Shared Memory Multiprocessing

edited by
Norihisa Suzuki

The MIT Press
Cambridge, Massachusetts
London, England

This book was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Shared memory multiprocessing / edited by Norihisa Suzuki.
p. cm.
Includes bibliographical references.
ISBN 0-262-19322-1
1. Multiprocessors. 2. Memory management (Computer science) I. Suzuki, Norihisa, 1946–
QA76.5.S4445 1992
004'.35—dc20 92-23489
CIP

Shared Memory Multiprocessing

Preface

The papers included in this book are selected from those presented at the International Symposium on Shared Memory Multiprocessing, which was held in Tokyo from April 2 to April 4 in 1991 under the sponsorship of the Information Processing Society of Japan. It was the first meeting to focus on shared memory multiprocessing, and I am happy to report that it was very successful, with excellent papers and an enthusiastic audience.

Shared memory multiprocessors have been in existence for quite some time. However, the invention of snoop cache in the early 1980's and the subsequent explosion in the computational power of microprocessors and in the density of ASIC chips made it possible to build shared memory multiprocessors economically and reliably. Since then, not only have research prototypes flourished, but many commercial products have also been introduced. Now the first decade is drawing to a close, and it was thus a very appropriate time to hold the conference.

If we look at the commercial market place, not only have many commercial products been introduced with considerable economic success, but also many of the commercial microprocessors and peripheral chips now contain mechanisms to support shared memory multiprocessors.

Mechanisms for supporting shared memory multiprocessing at the operating system level have been studied well, and they have been introduced into proprietary operating systems. Now, they are being incorporated into widely available operating systems. Compiling techniques are also progressing, and parallelizing FORTRAN and C compilers are now available on most systems.

On the research side, so-called scalable architecture is being hotly pursued and is the major theme of this book. Investigations are being made to see whether we can construct distributed memory multiprocessors with hardware support to realize a shared memory model. This architecture is considered to be able to support hundreds of microprocessors without performance degradation. The speed of microprocessors is also increasing very rapidly, so it is a big challenge to connect them efficiently and package them economically.

I believe that this book gives a clear perspective on the state of the art in shared memory multiprocessors.

I would like to thank the organizations that supported this conference financially: the International Information Science Foundation, IBM Japan, and NEC.

Finally, I would like to thank the following people who worked on organizing the symposium:

General Chair:	Takashi Masuda
Treasurer:	Kazuya Tago
Local arrangement:	Yoshikatsu Tada
Program committee:	Tilak Agerwala
	Forest Baskett
	Ed Clarke
	David Gifford
	Jim Goodman
	John Hennesy
	Paul Hilfinger
	Nobuhiko Koike
	Norihisa Suzuki (Chairman)
	Chuck Thacker
	Shinji Tomita
	Akinori Yonezawa

Tokyo, Japan
October 1991

Norihisa Suzuki

The MIT Press, with Peter Denning as general consulting editor, publishes computer science books in the following series:

ACL-MIT Press Series in Natural Language Processing
Aravind K. Joshi, Karen Sparck Jones, and Mark Y. Liberman, editors

ACM Doctoral Dissertation Award and Distinguished Dissertation Series

Artificial Intelligence
Patrick Winston, founding editor
J. Michael Brady, Daniel G. Bobrow, and Randall Davis, editors

Charles Babbage Institute Reprint Series for the History of Computing
Martin Campbell-Kelly, editor

Computer Systems
Herb Schwetman, editor

Explorations with Logo
E. Paul Goldenberg, editor

Foundations of Computing
Michael Garey and Albert Meyer, editors

History of Computing
I. Bernard Cohen and William Aspray, editors

Logic Programming
Ehud Shapiro, editor; Fernando Pereira, Koichi Furukawa, Jean-Louis Lassez, and David H. D. Warren, associate editors

The MIT Press Electrical Engineering and Computer Science Series

Research Monographs in Parallel and Distributed Processing
Christopher Jesshope and David Klappholz, editors

Scientific and Engineering Computation
Janusz Kowalik, editor

Technical Communication and Information Systems
Edward Barrett, editor

Contents

Preface ix

I EXPERIENCE

1 Experience with the Firefly Multiprocessor Workstation 3
Susan Owicki

2 Design and Evaluation of Snoop-Cache-Based Multiprocessor, TOP-1 25
Shigenori Shimizu, Nobuyuki Oba, Atsushi Moriwaki, and Takeo Nakada

3 Symbolic Computation Algorithms on Shared Memory Multiprocessors 53
E. M. Clarke, S. Kimura, D. E. Long, S. Michaylov, S. A. Schwab, and J. P. Vidal

4 Experimental Evaluation of Algorithmic Performance on Two Shared Memory Multiprocessors 81
Anand Sivasubramaniam, Gautam Shah, Joonwon Lee, Umakishore Ramachandran, and H. Venkateswaran

