

SCALABLE SHARED-MEMORY MULTIPROCESSING

DANIEL E. LENOSKI



WOLF-DIETRICH WEBER

S C A L A B L E S H A R E D - M E M O R Y
M U L T I P R O C E S S I N G

This page intentionally left blank

Scalable Shared-Memory Multiprocessing

**Daniel E. Lenoski
Wolf-Dietrich Weber**



**Morgan Kaufmann Publishers
San Francisco, California**

Executive Editor Bruce Spatz
Production Manager Yonie Overton
Assistant Production Editor Julie Pabst
Cover Design Feroza Unvala
Text Design Jim Love/Studio Arno
Composition Ed Sznyter
Copyeditor Ken DellaPenta
Proofreader Liz Sizensky
Printer Quebecor Printing

Morgan Kaufmann Publishers, Inc.
Editorial and Sales Office
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205
USA
Telephone 415/392-2665
Facsimile 415/982-2665
Internet mkp@mfp.com

© 1995 by Morgan Kaufmann Publishers, Inc.

All rights reserved
Printed in the United States of America

99 98 97 96 95 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, recording or otherwise—withou the prior written permission of the publisher.

Library of Congress Cataloging-in-Publication Data

Lenoski, Daniel E.

Scalable shared-memory multiprocessing / Daniel E. Lenoski, Wolf-Dietrich Weber.

p. cm.

Includes bibliographical references and index.

ISBN 1-55860-315-8

1. Multiprocessors. 2. Memory management (Computer science)

I. Weber, Wolf-Dietrich. II. Title.

QA76.5.L457 1995

004'.35--dc20

**95-19615
CIP**

*To Karen and Véronique
and Veronica, Steven, and Nicole, too*

This page intentionally left blank

Foreword

by
John L. Hennessy

The idea of multiprocessors dates back to the first electronic computers. As early as the 1950s, MPs were advanced as a technique both to increase reliability and to improve performance. Early efforts concentrated on building small-scale machines, which avoided a number of problems. Two exceptions were the C.mmp and Cm* machines built at Carnegie-Mellon University in the 1970s. These systems included many of the features found in more recent large-scale machines. However, simpler structures, adequate for the smaller systems being built at the time, became more popular.

In the early 1980s, the first commercially successful multiprocessors became available. Almost all of these designs were bus-based, shared-memory machines. Their development was enabled by two key factors: the incredible price-performance advantages of microprocessors and the extensive use of caches in microprocessor-based systems. These technological factors made it possible to put multiple processors on a bus with a single memory, relying on caching to reduce the memory bandwidth demands of the processors to an acceptable level. Coherency among the caches was maintained through a snoopy protocol, which, while simple, employed broadcast, making these systems fundamentally unscalable.

In the mid 1980s, the interest in scalable multiprocessors grew quickly. The bus-based, cache-coherent, shared-memory machines were clearly not scalable, so designers focused on two other approaches. One approach gave up on coherence but maintained shared memory. Three important designs of this genre were the BBN Butterfly, the IBM RP3, and the NYU Ultracomputer. As in earlier experiments, the absence of caching significantly increased the cost of shared access—each shared access had to traverse the network—and made it hard to obtain good performance on such machines. The Cray T3D is the major machine in this camp at the present. In practice, it appears that such machines are most effectively used as message-passing machines, with the shared address space providing a convenient naming mechanism.

The second approach to scalability relied on message passing rather than shared memory. The problem of memory latency on the non-coherent shared-memory machines, together with the impossibility of scaling the snoopy approach, led some designers to conclude that message passing was the only viable approach for scalable multiprocessors. However, message-passing machines have proved very difficult to program.

