Annual Report on NSF ESS Grant CCR-9705594
Integrated Software Shared Memory for Next-Generation Networks

fiscal year 1998-1999

Michael L. Scott
Sandhya Dwarkadas

Department of Computer Science
University of Rochester
Rochester, NY 14627-0226
{scott,sandhya}@cs.rochester.edu

April 1999

1 Introduction

It is widely accepted today that small-scale symmetric multiprocessors (SMPs) provide the best performance per dollar of any computing platform, and that shared memory provides the most desirable programming paradigm for parallel computing. Large-scale shared-memory systems can be supported via directory-based cache-coherent hardware or via software emulation of shared memory on top of message-passing hardware. Hardware coherence is usually faster, but software is cheaper and more easily customized. Software coherence may also be attractive on a large cache-coherent machine if the operating system does not support a system-wide shared address space. Unfortunately, software distributed shared memory (S-DSM) has traditionally provided acceptable performance for only a limited class of applications.

The Cashmere project attempts to provide most of the performance of a tightly-coupled cache-coherent multiprocessor on simpler, cheaper clusters of SMPs. Unlike most previous S-DSM systems, Cashmere integrates hardware coherence within SMP nodes with software coherence between nodes, and relies on an inter-node coherence protocol optimized for low-latency user-level communication. We also aim to integrate compile-time static analysis of sharing patterns with run-time coherence management. Our hardware platform consists of a 32-processor collection of 600 MHz AlphaServers, organized as an 8-node cluster of 4-way SMPs, connected by Compaq's Memory Channel network.

This document reports on results obtained from early 1998 through early 1999, and outlines plans for the third and final year of the Cashmere ESS grant.

2 Summary of Recent Results

Progress occurred this past year on four major fronts: support for very large (out-of-core) data sets, quantitative comparison of fine-grain and page-based coherence mechanisms, integration of compiler and run-time coherence management, and application studies. We also implemented several important but smaller system enhancements.

2.1 Support for Very Large Data Sets

Software distributed shared memory (DSM) systems have successfully provided the illusion of shared memory on distributed memory machines. However, most software DSM systems use the main memory of each machine as a level in a cache hierarchy, replicating copies of shared data in local memory. Since computer memories tend to be much larger than caches, DSM systems have largely ignored memory capacity issues, assuming there is always enough space in main memory in which to replicate data. Applications that access data that exceeds the capacity available in local memory will page to disk, resulting in reduced performance. We have implemented "VLM" (very large memory) extensions to Cashmere that take advantage of system-wide memory resources in order to reduce or eliminate paging overhead. Experimental results with Cashmere-VLM on a 4-node, 16-processor AlphaServer system at Compaq CRL were presented at *IPPS* this month [4]. An example of these results appears in figure 1.

