC) model [2], various consistency models have been proposed by researchers [3, 4]. The idea of these models is to postpone the propagation of coherence information until synchronization points are reached. There are two types of synchronization operations, `ACQUIRE` and `RELEASE`, used respectively to obtain and yield exclusive access to shared data.

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 198–205, 2004.

Lazy Release Consistency (LRC) [4] is a refinement of the *Release Consistency* (RC) model [3]. The RC model requires that shared memory accesses be performed globally upon a `RELEASE` operation only. The idea of LRC is to make those accesses visible only to the processor that acquires a lock rather than perform all operations globally. False-sharing is alleviated by allowing different processes to access the same page simultaneously if these operations are not synchronized. A home-based implementation of LRC (HLRC) was proposed by Iftode [5]. In this implementation each shared page has an assigned home node. This home node always hosts the most updated contents of the page, which can then be fetched by a non-home node that needs an updated version.

2 Contribution

This work compares the runtime performance of two memory coherence protocols: a multithreaded implementation of the HLRC model and an efficient multithreaded implementation of the SC model that uses a MultiView [6] memory mapping technique. **Both coherence protocols are implemented within the same DSM system, where all code that is not consistency-specific is used by both protocols**. We use the same benchmark suite, where all applications produce the same output for both tested protocols. We show that an efficient implementation of the SC memory model can match the performance of the HLRC coherency protocol for the majority of tested applications. We analyze the impact of multithreading on DSM performance and show that the majority of tested applications improve their performance in the multithreaded mode.

3 Implementation

Our implementation is based on a Millipede DSM system [6]. Millipede implements a technique called MultiView, which allows an efficient implementation of SC and fine-grain access to the shared memory. MultiView can completely eliminate false sharing, treating each shared variable as a coherency unit. If false sharing is to be eliminated, the DSM system must be aware of the size of each shared variable, and this information is naturally supplied by an application via allocation requests. That means that each variable must be allocated separately, and generally requires only a small change to an application source code. A small size of a coherency unit can result in a large number of faults. An application's data set can be allocated by chunks of few variables as a tradeoff between false sharing and data prefetching. Generally, there is an optimal allocation pattern for each application that results in a minimal execution time.

Niv and Shuster [7] proposed a mechanism that automatically changes the shared memory granularity during runtime. This mechanism, called the *dynamic granularity* was proven to be a successful technique for improving MultiView's performance. It is based on a history of shared memory accesses and aggregates variables in larger coherency units when the application accesses them coarsely.

When different nodes start to request different parts of this large coherency unit, it is disassembled to separate variables.

Our complementary work [8] details the efficient implementation of the HLRC memory coherence protocol that supports preemptive multithreading. Previous HLRC implementations proposed non-preemptive multithreading [9] or creating a process for each CPU in an SMP node [10, 11]. The only HLRC implementation that supports preemptive threads is mentioned by Antoniu and Bougé in [12].

The protocol is implemented over the *Virtual Interface Architecture* (VIA) [13] – a standard architecture for high-speed networking. The implementation details are not provided here for lack of space. To make a real "apple-to-apple" comparison, we port a previous version of MILLIPEDE to this communication layer in order to evaluate both protocols on the common substrate.

4 Performance Evaluation

In this section we compare and analyze the performance of two multithreaded shared memory coherence protocols: SC implemented with MULTIVIEW (further denoted as SC/MV) and HLRC. Our testbed environment is a cluster of twelve Compaq Professional Workstations AP550. Each node is an SMP PC with two 733MHZ Pentium-III processors, a 512KB L2 cache, a 512MB physical memory and a 32-bit/33MHz PCI bus. All nodes run the Win2000 operating system. The cluster is interconnected by the ServerNet-II [14] VIA-based network.

We also investigate the effect of chunk allocation in order to estimate the potential of the SC/MV technique and discover the best static granularity for the particular application. In addition, we try to estimate how the dynamic granularity change can boost the performance of the MULTIVIEW technique. Our system does not support the dynamic granularity protocol, but we try to estimate the runtime performance for the dynamic granularity on the basis of results presented in [7]. We estimate the performance gain achieved with dynamic granularity versus fixed granularity and the presented results are only an approximation.

4.1 Benchmark Application Suite

Our benchmark suite consists of two microbenchmarks, NBodyW and NBody; eight applications from the SPLASH-2 [15] benchmark suite (Barnes, Volrend, LU, Water-nsq, Water-sp, FFT, Radix and Ocean); and TSP and SOR from the TreadMarks [16] benchmark applications.