II CACHE COHERENCY

5 Formal Verification of the Gigamax Cache Consistency Protocol 111
Kenneth McMillan and James Schwalbe

6 An Evaluation of Cache Coherence Protocols for Multiprocessors 135
Sandra Johnson Baylor, Kevin P. McAuliffe, and Bharat D. Rathi

7 **KRPP: The Kyushu University Reconfigurable Parallel Processor: Cache Architecture and Cache Coherence Schemes** 165
Kazuaki Murakami, Shin-ichiro Mori, Eiji Iwata, Akira Fukuda, and Shinji Tomita

III SOFTWARE SYSTEM

8 **MUSTARD: A Multiprocessor UNIX for Embedded Real-Time Systems** 195
Shuichi Hiroya, Takeshi Momoi, and Katsutoshi Nihei

9 **An Empirical Investigation of the Effectiveness and Limitations of Automatic Parallelization** 213
Jaswinder Pal Singh and John L. Hennessy

10 **Fine-Grain Loop Scheduling for MIMD Machines** 241
Carrie J. Brownhill, Ki-chang Kim, and Alexandru Nicolau

11 **Restructuring a Parallel Simulator to Improve Cache Behavior** 261
David R. Cheriton, Hendrik A. Goosen, and Philip Machanick

12 **A Replay Mechanism for Mostly Functional Parallel Programs** 287
Robert H. Halstead, Jr., and David A. Kranz

13 **Abstracting Data-Representation and Partitioning-Scheduling in Parallel Programs** 315
Gail A. Alverson and David Notkin

IV SCALABLE SHARED MEMORY MULTIPROCESSOR

14 Latency Tolerance through Multithreading in Large-Scale Multiprocessors 341
Kiyoshi Kurihara, David Chaiken, and Anant Agarwal

15 Cenju: A Multiprocessor System with a Distributed Shared Memory Scheme for Modular Circuit Simulation 363
Toshiyuki Nakata, Norio Tanabe, Nobuki Kajihara, Satoshi Matsushita, Hiromi Onozuka, Yoshihiro Asano, and Nobuhiko Koike

16 Overview and Status of the Stanford DASH Multiprocessor 391
Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, and John Hennessy

17 An Analysis of Shared-Memory Synchronization Mechanisms 407
Philip J. Woest and James R. Goodman

18 A Cache Coherence Mechanism for Scalable, Shared-Memory Multiprocessors 437
Steven Scott

19 Dynamic Pointer Allocation for Scalable Cache Coherence Directories 463
Richard Simoni and Mark Horowitz

20 Fault-Tolerant Design for Multistage Routing Networks 483
André DeHon, Thomas Knight Jr., and Henry Minsky

List of Contributors 505

I EXPERIENCE

1 Experience with the Firefly Multiprocessor Workstation

Susan Owicki

Commercial multiprocessors are used successfully for a range of applications, including intensive numeric computations, time-sharing, and shared servers. The value of multiprocessing in a single-user workstation is not so obvious, especially in an environment where numeric problems do not dominate. The Digital Equipment Corporation, Systems Research Center has had several years of experience using the five-processor Firefly workstation in such an environment. This report is an initial assessment of how much is gained from multiprocessing on the Firefly.

Reported here are measurements of speedup and utilization for a variety of programs. They illustrate four sources of concurrency: between independent tasks, within a server, between client and server, and within an application. The nature of the parallelism in each example is explored, as well as the factors, if any, that constrain multiprocessing. The examples cover a wide range of multiprocessing, with speedups on a five-processor machine varying from slightly over 1 to nearly 6. Most uses derive most of their speedup from two or three processors, but there are important applications that can effectively use five or more.

1.1 Introduction

Commercial multiprocessors are used successfully for a range of applications, including intensive numeric computations, time-sharing, and shared servers. In these uses, there is abundant scope for multiprocessing in handling simultaneous requests from separate users or in single-user computations where there is substantial concurrency in the structure of the problems.

The value of multiprocessing in a single-user workstation is not so obvious, especially in an environment where numeric problems do not dominate. Can other applications besides scientific computation exploit multiple processors? Are multiple users essential to generate a reasonable workload for the system?

For several years, the Firefly multiprocessor workstation [9] has been the primary source of computing at the Digital Equipment Corporation, Systems Research Center (SRC). Software for the Firefly spans a wide range of systems and applications programs. Most Firefly programs are written in an extension of Modula 2 [12] called Modula 2+ [7], which provides threads and synchronization primitives for concurrent programming. The Firefly workload does not include the sort of lengthy numeric

calculations that have traditionally benefited from concurrency. Nevertheless, much of the software has been designed to take advantage of multiprocessing. Thus there has been substantial experience at SRC with using multiprocessor workstations for non-numeric computation, and it seems appropriate now to assess their value. To do so, a number of instances of multiprocessing on the Firefly were examined. This report gives measures of concurrency for these examples, and describes the sources and limits of their parallelism.

