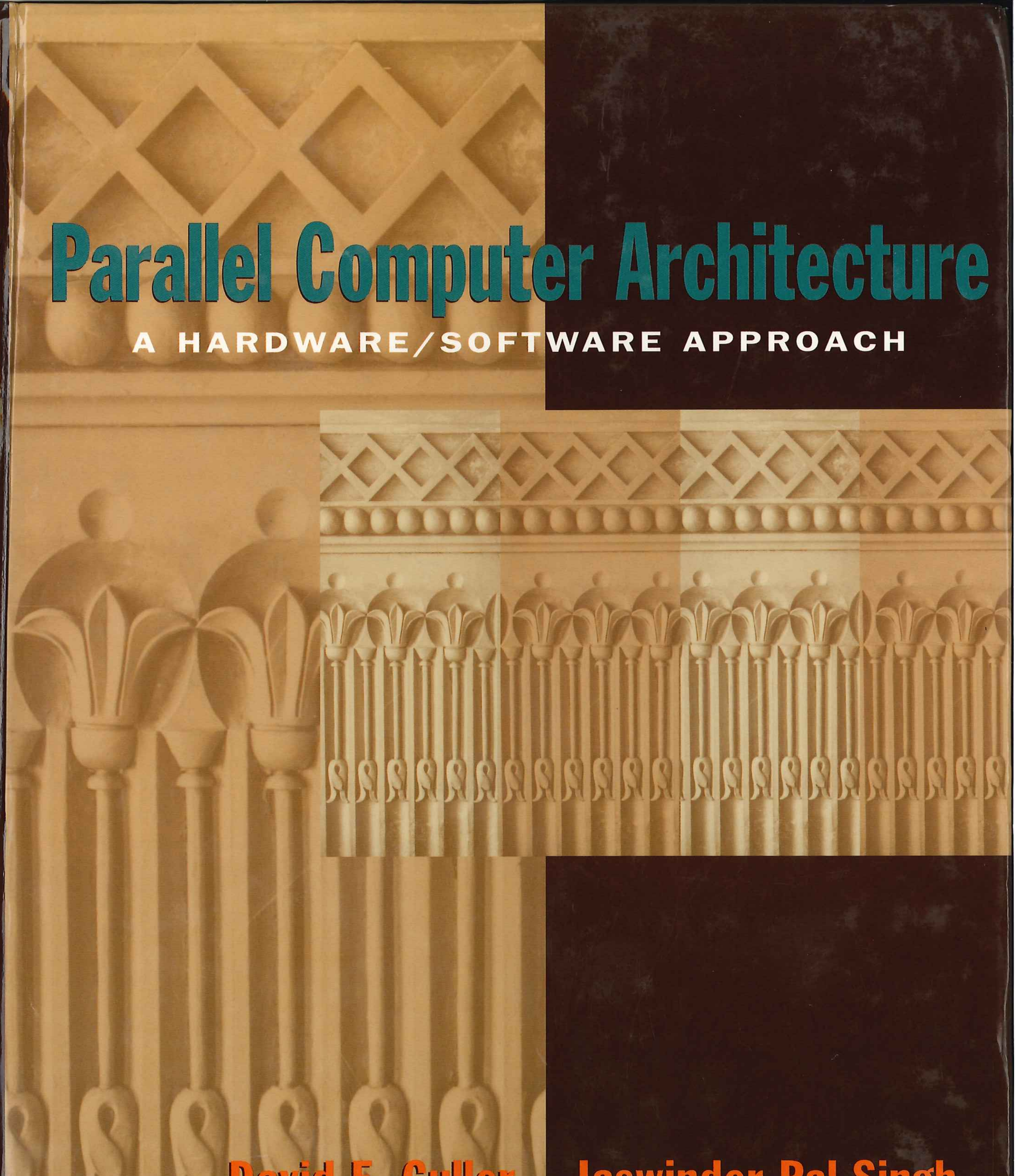
Parallel Computer Architecture

A HARDWARE/SOFTWARE APPROACH



David E. Culler   Jaswinder Pal Singh

Parallel Computer Architecture

A Hardware/Software Approach

David E. Culler Jaswinder Pal Singh

with Anoop Gupta

MORGAN KAUFMANN PUBLISHERS, INC.
San Francisco, California

Senior Editor	Denise E.M. Penrose
Director of Production and Manufacturing	Yonie Overton
Senior Production Editor	Elisabeth Beller
Editorial Coordinator	Meghan Keeffe
Cover Design	Martin Heirakuji Graphic Design
Cover Photo	Image copyright © 1998 PhotoDisc, Inc.
Text Design	Mark Ong, Side by Side Studios
Copyeditor	Jennifer McClain
Proofreaders	Jennifer McClain, Jeff Van Beuren, Ken DellaPenta, Christine Sabooni
Compositor	Nancy Logan
Illustrators	Nancy Logan, Dartmouth Publishing, Inc., Cherie Plumlee
Indexer	Steve Rath
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
Email mkp@mkp.com
WWW http://www.mkp.com

Order toll free 800/745-7323

Advice, Praise, and Errors: Any correspondence related to this publication or intended for the authors should be addressed to the Editorial and Sales Office of Morgan Kaufmann Publishers, Inc., Dept. PCA APE or sent electronically to *pca@mkp.com*. Please report errors by email to *pcabugs@mkp.com*. Please check the errata page at *http://www.mkp.com/pca* to see if the bug has already been reported and fixed.