NBodyW is a microbenchmark that imitates a kernel of n-body applications. The program operates with a large set of 64-byte bodies and performs three phases as follows: (1) Each of the P application's threads reads the entire set of bodies. (2) Each of the P application's threads processes and updates $1/P$ of the bodies. The processing of a body is simulated by a constant-length busy loop. (3) A single thread updates all the bodies (sequential phase). NBody is a shortened modification of NBodyW that contains only the first two phases. Hence, this application contains one coarse phase and one fine phase.

Table 1. Benchmark characteristics. B stands for barriers, L stands for locks.

Application	Input data set	Shared memory	Sharing granularity	Synch	Allocation pattern
Water-nsq	8000 molecules	5.35MB	a molecule (672B)	B, L	fine
Water-sp	8000 molecules	10.15MB	a molecule (680B)	B, L	fine
LU	3072×3072	72.10MB	block (coarse)	B	coarse
FFT	2^{20} numbers	48.25MB	a row segment	B	coarse
TSP	A graph of 32 cities	27.86MB	a tour (276B)	L	fine
SOR	2066×10240	80.73MB	a row (coarse)	B	coarse
Barnes	32768 bodies	41.21MB	body fields (4–32B)	B, L	fine
Radix	10240000 keys	82.73MB	an integer (4B)	B, L	coarse
Volrend	a file "head.den"	29.34MB	a 4×4 box (4B)	B, L	fine
Ocean	a 514×514 grid	94.75MB	grid point (8B)	B, L	coarse
NBody	32768 bodies	2.00MB	a body (64B)	B	fine
NBodyW	32768 bodies	2.00MB	a body (64B)	B	fine

A detailed description of the other benchmarks can be found in related papers and is not provided here for lack of space. Table 1 summarizes input data sets and memory sharing characteristics of all tested benchmarks.

4.2 Performance Analysis and Comparison

Fig. 1 summarizes the speedups obtained for all applications, with one thread running on each node. We can divide the tested benchmarks into three groups, according to their performance with each of the two memory coherence protocols mentioned above.

The first group contains the majority of tested applications, for which the SC/MV protocol matches HLRC's performance, or for which the gap between the two can be eliminated by using the proper granularity level or dynamic granularity protocol. These applications are: NBody, NBodyW, Volrend, TSP, Water-nsq, Water-sp, Radix and FFT.

The second group comprises three applications: SOR, LU and Ocean, which achieve a better speedup with the SC/MV protocol. HLRC's poor performance for these applications is due to the excess coherence operations that are required by the consistency model but not justified in this specific case. Generally, these operations are performed on barriers, when all threads are notified about modifications performed by all other threads. Nevertheless, each thread processes its own part of a data set and is not interested in the modifications of other threads.

To the third group we ascribe Barnes only. For this application, the HLRC coherence protocol significantly outperforms the SC/MV, and this gap cannot be bridged by optimal or dynamic chunking levels. Dynamic granularity does improve the SC/MV protocol's performance, but it is still a long way from that of HLRC.

Investigating the impact of multithreading on DSM performance, we found that while generally beneficial, multithreading can also be detrimental in some

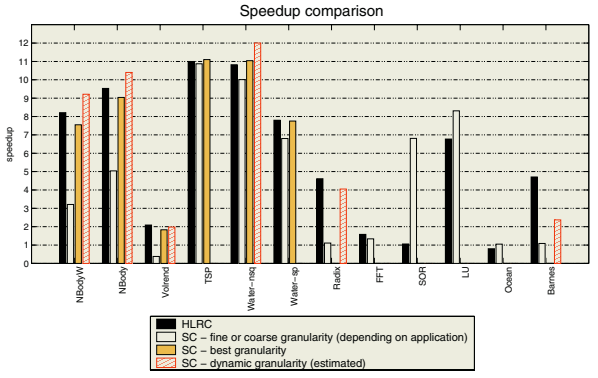

Fig. 1. Speedup comparison for all tested applications. The speedup is measured as the relation between the serial execution time and the minimal execution time on a 12-node cluster, where each node runs only one application thread.

cases. First of all, there are sequential phases performed by a single thread (as in NBodyW, Volrend and LU) or phases where the same task is performed by all threads regardless of their location, as in the first phase of NBody(W). Furthermore, multithreading does not always provide computation-communication overlap and can lead to increased contention in the shared memory protocol and communication layer. Consequently, multithreading affects all applications to a different extent, as can be observed from Fig. 2. In summary, the HLRC protocol benefits from multithreading, with an average performance improvement of 35.5%. The SC/MV protocol without optimized allocation improves its performance by 21.9% on average. With the optimized allocation pattern, the SC/MV protocol gains 30.7%.