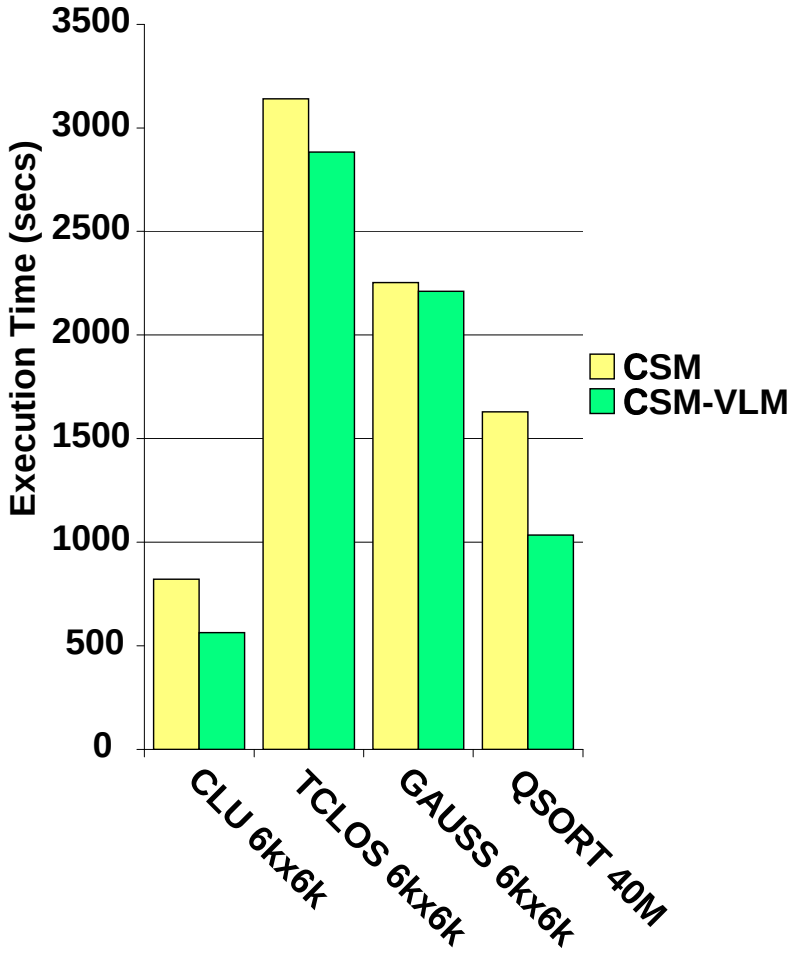

Figure 1: Cashmere-VLM performance results: execution times for four applications with very large data sets.

2.2 Comparison of Fine-Grain and Page-Based Coherence

Any S-DSM system must incorporate a mechanism to detect accesses to missing or out-of-date data. Two distinct approaches have been used: the *fine-grain* approach that instruments application loads and stores to support a small coherence granularity, and the *coarse-grain* approach based on virtual memory hardware that provides coherence at a page granularity. Fine-grain systems offer a simple migration path for applications developed on hardware multiprocessors by supporting coherence protocols similar to those implemented in hardware. On the other hand, coarse-grain systems can potentially provide higher performance through more optimized protocols and larger transfer granularities, while avoiding instrumentation overheads. Numerous studies have examined each approach individually, but major differences in experimental platforms and applications make comparison of the approaches difficult.

We capitalized on a unique opportunity this past year to perform a direct and balanced comparison. Specifically, we collaborated with the Shasta group at Digital/Compaq WRL to compare their S-DSM system to ours. Shasta and Cashmere are both mature systems, and they run on the same hardware platform. Our results show that Shasta is comparatively insensitive to the granularity of sharing and synchronization in programs, and in particular that it is better able to tolerate the fine-grain synchronization often found in applications originally developed for hardware multiprocessors. In contrast, Cashmere's performance can degrade dramatically in the presence of fine-grain synchronization, but for more scalable applications it provides better overall performance. Our results were presented at *HPCA* in January [3]; a summary appears in figure 2. As shown by the stacked bars in several applications, performance differences between Cashmere and Shasta can sometimes be reduced by program modifications that address coherence and synchronization granularity.

2.3 Compiler Integration

Work continued this past year on the integration of static compiler analysis with dynamic run-time coherence. Based on results reported last year, we are currently developing an interface to the DSM system that can be used as the target of a parallelizing compiler.

While workstation environments provide powerful and cost-effective computing platforms, they are often shared by multiple users, or may consist of heterogeneous machines. As a result, parallel applications executing in these environments must operate despite unequal (and varying) computational resources. For maximum performance, applications should automatically adapt execution to maximize use of the available resources. Ideally, this adaptation should be transparent to the application programmer. We are implementing CRAUL (Compiler and Run-Time Integration for Adaptation Under Load), a system that dynamically balances computational load in a parallel application. Results appear at LCR '98 [11] and in the Spring '99 issue of the *Journal of Scientific Programming* [10].

Our target run-time is software DSM—in particular, both the Cashmere and TreadMarks run-time systems. CRAUL combines compile-time support (implemented using SUIF as the base compiler) to identify data access patterns with a run-time system that uses the access information to intelligently distribute the parallel workload. The distribution is chosen according to the relative power of the processors and so as to minimize software DSM overhead. This same (access pattern)