This book explores a new class of machines supporting both cache-coherent shared memory and scalability to a large number of processors. The first part of the book looks at the general concepts involved in building scalable multiprocessors, especially scalable shared-memory machines. The authors begin with a description of the challenges in designing scalable architectures. They then discuss the trade-offs between shared memory and message passing, as well as the challenges of scalable cache coherency. In considering the scalability of performance of any particular scheme for shared memory, the results are substantially affected by the communication and sharing characteristics in the applications. Chapter 2 presents extensive data on this important topic. Designing a large-scale, shared-memory machine requires attention to minimizing memory latency and hiding that latency whenever possible. These two topics provide the key focus of Chapter 3. Chapter 4 examines a variety of design alternatives that arise in the actual implementation of such systems. These first four chapters are valuable to both designers and programmers of scalable machines who wish to understand the performance of various architectures, as well as the variety of design alternatives. Finally, Chapter 5 surveys historical and proposed scalable shared-memory machines, including the Stanford DASH multiprocessor.

The DASH project began in 1987 to explore whether it was possible to build scalable shared-memory multiprocessors that efficiently support cache coherence. By using a distributed-memory model, explored in both the Cm* and RP3 projects, DASH allows memory bandwidth to be scaled, unlike that in a bus-based system. To implement scalable cache coherence, DASH uses a distributed directory protocol. In 1991, DASH became the first operational shared-memory multiprocessor with a scalable hardware cache coherence mechanism. Kendall Square Research and Convex became the first companies to market scalable shared-memory multiprocessors commercially. Such machines have become known as Distributed Shared Memory (DSM). The interest in DSM architectures is growing quickly, both because of the scalability advantages and because of the increasing difficulty of building bus-based multiprocessors as processors increase in speed.

The second part of this book is a retrospective examination of the design of the DASH multiprocessor. It provides not only a detailed description of the design, but also a rationale for the design and an exposition of the challenges arising in any such machine. For people interested in scalable machines, this portion of the book is a must-read. Unlike the documentation of industrial projects, which rarely criticizes their products, this book supplies many frank assessments of the strengths and weaknesses of the DASH design, providing readers with extensive input on how the design might be improved upon in the future.

Although the largest multiprocessors seem to be moving toward shared memory and away from message passing, it remains a point of debate whether cache coherency will play a major role in very large machines. The final section of the book explains how a cache-coherent design like DASH could be extended to the range of thousands of processors and TERAOps of performance. In addition to examining the structure of such a design, this section estimates the performance that might be achievable. The analysis shows that building large-scale parallel machines does not mean abandoning the efficient and easy-to-use shared-memory programming model.

I found this book extremely interesting. The first portion of the book lays a solid foundation, showing why it is possible to build a shared-memory scalable machine. Then, in the second portion, the description of the landmark DASH machine by one of its original designers is both fascinating and informative. Finally, the third portion gives a glimpse at the not-so-distant future of scalable shared-memory multiprocessors. Such scalable, cache-coherent machines appear to be major candidates for next-generation multiprocessors. Anyone involved in the design of such machines, in software development for them, or even in using such machines will find this book useful and enlightening.

This page intentionally left blank

Contents

Foreword	vii
Preface	xvii

P A R T 1 **GENERAL CONCEPTS**

CHAPTER 1	Multiprocessing and Scalability	3
1.1	Multiprocessor Architecture	6
1.1.1	Single versus Multiple Instruction Streams	7
1.1.2	Message-Passing versus Shared-Memory Architectures	8
1.2	Cache Coherence	13
1.2.1	Uniprocessor Caches	14
1.2.2	Multiprocessor Caches	16
1.3	Scalability	20
1.3.1	Scalable Interconnection Networks	24
1.3.2	Scalable Cache Coherence	31
1.3.3	Scalable I/O	33
1.3.4	Summary of Hardware Architecture Scalability	34
1.3.5	Scalability of Parallel Software	35
1.4	Scaling and Processor Grain Size	37
1.5	Chapter Conclusions	39
CHAPTER 2	Shared-Memory Parallel Programs	41
2.1	Basic Concepts	41
2.2	Parallel Application Set	46
2.2.1	MP3D	48
2.2.2	Water	48
2.2.3	PTHOR	49
2.2.4	LocusRoute	49
2.2.5	Cholesky	49

