JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 29, 105–107 (1995)

# Special Issue on Distributed Shared Memory Systems

## Guest Editors' Introduction

NIAN-FENG TZENG* AND PEN-CHUNG YEW†

*Center for Advanced Computer Studies, University of Southwestern Louisiana, P.O. Box 44330, Lafayette, Louisiana 70504-4330; and
†Department of Computer Science, University of Minnesota, 200 Union Street SE, Minneapolis, Minnesota 55455-0159

Message-passing distributed-memory multiprocessors can be provided with a layer of shared-memory abstraction so that programmers can envision a globally shared address space. In such a globally shared address space, programmers can carry out data references through read and write operations in the same way as they fetch their local memory—a much more intuitive programming paradigm. This relieves programmers from taking care of data movement explicitly using communication primitives and of data partition specific to each machine. As a result, distributed shared memory (DSM) systems could overcome major obstacles to the widespread use of distributed-memory multiprocessors, i.e., poor programmability and portability, while retaining the attractive features of low cost and good scalability common to distributed-memory machines. In other words, a DSM system allows a natural and portable programming model on distributed-memory machines, making it possible to construct a relatively inexpensive and scalable parallel system on which programmers can develop parallel application code. The code developed for a DSM system, like that for a shared-memory machine, is often much shorter and more comprehensible than an equivalent program based on message-passing.

Due to its potential advantages, DSM has been an active research area for the past decade, with many prototype systems implemented and demonstrated [2]. Those systems have been developed on Ethernet- or ring-connected workstations or on hypercube- or mesh-based message-passing machines. While some implemented prototypes have shown that DSM may achieve performance comparable to that of a loosely coupled distributed system [1, 3], challenges in building efficient DSM systems for a wide range of applications still remain. Memory consistency in the DSM system must be enforced, and this consistency enforcement can be accomplished using hardware, software, or a combination of the two. Hardware consistency implementation is similar to the cache coherence design found in non-distributed shared-memory multiprocessors, and it is typically expensive and inflexible. To the contrary, software coherence implementation tends to result in long access latency and high communication overhead due to consistency enforcement traffic. Several important issues related to DSM implementation are to be resolved before the DSM system delivers its full promise and gains widespread acceptance.

This special issue contains six regular papers and three research notes, each addressing certain important DSM design and implementation issues. These papers are selected from many high-quality manuscripts submitted to this special issue. The paper by Dubois, Skeppstedt, and Stenstrom introduces a classification of the components of the miss rate and data traffic in message-passing shared-memory systems with hardware support for coherence. They define the set of essential misses and essential traffic, referring respectively to the minimum set of misses necessary and to the smallest amount of traffic, for a correct execution under a given coherence protocol. Their classification scheme is applied to three selected benchmarks to compare degrees of overhead created by different hardware coherence mechanisms.

The next two papers deal with software-based coherence implementation. Keleher, Cox, Dwarkadas, and Zwaenepoel examine three software protocols for release consistency. Release consistency allows data communication to be aggregated and also allows multiple writers to modify a single page at the same time. The protocols are implemented on a set of DECstations connected by an ATM network to compare their performance. Communication overhead and network bandwidth are varied using a parallel execution-driven simulator to explore performance trade-offs. It is found that the relative performance of the protocols is dependent on the performance of the network, processor, and communication network.

Sandhu investigates a family of strategies for managing cache coherence dynamically through software, using an abstraction for shared data called Shared Regions (SR). An SR is a unit of sharing in the program and is identified by the application through a set of annotations. The SR annotations can be inserted into the program according to only local information without regard to the global behavior of the program, making it possible for future compilers to insert annotations automatically. The SR approach offers the ability to manage data dynamically according to the granularity of the data shared within the application.



0743-7315/95 $12.00

Through detailed execution-driven simulation, Sandhu shows that the performance of the best SR strategy for each application is comparable to, if not better than, that of a representative hardware coherence mechanism.

Two subsequent papers consider software-based coherence implementation, with some hardware support. Petersen and Li describe techniques using virtual memory management software to enforce coherence, after any potential inconsistency is detected by virtual memory translation hardware existing at each processor. It also requires the use of the memory-mapped network interfaces that support a globally shared physical address space. Memory references which might cause inconsistency are trapped by the virtual memory translation hardware and handled by virtual memory management software. They present the design trade-off between the simplicity and the benefit from increased software overhead. Their evaluation is based on trace driven simulations of several scientific applications. Next, a software coherence protocol for non-cache-coherent, non-uniform memory access (NCC-NUMA) machines is treated by Kontothanassis and Scott. Like Petersen and Li's techniques, this coherence protocol requires memory-mapped network interfaces to support a global address space, which is exploited in three specific ways. A simulation study is conducted for evaluating the performance of the proposed software coherence protocol using an application suite. The experimental results suggest that software coherence on NCC-NUMA machines is a cost-effective approach.

The single program multiple data (SPMD) execution model is a popular model for exploiting parallelism. Under the SPMD model, there is only one fork operation at the beginning and one join operation at the end of the program, with other synchronization points specified explicitly when cross-processor dependences exist. O'Boyle, Kervella, and Bodin present an algorithm for synchronization placement, with an attempt to reduce the amount of synchronization in DSM systems. Their algorithm places barrier synchronization optimally for a given program, based on accurate data dependence and scheduling information. The algorithm has been implemented in an experimental compiler and its initial results are presented.