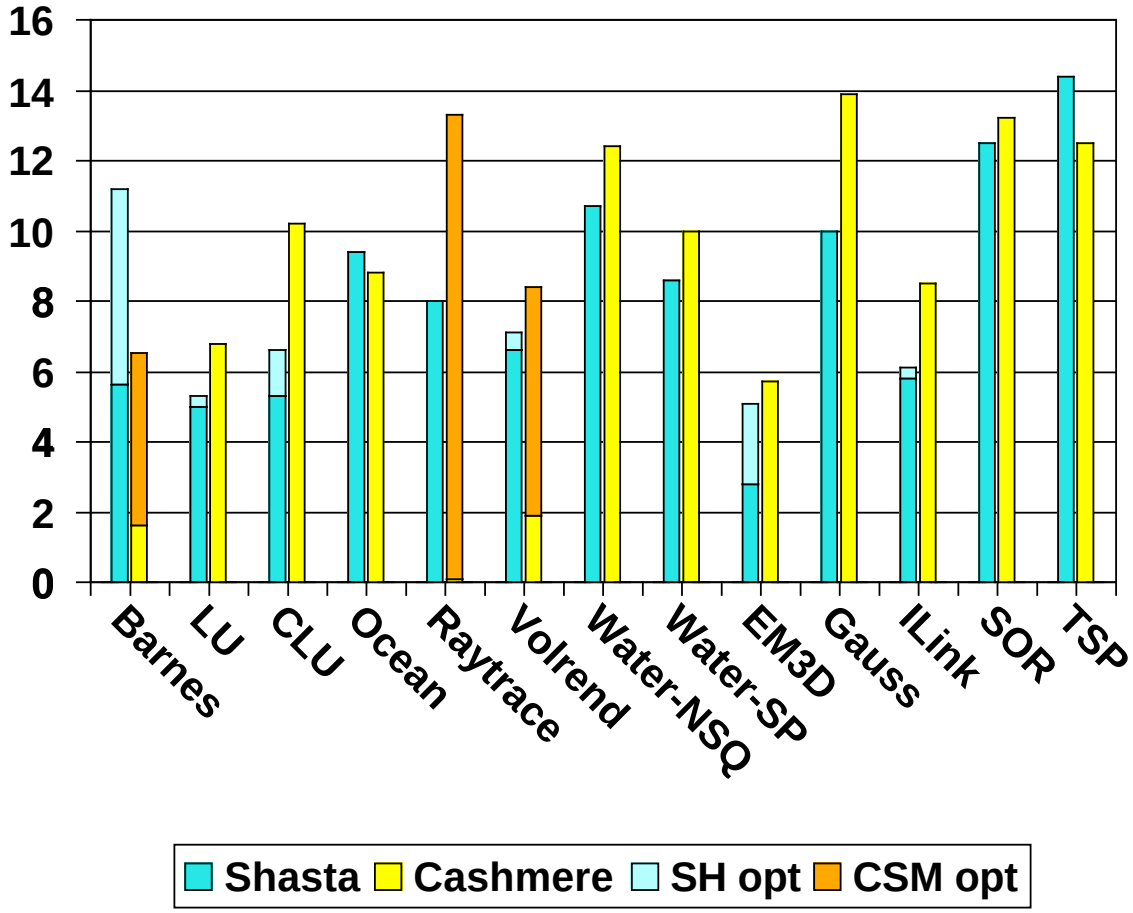

Figure 2: Cashmere-Shasta performance comparison: speedup on a 16-processor system. In the case of divided bars, the higher speedup reflects results after minor source-code changes that better tune the application to the underlying coherence protocol.

information is also used to optimize communication whenever possible by tailoring it to the underlying system. We have evaluated the resulting *locality conscious* load distribution in the presence of different types of load—computational, computational and memory intensive, and network load. CRAUL performs within 5—23% of ideal in the presence of load, and is able to improve on naive compiler-based work distribution that does not take locality into account even in the absence of load.

2.4 Applications

We made significant progress this year in application development for data mining and computer vision. We are in the early stages of collaboration with allied researchers in several other areas.

In the data mining domain, one area we have examined is the discovery of association rules [18, 13]. Association mining is the process of identifying commonly-occurring subsets in a database whose elements are sets. Our algorithms are a result of a detailed study of the available parallelism and the properties of associations. The algorithm uses a graph-theoretic (lattice) representation of frequent itemsets, and then partitions them among the processors. At the same time it uses a novel database layout that clusters related transactions together, and selectively replicates the database so that the portion of the database needed for the computation of associations is local to each processor. After the initial set-up phase, the algorithm eliminates the need for further communication or synchronization, and balances the computation and data across the processors. The application uses complex pointer-based data structures that typically suffer from sub-optimal locality. We found that for effective parallelization, attention needed to be paid to memory placement for improved locality and false sharing avoidance.

The discovery of frequent sequences in temporal databases is another important data mining problem. Most current work assumes that the database is static, and a database update requires rediscovering all the patterns by scanning the entire old and new database. We propose (and are in the process of implementing) novel techniques for maintaining sequences in the presence of a) database updates, and b) user interaction (e.g. modifying mining parameters). This is a very challenging task, since such updates can invalidate existing sequences or introduce new ones. In both the above scenarios, we avoid re-executing the algorithm on the entire dataset, thereby reducing execution time. Experimental results confirm that our approach results in substantial performance gains [14].