2.2.6 Barnes-Hut	50
2.3 Simulation Environment	50
2.3.1 Basic Program Characteristics	51
2.4 Parallel Application Execution Model	52
2.5 Parallel Execution under a PRAM Memory Model	53
2.6 Parallel Execution with Shared Data Uncached	55
2.7 Parallel Execution with Shared Data Cached	56
2.8 Summary of Results with Different Memory System Models	58
2.9 Communication Behavior of Parallel Applications	59
2.10 Communication-to-Computation Ratios	59
2.11 Invalidation Patterns	62
2.11.1 Classification of Data Objects	62
2.11.2 Average Invalidation Characteristics	64
2.11.3 Basic Invalidation Patterns for Each Application	65
2.11.4 MP3D	67
2.11.5 Water	67
2.11.6 PTHOR	69
2.11.7 LocusRoute	71
2.11.8 Cholesky	73
2.11.9 Barnes-Hut	73
2.11.10 Summary of Individual Invalidation Distributions	76
2.11.11 Effect of Problem Size	76
2.11.12 Effect of Number of Processors	76
2.11.13 Effect of Finite Caches and Replacement Hints	78
2.11.14 Effect of Cache Line Size	80
2.11.15 Invalidation Patterns Summary	83
2.12 Chapter Conclusions	84
CHAPTER 3 System Performance Issues	87
3.1 Memory Latency	88
3.2 Memory Latency Reduction	89
3.2.1 Nonuniform Memory Access (NUMA)	90
3.2.2 Cache-Only Memory Architecture (COMA)	91
3.2.3 Direct Interconnect Networks	93
3.2.4 Hierarchical Access	93
3.2.5 Protocol Optimizations	94
3.2.6 Latency Reduction Summary	95
3.3 Latency Hiding	95
3.3.1 Weak Consistency Models	96
3.3.2 Prefetch	100
3.3.3 Multiple-Context Processors	103
3.3.4 Producer-Initiated Communication	108

3.3.5	Latency Hiding Summary	110
3.4	Memory Bandwidth	111
3.4.1	Hot Spots	112
3.4.2	Synchronization Support	113
3.5	Chapter Conclusions	116
CHAPTER 4	System Implementation	117
4.1	Scalability of System Costs	117
4.1.1	Directory Storage Overhead	119
4.1.2	Sparse Directories	127
4.1.3	Hierarchical Directories	132
4.1.4	Summary of Directory Storage Overhead	133
4.2	Implementation Issues and Design Correctness	134
4.2.1	Unbounded Number of Requests	134
4.2.2	Distributed Memory Operations	136
4.2.3	Request Starvation	139
4.2.4	Error Detection and Fault Tolerance	139
4.2.5	Design Verification	141
4.3	Chapter Conclusions	142
CHAPTER 5	Scalable Shared-Memory Systems	143
5.1	Directory-Based Systems	143
5.1.1	DASH	144
5.1.2	Alewife	144
5.1.3	S3.mp	146
5.1.4	IEEE Scalable Coherent Interface	147
5.1.5	Convex Exemplar	149
5.2	Hierarchical Systems	150
5.2.1	Encore GigaMax	151
5.2.2	ParaDiGM	152
5.2.3	Data Diffusion Machine	154
5.2.4	Kendall Square Research KSR-1 and KSR-2	155
5.3	Reflective Memory Systems	157
5.3.1	Plus	157
5.3.2	Merlin and Sesame	158
5.4	Non-Cache-Coherent Systems	159
5.4.1	NYU Ultracomputer	159
5.4.2	IBM RP3 and BBN TC2000	160
5.4.3	Cray Research T3D	161
5.5	Vector Supercomputer Systems	162
5.5.1	Cray Research Y-MP C90	163
5.5.2	Tera Computer MTA	164
5.6	Virtual Shared-Memory Systems	166