03 02 01 00 99 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data
Culler, David E.
Parallel computer architecture: a hardware/software approach / David E. Culler, Jaswinder Pal Singh, with Anoop Gupta.
p. cm.
Includes bibliographical references and index.
ISBN 1-55860-343-3
1. Parallel computers. 2. Computer architecture. I. Singh, Jaswinder Pal. II. Gupta, Anoop. III. Title.
QA76.58.C85 1999
004′.35—dc21 98-28034
CIP

enhanced version of it is used in the Sun SparcServer multiprocessors (Catanzaro 1997).

The Dragon protocol consists of four states: exclusive-clean (E), shared-clean (Sc), shared-modified (Sm), and modified (M). Exclusive-clean (or exclusive) has the same meaning and the same motivation as before: only one cache (this cache) has a copy of the block, and it has not been modified (i.e., the main memory is up-to-date). *Shared-clean* means that potentially two or more caches (including this one) have this block, and main memory may or may not be up-to-date. *Shared-modified* means that potentially two or more caches have this block, main memory is not up-to-date, and it is this cache's responsibility to update the main memory at the time this block is replaced from the cache (i.e., this cache is the owner). A block may be in Sm state in only one cache at a time. However, it is quite possible that one cache has the block in Sm state, while others have it in Sc state. Or it may be that no cache has it in Sm state, but some have it in Sc state. This is why, when a cache has the block in Sc state, memory may or may not be up-to-date; it depends on whether some other cache has it in Sm state. M signifies exclusive ownership as before: the block is modified (dirty) and present in this cache alone, main memory is stale, and it is this cache's responsibility to supply the data and to update main memory on replacement. Note that there is no explicit invalid (I) state as in the previous protocols. This is because Dragon is an update-based protocol; the protocol always keeps the blocks in the cache up-to-date, so it is always okay to use the data present in the cache if the tag match succeeds. However, if a block is not present in a cache at all, it can be imagined in a special invalid or not-present state.[4]

The processor requests, bus transactions, and actions for the Dragon protocol are similar to the Illinois MESI protocol. The processor is still assumed to issue only read (PrRd) and write (PrWr) requests. However, since we do not have an invalid state, to specify actions on a tag mismatch we add two more request types: processor read miss (PrRdMiss) and write miss (PrWrMiss). As for bus transactions, we have bus read (BusRd), bus write back (BusWB), and a new transaction called bus update (BusUpd). The BusRd and BusWB transactions have the usual semantics. The BusUpd transaction takes the specific word (or bytes) written by the processor and broadcasts it on the bus so that all other processors' caches can update themselves. By broadcasting only the contents of the specific modified word rather than the whole cache block, it is hoped that the bus bandwidth is more efficiently utilized. (See Exercise 5.4 for reasons why this may not always be the case.) As in the MESI protocol, to support the E state, a shared signal (S) is available to the cache controller. Finally, the only new capability needed is for the cache controller to update a locally cached memory block (labeled an Update action) with the contents that are being broadcast on the bus by a relevant BusUpd transaction.

4. Logically, there is another state as well, but it is rather crude and is used to bootstrap the protocol. A "miss mode" bit is provided with each cache line to force a miss when that block is accessed. Initialization software reads data into every line in the cache with the miss mode bit turned on to ensure that the processor will miss the first time it references a block that maps to that line. After this first miss, the miss mode bit is turned off and the cache operates normally.

selves do not translate into performance directly but only indirectly by increasing the cost of misses due to contention. Contention is very difficult to estimate because it depends on the timing parameters used and on the burstiness of the traffic, which is not captured by the frequency measurements. Contention, timing, and hence performance are also affected by lower-level interactions with hardware structures (like queues and buffers) and policies.

The simulations used in this section do not model contention. Instead, they use a simple PRAM cost model: all memory operations are assumed to complete in the same amount of time (here a single cycle) regardless of whether they hit or miss in the cache. There are three main reasons for this. First, the focus is on understanding inherent protocol behavior and trade-offs in terms of event frequencies, not so much on performance. Second, since we are experimenting with different cache block sizes and organizations, we would like the interleaving of references from application processes on the simulator to be the same regardless of these choices; that is, all protocols and block sizes should see the same trace of references. With the execution-driven rather than trace-driven simulation we use, this is only possible if we make the cost of every memory operation the same in the simulations. Otherwise, if a reference misses with a small cache block but hits with a larger one, for example, then it will be delayed by different amounts in the interleaving in the two cases. It would therefore be difficult to determine which effects are inherently due to the protocol and which are due to the particular parameter values chosen. Third, realistic simulations that model contention take much more time. The disadvantage of using this simple model even to measure frequencies is that the timing model may affect some of the frequencies we observe; however, this effect is small for the applications we study.

The illustrative workloads we use are the six parallel programs (from the SPLASH-2 suite) and one multiprogrammed workload described in Chapters 3 and 4. The parallel programs run in batch mode with exclusive access to the machine and do not include operating system activity in the simulations, whereas the multiprogrammed workload includes operating system activity. The number of applications used is relatively small, but the applications are primarily for illustration as discussed in Chapter 4; the emphasis here is on choosing programs that represent important classes of computation and with widely varying characteristics. The frequencies of basic operations for the applications appear in Table 4.1. We now study them in more detail to assess design trade-offs in cache coherency protocols.

5.4.2 Bandwidth Requirement under the MESI Protocol

We begin by using the default 1-MB, single-level caches per processor, as discussed in Chapter 4. These are large enough to hold the important working sets for the default problem sizes, which is a realistic scenario for all applications. We use four-way set associativity (with LRU replacement) to reduce conflict misses and a 64-byte cache block size for realism. Driving the workloads through a cache simulator that models the Illinois MESI protocol generates the state transition frequencies shown in Table 5.1. The data is presented as the number of state transitions of a particular type per 1,000 references issued by the processors. Note in the table that a new state,

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.