In the area of computer vision we designed and analyzed the performance of a parallel real-time probabilistic search algorithm to solve the *correspondence problem* [2]. We also have a preliminary implementation of a parallel object recognition algorithm. The object recognition system uses a multi-gigabyte in-core database composed of curve features representing various objects. An input image is decomposed into its curve elements and the strongest curves are matched against curves stored in the database to generate hypotheses about object identities and poses in the input image. The matching process in the database takes a large amount of time, most of it spent reading the database. The database is initially split with approximately the same number of elements in each partition and divided up among the processes. Each process is provided the key to search for and performs the search in its partition of the database. The lists of matches found by each thread are then concatenated and used to compose the hypotheses. The parallelization follows a manager-worker model, where the work is distributed by the manager when needed. Each worker performs

the search in the database, resynchronizing with the manager when done. The manager combines the generated lists and continues with the rest of the program. Efforts are currently under way to parallelize part of the hypothesis generation routines in a similar manner, and to determine the performance of the parallelization strategy.

Our other collaborations, on which we hope to report more completely next year, include a cosmological modeling application from Prof. Adam Frank of the UR Physics and Astronomy department, a cone-beam X-ray tomography application from Prof. Ruola Ning of the UR Radiology department, a 3-D surface reconstruction algorithm from Prof. Kyros Kutulakos of the UR Dermatology department, and a neural coding application from Prof. Dana Ballard of our own department.

2.5 Protocol Enhancements

We have this past year made several enhancements to the Cashmere programming environment. We now support Fortran 77 applications, in addition to C and C++, and we have developed a load-balancing interface for loop-based scientific applications.

As an outgrowth of our Cashmere/Shasta comparison study, we have developed a version of Cashmere that relies on fast message passing, but not on the remote-write, broadcast, or total-ordering properties of the Memory Channel. This version has allowed us to evaluate the relative importance of the various features of the Memory Channel. Our preliminary results [15] indicate that the latency of remote operations is the dominant factor in performance. Remote writes, broadcast, and total ordering all help, but on all but one of our applications the additional performance improvement is less than 10%. Moreover, by refraining from using remote writes to update home node copies we can eliminate Memory Channel-imposed restrictions on data set size (this step is required for the VLM version of Cashmere described above). We can also implement a straightforward home node migration mechanism that leads to significant performance improvements for migratory data. In general we found that the benefits of home node migration outweight the benefits of data propagation via remote write.

Finally, in a preipherally-related project called Coign, we developed a set of mechanisms that can be used to automatically partition a component (object) based application across a distributed system [8, 7, 6]. This work was supported in part by a graduate fellowship and hardware grant from Microsoft Research.

Coign operates on shrink-wrapped binary software. The Coign user exercises the application through a series of "profiling" runs, during which Coign collects statistics on cross-component communication. Coign then employs graph cutting and binary re-writing techniques to build a new version of the application that will distribute components across a network in a way that minimizes communication.

From the point of view of Cashmere, the key innovation in Coign is the ability to move software transparently across machine boundaries. In a follow-on project with Microsoft [16] we are using the Coign instrumentation tools to build a "component-ized" wrapper around the NT kernel interface. In an NT version of Cashmere, this wrapper would allow us to provide a single system image across a cluster of machines, with transparent access to the kernel resources of each machine.

3 Plans for the Upcoming Year

Plans for the coming year include

- Completion of our study of the impact of special network features on coherence protocol performance.
- Continued application development.
- Implementation of support for Java. We are investigating several alternative strategies. Our goals are to provide as convenient a programming model as possible (ideally extending Java's existing shared memory mechanisms across the cluster) while achieving high performance and minimizing or avoiding changes to the JVM.
- Development of a "three level" version of Cashmere that augments the current two-level system (hardware shared memory within nodes, low-latency software shared memory across nodes) with higher-latency sharing across wide area nets. In particular, we envision a graphical application front end (e.g. for program steering or interactive data mining) that shares access to the variables of a simulation running on a remote compute cluster.
- Investigation of temporally-based coherence extensions for chaotic (unsynchronized) applications.
- Implementation of cross-cluster support (e.g. with tools based on Coign) for a single "system image" for file access, process control, memory management, etc.

Depending on time constraints, we may also port Cashmere to a network of PCs running NT and connected by a Giganet network supporting the new VIA interface standard.

4 Education and Human Resources

Robert Stets, the senior student on the project, gave talks at SOSP '97, HPCA '99, and IPPS '99. Sotiris Ioannidis gave a talk at LRC '98. Srinivasan Parthasarathy presented a poster at LCR '98, and gave talks at KDD (Knowledge Discovery and Data Mining) '98, and the '98 Workshop on Distributed Data Mining. Nikolaos Hardavellas gave a talk at the 1998 SSMM (Scalable Shared Memory Multiprocessor) workshop held in conjunction with ISCA '98. Undergraduate Elliot Barnett, who worked on Cashmere for several months last year, has accepted a job with Frontier Communications. Two other undergraduates, Shawn Hershey and Peenak Inamdar, also worked on the project this year.