5.6.1	Ivy and Munin/Treadmarks	166
5.6.2	J-Machine	167
5.6.3	MIT/Motorola *T and *T-NG	169
5.7	Chapter Conclusions	170

P A R T 2 **EXPERIENCE WITH DASH**

CHAPTER 6 DASH Prototype System **173**

6.1	System Organization	174
6.1.1	Cluster Organization	175
6.1.2	Directory Logic	177
6.1.3	Interconnection Network	180
6.2	Programmer's Model	181
6.3	Coherence Protocol	184
6.3.1	Nomenclature	185
6.3.2	Basic Memory Operations	187
6.3.3	Prefetch Operations	192
6.3.4	DMA/Uncached Operations	193
6.4	Synchronization Protocol	198
6.4.1	Granting Locks	198
6.4.2	Fetch&Op Variables	200
6.4.3	Fence Operations	200
6.5	Protocol General Exceptions	201
6.6	Chapter Conclusions	202

CHAPTER 7 Prototype Hardware Structures **205**

7.1	Base Cluster Hardware	206
7.1.1	SGI Multiprocessor Bus (MPBUS)	206
7.1.2	SGI CPU Board	207
7.1.3	SGI Memory Board	210
7.1.4	SGI I/O Board	211
7.2	Directory Controller	211
7.3	Reply Controller	218
7.4	Pseudo-CPU	224
7.5	Network and Network Interface	226
7.6	Performance Monitor	229
7.7	Logic Overhead of Directory-Based Coherence	232
7.8	Chapter Conclusions	236

CHAPTER 8	Prototype Performance Analysis	237
8.1	Base Memory Performance	237
8.1.1	Overall Memory System Bandwidth	238
8.1.2	Other Memory Bandwidth Limits	240
8.1.3	Processor Issue Bandwidth and Latency	241
8.1.4	Interprocessor Latency	244
8.1.5	Summary of Memory System Bandwidth and Latency	244
8.2	Parallel Application Performance	246
8.2.1	Application Run-Time Environment	246
8.2.2	Application Speedups	247
8.2.3	Detailed Case Studies	250
8.2.4	Application Speedup Summary	257
8.3	Protocol Effectiveness	260
8.3.1	Base Protocol Features	260
8.3.2	Alternative Memory Operations	264
8.4	Chapter Conclusions	271

P A R T 3

FUTURE TRENDS

CHAPTER 9	TeraDASH	277
9.1	TeraDASH System Organization	277
9.1.1	TeraDASH Cluster Structure	278
9.1.2	Intracluster Operations	280
9.1.3	TeraDASH Mesh Network	283
9.1.4	TeraDASH Directory Structure	284
9.2	TeraDASH Coherence Protocol	286
9.2.1	Required Changes for the Scalable Directory Structure	286
9.2.2	Enhancements for Increased Protocol Robustness	288
9.2.3	Enhancements for Increased Performance	294
9.3	TeraDASH Performance	296
9.3.1	Access Latencies	297
9.3.2	Potential Application Speedup	298
9.4	Chapter Conclusions	303
CHAPTER 10	Conclusions and Future Directions	305
10.1	SSMP Design Conclusions	306
10.2	Current Trends	307
10.3	Future Trends	308

APPENDIX	Multiprocessor Systems	311
References		317
Index		333

This page intentionally left blank

Preface

Scalable shared-memory multiprocessing (SSMP) systems provide the power of hundreds to thousands of high-performance microprocessors all sharing a common address space. SSMP is appealing because it addresses a number of problems with existing parallel systems. Relative to large-scale message-passing machines, SSMP provides a much simpler programming model, which makes the system easier to use. The shared-memory paradigm also reduces communication overhead, making finer-grained work sharing possible. Compared with traditional bus-based shared-memory machines, SSMP does not have the bus bandwidth problem, which limits the number of processors that can be combined in a single system.

SSMP has been a popular research topic since the development of the IBM RP3 and NYU Ultracomputer in the early 1980s. The problem with these early SSMP machines was that the performance of individual processing nodes was limited by memory latency and by the modest performance of the highly integrated processors (i.e., microprocessors) available at the time. The performance of the latest microprocessors has eliminated the internal performance limits, but the challenge of keeping memory latency low enough so that performance scales when the processors are combined remains.