The Firefly used for these measurements has five 1-MIP MicroVAX processors, so it is called a 5-MIP machine. In actual use, though, the Firefly has less computational power than a 5-MIP uniprocessor. This is partly because some of the software was originally written for a uniprocessor. But even code written for the Firefly seldom exploits all the concurrency that the processors provide. There are many reasons: some problems have limited parallelism, sometimes the overhead of concurrency is too high, sometimes another part of the machine is a bottleneck, and sometimes the implementor chose to avoid the complexity of concurrent programming.

However, the Firefly doesn't have to provide a full 5 MIPS of computing power to be cost-effective, since it is generally cheaper to build a multiprocessor than a uniprocessor with the same MIPS rating. Building a multiprocessor with, say, three to thirty processors may not cost a great deal more than building a uniprocessor with the same CPU. Thus the multiprocessor may be economically attractive even if its processors are not always fully utilized. The benefit gained from greater computing power must be weighed against the increased cost of the multiprocessor.

This report is concerned with assessing the benefit side of the cost-benefit equation. Benefit is estimated using the standard metrics speed-up and processor utilization. These metrics, which are discussed in Section 2, are less than ideal, but they do give some feeling for the degree of success in exploiting multiprocessing.

Four sources of concurrency were identified in day-to-day workstation activities:

- single-user time-sharing: concurrency between independent tasks. A user may undertake several tasks in parallel, such as editing or reading mail while a compilation is in progress.

- concurrency within a server. The window system, the file system, and other basic services are implemented with algorithms that use multiprocessing.
- concurrency between client and server. Sometimes a server can return an immediate answer to a request, then compute in parallel with its client to complete processing the request.
- concurrency within an application: some application programs are coded with multiple threads for performance.

Note that the first three sources of concurrency are available in all uses of the workstation, without requiring an application programmer to write multi-threaded code. So multiprocessors in a workstation can be useful even when running applications that do not attempt to exploit concurrency.

Sections 3 to 6 contain speedup and utilization data for a number of examples from each of the areas above. All the measurements were taken on a 5-processor, 16 Megabyte Firefly, unless otherwise noted.

1.2 Metrics

Parallel sorting will be used as an example to illustrate speedup and processor utilization. Figure 1.1 gives graphs of speedup and utilization for a quicksort program [11] working on an array of 10000 integers. The algorithm sequentially partitions the array and then recursively applies quicksort in parallel to the two subarrays.

The speedup reported in Figure 1.1a is defined in the conventional way: speedup for n processors is

$$S(n) = T(1)/T(n), \tag{1.1}$$

where $T(k)$ is the execution time when the program is run using k processors. Speedup is determined using an option of Taos, the Firefly operating system, that restricts the set of processors available to the scheduler. Thus $T(k)$ is measured by disabling all but k processors and noting the elapsed time to execute the program. Since elapsed time can fluctuate due to other activities in the system, $T(k)$ was taken as the median elapsed time of three to five runs.

The dotted line in Figure 1.1a represents the theoretical "perfect" speedup. For two and three processors, Quicksort has a nearly optimal

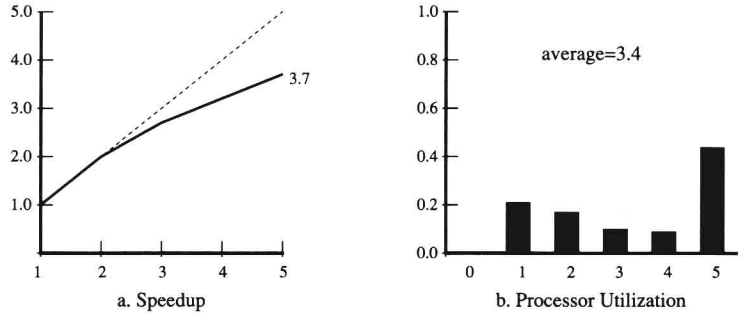

Figure 1.1
Quicksort of 10000 integers.

speedup. There is some additional benefit from the fourth and fifth processors, but it is less significant. The amount of parallel computation is limited by the structure of the algorithm: during the initial partitioning, for example, only one thread is active.

Processor utilization for a five-processor run of Quicksort is shown in Figure 1.1b. The height of the kth bar represents the fraction of the time when exactly k processors were busy. There was essentially no time when all processors were idle, which is to be expected in a compute-bound task like sorting an array. All five processors were busy for more than 40% of the time. This is somewhat surprising, given that the speedup figures for four to five processors were not impressive. With five processors, the highly parallel parts of the run keep all the processors busy, leaving a substantial amount of time when only one processor is busy. With fewer processors, sequential segments in one partition are more likely to overlap parallel segments of another, and each processor is busy a larger fraction of the time. The average processor utilization on a five-processor Firefly was 3.4.

Processor utilization was measured by instrumenting the operating system to record the amount of time spent with k processors busy. Once this instrumentation has been done, measuring utilization is much easier than measuring speedup, because speedup requires multiple runs for each value of n.

There is a correspondence between speedup and average processor utilization. If a program does the same amount of work when run with 1 processor or with n processors, its speedup and average utilization