Current graduate students working on various aspects of Cashmere include Robert Stets, Srinivasan Parthasarathy, Umit Rencuzogullari, DeQing (Luke) Chen, Rajeev Balasubramonian, and Gregorios Magklis. Rob's thesis is on core Cashmere material, mainly coherence protocol implementation. Srini is working on systems support for client-server data-mining applications. Umit is working on compiler-runtime integration. Rob and Srini expect to graduate this summer. Umit should finish in 2002. Luke, Rajeev, and Gregor are first-year graduate students. Amit Singhal and

Rodrigo Carceroni, both computer vision students, have worked part-time on applications development.

For the second year, Cashmere was used as the platform for a group project in our first-year "immigration" course for graduate students. It also formed the bases of the individual projects of Luke Chen and Rajeev Balasubramonian.

References

[1] C. Amza, A.L. Cox, S. Dwarkadas, L-J. Jin, K. Rajamani, and W. Zwaenepoel. Adaptive protocols for software distributed shared memory. *Proceedings of the IEEE*, March 1999.

[2] R. L. Carceroni, W. Meira, R. Stets, and S. Dwarkadas. Evaluating the trade-offs in the parallelization of probabilistic search algorithms. In *Proceedings of the 9th Brazilian Symposium on Computer Architecture and High Performance Processing*, October 1997.

[3] S. Dwarkadas, K. Gharachorloo, L. Kontothanassis, D. Scales, M. L. Scott, and R. Stets. Comparative evaluation of fine- and coarse-grain approaches for software distributed shared memory. In *Proceedings of the Fifth High Performance Computer Architecture Symposium*, pages 260–269, January 1999.

[4] S. Dwarkadas, N. Hardavellas, L. Kontothanassis, R. Nikhil, and R. Stets. Cashmere-VLM: Remote memory paging for software distributed shared memory. In *International Parallel Processing Symposium*, April 1999.

[5] S. Dwarkadas, H. Lu, A. L. Cox, R. Rajamony, and W. Zwaenepoel. Combining compile-time and run-time support for efficient software distributed shared memory. *Proceedings of the IEEE*, March 1999.

[6] G. C. Hunt and M. L. Scott. A guided tour of the Coign automatic distributed partitioning system. In *Proceedings, Enterprise Distributed Object Computing '98*, San Diego, CA, November 1998. Also MSR-TR-98-32, Microsoft Research Laboratory, 1998.

[7] G. C. Hunt and M. L. Scott. The Coign automatic distributed partitioning system. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, LA, February 1999.

[8] G. C. Hunt and M. L. Scott. Intercepting and instrumenting COM applications. In *Proceedings of the Fifth Conference on Object-Oriented Technologies and Systems*, San Diego, CA, May 1999.

[9] S. Ioannidis and S. Dwarkadas. Compiler and run-time support for adaptive load balancing in software distributed shared memory systems. In *Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, pages 107–122, May 1998.

[10] S. Ioannidis, U. Rencuzogullari, R. Stets, and S. Dwarkadas. CRAUL: Compiler and run-time integration for adaptation under load. *Journal of Scientific Programming*, Spring 1999.

[11] S. Parthasarathy and S. Dwarkadas. InterAct: Virtual sharing for interactive client-server applications. In *Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, May 1998.

[12] S. Parthasarathy, R. Subramonian, and R. Venkata. Generalized discretization for summarization and classification. In *3rd International Conference on the Practical Applications of Discovery and Datamining (PADD98)*, March 1998.

[13] S. Parthasarathy, M. J. Zaki, and W. Li. Memory placement techniques for parallel association mining. In *4th International Conference on Knowledge Discovery and Data Mining (KDD)*, August 1998.

[14] S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and interactive sequence mining. Submitted for publication, November 1998.

[15] R. Stets. The effect of network total order and remote-write capability on network-based shared memory computing. Technical Report 711, Computer Science Department, University of Rochester, February 1999.

[16] R. Stets, G. C. Hunt, and M. L. Scott. Component-based operating system APIs: A versioning and distributed resource solution. Submitted for publication, December 1998.

[17] R. Subramonian and S. Parthasarathy. A framework for distributed data mining. In *International Workshop on Distributed Data Mining (with KDD98)*, August 1998.

[18] M.J. Zaki, S. Parthasarathy, and W. Li. A localized algorithm for parallel association mining. *Journal of Parallel and Distributed Computing*, 43(2):156–162, June 1997.