This book describes the architecture and hardware design of SSMP systems that combine high-performance processors with a scalable cache coherence mechanism. Caching helps to keep the effective memory latency low and processor utilization high. The material from the book is drawn from experience with the DASH project at Stanford University. In early 1991, this work resulted in the first operational SSMP machine with a scalable cache coherence mechanism. The prototype machine combines up to 64 RISC processors, with an aggregate performance of 1.6 GIPS and 600 MFLOPS, all sharing a hardware-supported cache-coherent memory address space.

We wrote this book with three audiences in mind: (1) designers and users of commercial SSMP machines (the first generation of SSMP systems such as the Kendall Square Research KSR-1 and Convex Exemplar are already in the market), (2) researchers of parallel machine architecture, and (3) anyone studying advanced computer architecture and parallel processing.

The book is broken into three major parts. Part I introduces general SSMP concepts and the design issues that all SSMP systems face. Part II is a detailed study of the DASH

prototype machine and its performance. Part III concludes with an outline of a very large-scale machine based on the DASH design that could be built with today's state-of-the-art technology. It combines up to 2,000 processors to deliver a peak performance of over one trillion instructions per second and 400 GFLOPS.

The general SSMP concepts presented in Part I are broken into five chapters. Chapter 1 relates the base SSMP design to other major classes of parallel machines. Chapter 2 analyzes the characteristics of a number of parallel applications and their implications for system design. Chapter 3 discusses the important performance issues that every SSMP system must address. Chapter 4 covers design issues related to scalability of the system. Chapter 5 surveys the major SSMP systems that have been proposed and highlights their unique features.

Part II of the book details the design and performance of the DASH prototype machine. Chapter 6 gives an overview of the DASH architecture and a description of the coherence protocol. Chapter 7 describes the detailed hardware design of the system and calculates the overhead of the scalable coherence mechanism quantitatively. Chapter 8 summarizes performance measurements taken from full applications run on the prototype. It also uses stress tests to demonstrate the effectiveness of extensions made to the standard memory protocol.

Part III addresses the future of SSMP systems. Chapter 9 outlines the design of a very large-scale system based on the DASH prototype. It gives solutions to the major problems not addressed in the smaller DASH prototype and an estimate of the performance of the system. Chapter 10 concludes the book with an overview of the trends in SSMP systems and an outlook for this class of system.

The DASH project at Stanford was an exciting research project to which many individuals contributed. We are proud to have been part of such an inspiring group. In particular, we would like to acknowledge the guidance of our advisors, Anoop Gupta and John Hennessy, and the hard work of our DASH hard-core teammates (Jim Laudon, Dave Nakahira, Truman Joe, and Luis Stevens) that made DASH a reality. The project also received contributions from a much wider audience of Stanford faculty (especially Mark Horowitz and Monica Lam) and students (too many to mention). We would also like to acknowledge the generous support from DARPA and Silicon Graphics, without which DASH would not have been possible.

For help with this book, we would first like to thank those who reviewed the manuscript: Rick Bahr (Silicon Graphics), Andreas Nowatzky (Sun), J. P. Singh (Princeton), Jim Smith (University of Wisconsin), Len Widra (Silicon Graphics), and others. Their input greatly improved the presentation and accuracy of the book. We would especially like to thank Jim Smith, whose careful reading and insightful comments led to a major restructuring of the material. The leaders of the projects surveyed in this book also reviewed the manuscript to verify that their projects were discussed accurately. We'd like to thank the following for their input: Anant Agarwal (MIT), Mike Beckerle (Motorola), David Cheriton (Stanford), Bill Dally (MIT), Allan Gottlieb (NYU), Erik Hagersten (Sun), Dave James (Apple), Kai Li (Princeton), Burton Smith (Tera Computer), and Steve Wallach (Convex). We would also like to thank the folks at Morgan Kaufmann for making the book a reality. In