Checkpointing and rollback techniques can be used in DSM systems to allow continued computation despite the temporary failure of one or multiple components. Janssens and Fuchs introduce the design of an independent checkpointing method for DSM systems with reduced error-free and rollback overhead by taking advantage of some specific properties of DSM systems. They show that dependency in their recovery method can be derived from the traditional message-passing model and that in-transit messages can be recovered without the use of logging. Their method further reduces the frequency of dependency when applied to a DSM system with lazy release consistency.

Implementing a DSM system on a distributed memory machine could incur high communication overhead resulting from a large amount of messages needed to ensure consistency. To reduce this communication overhead, the Munin DSM system presented by Carter incorporates the use of multiple consistency protocols and the support for multiple concurrent writers. The major data structures, the protocols for memory consistency, and the lessons learned from the prototype implementation of Munin are presented in the paper.

Agarwala and Das present the design and the implementation of a shared virtual memory (SVM) system on the nCUBE/2 machine. They improve the algorithm for maintaining coherency to reduce the number of internode messages. Timing analysis is conducted for the feasibility study, and four parallel programs are tested on their SVM implementation. Experimental results show linear speedup for parallel program execution on the SVM system.

## ACKNOWLEDGMENTS

We thank all the authors and referees who contributed to this special issue. Numerous referees had done superb reviews by providing helpful and detailed reports in a short period of time. A couple of submissions which were unable to be included in this special issue are forwarded to future regular issues for consideration after revisions. Finally, we are deeply grateful to Professor Kai Hwang for his constant encouragement, advice, and support in making this special issue possible.

## REFERENCES

1. Li, K., and Hudak, P. Memory coherence in shared virtual memory systems. *ACM Trans. Comput. Systems* **7** (Nov. 1989), 321–359.
2. Nitzberg, B., and Lo, V. Distributed shared memory: A survey of issues and algorithms. *IEEE Comput.* **24** (Aug. 1991), 52–60.
3. Zhou, S., Stumm, M., and McInerney, T. Extending distributed shared memory to heterogeneous environments. *Proc. 10th Int'l Conf. Distributed Computing Systems,* May 1990, pp. 30–37.

## SPECIAL ISSUE REFEREE LIST

Those who served as reviewers for the Special Issue on Distributed Shared Memory Systems are listed below.

S. A. Abraham
S. V. Adve
M. Ahamad
R. Alverson
J. M. Anderson
N. Ansari
J.-L. Baer
N. Bagherzadeh
L. N. Bhuyan
J. B. Carter
J. M. Chapin
V. Chaudhary
H.-A. Choi
M. Choy
C. R. Das
M. Dubois
R. Eigenmann
C. Ellis
E. N. Elnozahy
Dror Feitelson
B. D. Fleisch
S. Fu at
K. Gharachorloo
R. Govindarajan

J. C. Harden
R. Hyde
B. Janssens
S. Kim
C.-T. King
U. Kremer
T. Kumagai
L. K. John
S. Kyo
J. Larus
J. Li
Z. Li
J.-C. Liu
V. M. Lo
E. Ma
M. Mizuno
B. Nitzbery
K. Olukotun
D. Palermo
D. K. Panda
K. Petersen
P. Petersen
J. S. Plank
M. J. Quinn
R. W. Quong
C. S. Raghavendra
M. Ramachandran
U. Ramachandran
M. K. Ravishankar
M. Raynal
A. Ricardo
T. H. Romer
H. S. Sandhu
D. J. Scales
K. Schwan
M. L. Scott
P. Stenstrom
E. Stoltz
M. Stumm
C. B. Stunkel
V. S. Sunderam
J. Torrellas
D. M. Tullsen
N. H. Vaidya
A. Varma
W.-D. Weber
J. L. Welch
J. Wu
K.-L. Wu
H. Xu
Q. Yang
T. Yang
D. Yeung
X. Zhang
Y. Zhu

---

NIAN-FENG TZENG received the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign in 1986. He is currently an associate professor in the Center for Advanced Computer Studies at the University of Southwestern Louisiana, Lafayette. From 1986 to 1987, he was a member of the technical staff at AT&T Bell Laboratories, Columbus, OH. Dr. Tzeng is the recipient of the Outstanding Paper Award from the 10th International Conference on Distributed Computing Systems, May 1990. He is on the editorial board of the *IEEE Transactions on Computers*, has served on the program committees of several conferences, and is a Distinguished Visitor of the IEEE Computer Society. His research interests include parallel and distributed processing, high-performance computer systems, fault-tolerant computing, and high-speed networking.

PEN-CHUNG YEW received the Ph.D. in computer science from the University of Illinois at Urbana–Champaign in 1981. He has been a professor in the Department of Computer Science at the University of Minnesota since 1994. Before that, he was an associate director of the Center for Supercomputing Research and Development at the University of Illinois. He was also on the faculty of the Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Illinois at Urbana–Champaign. From 1991 to 1992, he served as the program director of the Microelectronic Systems Architecture Program in the Division of Microelectronic Information Processing Systems at the National Science Foundation, Washington, DC. Dr. Yew is on the advisory committees of the IEEE Technical Committee on Computer Architecture and the IEEE Technical Committee on Parallel Processing. He has served on the program committees of various conferences. He is currently on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems* and is a subject-area editor of the *Journal of Parallel and Distributed Computing*. He was a Distinguished Visitor of the IEEE Computer Society from 1990 to 1993. He also served as Program Co-Chairman of the 1990 International Conference on Parallel Processing and was General Co-Chairman of the 1994 International Symposium on Computer Architecture. His research interests include high-performance multiprocessor system design, parallelizing compilers, computer architecture, and performance evaluation.