5 Conclusions

The coherency state information that must be kept by the SC/MV protocol is quite simple. Only the presence of all page replicas must be tracked. In contrast, the HLRC protocol is much more memory-consuming and cumbersome. Tracking the causality relation between memory accesses requires complex data structures that must be referenced by different threads. This requires a very accurate mutual exclusion mechanism to keep the state data valid.

If data has to be fetched from a remote node as the result of a fault, this will always take one round trip in HLRC – a request message is sent to a home node that answers with the requested content. In the SC/MV protocol, however, this operation generally takes three messages. The first message is sent to a manager (if the faulting node is not the manager itself), and the second is forwarded by the manager to the page owner (if the manager is not the owner itself), which, in turn, replies to a requester. This results in a more costly fault in the SC/MV implementation. It should be noted that although chunk allocation increases the

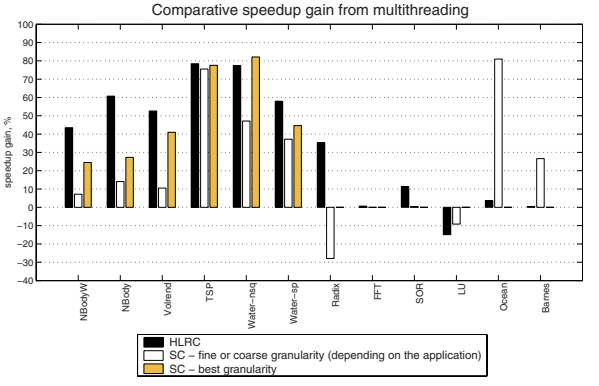

Fig. 2. Speedup gain from multithreading. The presented measure is the difference in the speedup achieved on a 12-node cluster with each node running one vs. two application threads.

average cost of a fault (due to the increased portion of data that must be sent to the requester), the overall number of faults is reduced, alleviating page fault overhead.

We found that the average speedup of the HLRC protocol is 5.97, while the average speedup of the SC/MV protocol with non-optimized allocation is 4.5. This means that HLRC exhibits a 32.7% performance advantage over a non-optimized SC/MV. Considering the best possible allocation pattern for each application, we raise the average speedup of the SC/MV protocol to 5.6, which is very close to the average HLRC speedup. This decreases the HLRC performance advantage to only 6.6%. Taking into account the speedup achieved by the SC/MV protocol if the granularity is changed dynamically at runtime, we get an average SC/MV speedup of 6.2.

6 Related Works

Dwarkadas et al. [17] compared the Shasta [18] (SC) and Cashmere [19] (RC) DSM systems, both of which were tuned to run on a cluster of four-CPU 400 MHz multiprocessors connected via a Memory Channel network. The authors concluded that for the eight applications that were written and tuned for hardware DSM systems, Shasta performed 6.1 times better than Cashmere. For the five programs that were written or tuned for page-based DSM, Cashmere performed 1.3 times better than Shasta. When all the tested applications were optimized separately for both coherence protocols, Cashmere performed 1.15 times better than Shasta. Both of the aforementioned protocols differ from those investigated in our study, and we ran our DSM on nodes with more powerful CPUs. Nevertheless, the results confirm our conclusion that the SC protocol, if implemented efficiently, performs no worse than relaxed memory consistency models. In addition, Shasta and Cashmere are two completely different DSM systems: they differ

not only in coherence protocols but also in implementing services such as synchronization primitives, communication, and the tracking of memory accesses. We implement both coherence protocols within the same DSM system where all code that is not consistency-specific is used identically by both protocols.

Additional research was conducted by Zhou et al. [20] on a noncommodity hardware system that supports access control at custom granularity. This allows the use of a uniform access control mechanism for both fine-grain and coarse-grain protocols. The authors tried to find the best combination of granularity and consistency protocols for different classes of applications. Three consistency protocols with four sizes of coherence granularity were tested: SC, single-writer LRC, and HLRC. The results show that no single combination of protocol and granularity performs best for all the applications. A combination of the SC protocol and fine granularity works well with 7 of the 12 applications. The combination of a HLRC protocol and page granularity works well with 8 of the 12 applications. It should be noted that the testbed environment used in [20] consisted of 66MHz processors, while our processors are an order of magnitude faster. In contrast to this study, we use standard hardware to implement DSM protocols.

Keleher [21] compared single- and multiple-writer versions of the LRC protocol, and the SC protocol, all implemented in the CVM software DSM system. Keleher found that the multiple writer LRC protocol performs an average of 34% better than the SC protocol. While not specified, the SC protocol used in this study is probably a page-based protocol and therefore cannot perform well for applications that use fine-grain data sharing. Nevertheless, it performs well for coarse-grain applications like FFT and LU. The results show that the SC protocol performs like multiple-writer LRC for FFT and outperforms it for LU. This confirms our results for these applications. The use of slow (66MHz) processors make it difficult to compare this study to our research.

References

1. Li, K.: Ivy: A shared virtual memory system for parallel computing. In: Proc. of the Int'l Conf. on Parallel Processing (ICPP'88). Volume 2. (1988) 94–101
2. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers **28** (1979) 690–691
3. Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P.B., Gupta, A., Hennessy, J.L.: Memory consistency and event ordering in scalable shared-memory multiprocessors. In: 25 Years ISCA: Retrospectives and Reprints. (1998) 376–387
4. Keleher, P., Cox, A.L., Zwaenepoel, W.: Lazy release consistency for software distributed shared memory. In: Proc. of the 19th Annual Int'l. Symp. on Computer Architecture (ISCA'92). (1992) 13–21
5. Iftode, L.: Home-based shared virtual memory (thesis). Technical Report TR-583-98, Princeton University, Computer Science Department (1998)
6. Itzkovitz, A., Schuster, A.: MultiView and Millipage–fine-grain sharing in page-based DSMs. In: Proc. of the 3rd Symp. on Operating Systems Design and Implementation (OSDI-99), Berkeley, CA (1999) 215–228
7. Niv, N., Schuster, A.: Transparent adaptation of sharing granularity in MultiView-based DSM systems. Software Practice and Experience **31** (2001) 1439–1459

8. Iosevich, V., Schuster, A.: Multithreaded home-based lazy release consistency over VIA. In: Proc. of the 18th Int'l. Parallel and Distributed Processing Symp. (IPDPS'04). (2004)
9. Rangarajan, M., Divakaran, S., Nguyen, T.D., Iftode, L.: Multi-threaded home-based LRC distributed shared memory. In: The 8th Workshop of Scalable Shared Memory Multiprocessors (held in conjunction with ISCA). (1999)
10. Stets, R., Dwarkadas, S., Hardavellas, N., Hunt, G.C., Kontothanassis, L.I., Parthasarathy, S., Scott, M.L.: Cashmere-2L: Software coherent shared memory on a clustered remote-write network. In: Symp. on Operating Systems Principles. (1997) 170–183
11. Bilas, A.: Improving the Performance of Shared Virtual Memory on System Area Networks. PhD thesis, Dept. of Computer Science, Princeton University (1998)
12. Antoniu, G., Bougé, L.: DSM-PM2: A portable implementation platform for multithreaded DSM consistency protocols. In: Proc. 6th Int'l. Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS '01). (2001)
13. Compaq, Intel and Microsoft Corporations: Virtual Interface Architecture Specification. Version 1.0. http://www.viarch.org (1997)
14. Heirich, A., Garcia, D., Knowles, M., Horst, R.: ServerNet-II: A reliable interconnect for scalable high performance cluster computing. Technical report, Compaq Computer Corporation, Tandem Division (1998)
15. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: Characterization and methodological considerations. In: Proc. of the 22th Int'l Symp. on Computer Architecture. (1995) 24–36
16. Keleher, P., Dwarkadas, S., Cox, A.L., Zwaenepoel, W.: Treadmarks: Distributed shared memory on standard workstations and operating systems. In: Proc. of the Winter 1994 USENIX Conf. (1994) 115–131
17. Dwarkadas, S., Gharachorloo, K., Kontothanassis, L., Scales, D.J., Scott, M.L., Stets, R.: Comparative evaluation of fine- and coarse-grain approaches for software distributed shared memory. In: Proc. of the 5th IEEE Symp. on High-Performance Computer Architecture (HPCA-5). (1999) 260–269
18. Scales, D.J., Gharachorloo, K., Thekkath, C.A.: Shasta: A low overhead, software-only approach for supporting fine-grain shared memory. In: Proc. of the 7th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOSVII). (1996) 174–185
19. Kontothanassis, L.I., Hunt, G., Stets, R., Hardavellas, N., Cierniak, M., Parthasarathy, S., Meira, Jr., W., Dwarkadas, S., Scott, M.L.: VM-based shared memory on low-latency, remote-memory-access networks. In: Proc. of the 24th Annual Int'l. Symp. on Computer Architecture (ISCA'97). (1997) 157–169
20. Zhou, Y., Iftode, L., Li, K., Singh, J.P., Toonen, B.R., Schoinas, I., Hill, M.D., Wood, D.A.: Relaxed consistency and coherence granularity in DSM systems: A performance evaluation. In: Proc. of the Sixth ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'97). (1997) 193–205
21. Keleher, P.: The relative importance of concurrent writers and weak consistency models. In: Proc. of the 16th Int'l. Conf. on Distributed Computing Systems (ICDCS-16). (1996) 91–98