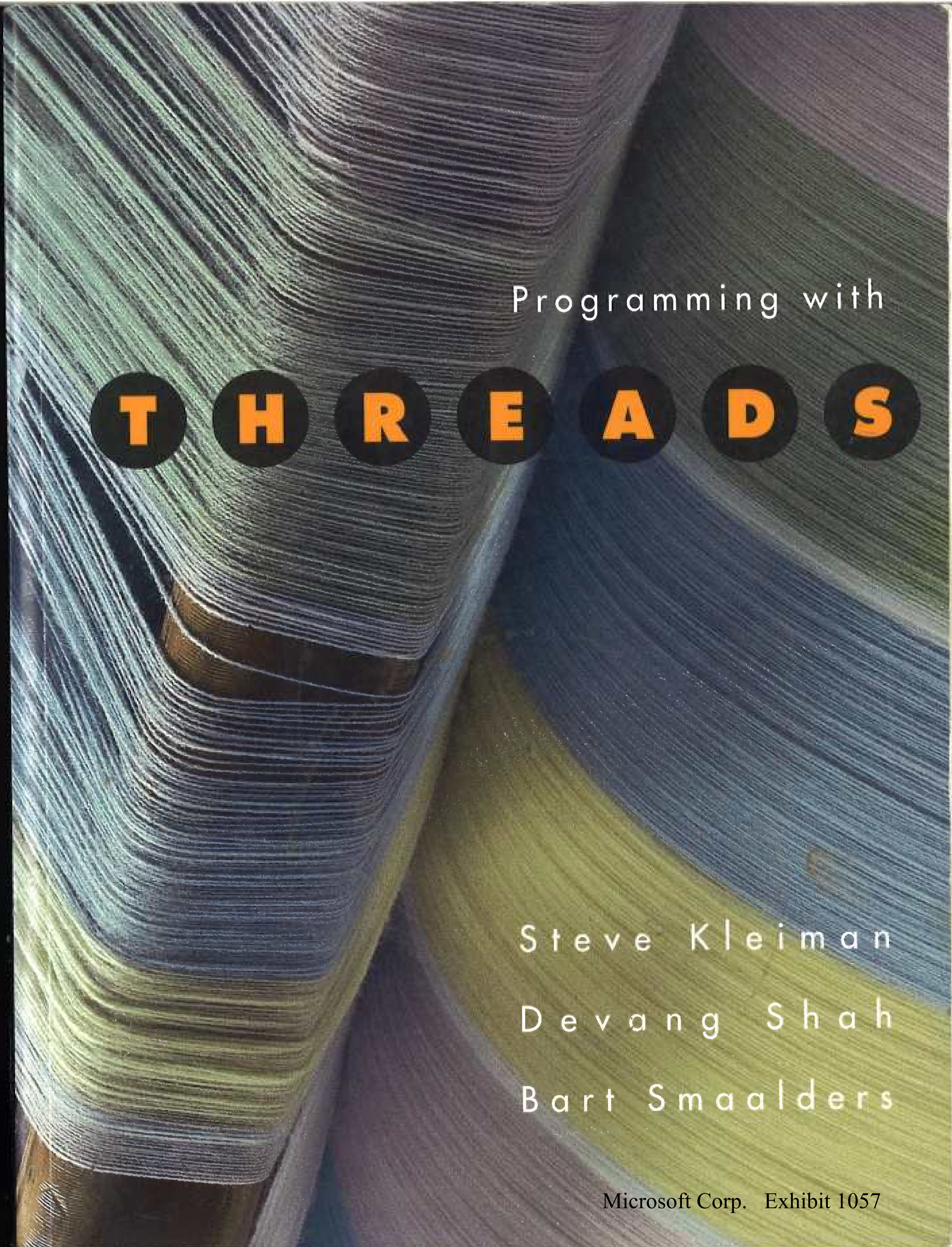

Programming with

THREADS

Steve Kleiman
Devang Shah
Bart Smaalders

Programming with Threads

Steve Kleiman
Devang Shah
Bart Smaalders

SunSoft Press
A Prentice Hall Title

The publisher offers discounts on this book when ordered in bulk quantities. For more information, contact: Corporate Sales Department, Prentice Hall PTR, One Lake Street, Upper Saddle River, NJ 07458. Phone: 800-382-3419 or 201-236-7156, Fax: 201-236-7141, email: corpsales@prenhall.com

Cover designer: *M & K Design, Palo Alto, California*
Manufacturing manager: *Alexis R. Heydt*
Acquisitions editor: *Gregory G. Doench*
Cover Design Director: *Jerry Votta*
Production Supervisor: *Joanne Anzalone*

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-172389-8

SunSoft Press
A Prentice Hall Title

Parallel Computation 16

One of the most powerful uses of threads is to speed up the execution of computationally intensive programs on shared memory multiprocessors. Unfortunately, effectively applying threads in this environment usually requires a deep understanding of the structure of the algorithm and how a proposed parallelized version of the algorithm is affected by the overheads of threads and the shared memory multiprocessor environment. Fortunately, many different algorithms have similar structures. This chapter covers in more detail a number of the thread paradigms introduced in Chapter 7, "Using Threads," that are useful in many parallel computation situations. Detailed templates and examples are presented.

Using Threads to Parallelize Algorithms

Threads running on separate processors in a shared-memory multiprocessor allow you to use "parallel processing algorithms" in your program. Unlike the other uses of threads described in this book, using threads to implement parallel algorithms can be frustrating:

- There are lots of techniques for "parallelizing" your program. How do you choose one that's not too hard to program and that offers substantial speedups compared to uniprocessor execution? Does the performance of the technique scale up in proportion to the number of processors you use?
- The overheads involved in synchronizing threads and sharing data among multiple processors may actually reduce the performance of your program. How can you anticipate and mitigate these problems?
- Like many performance improvements, parallelizing increases the complexity of your program. How can you be sure it's still correct?

These are all tough problems: we do not yet know how to solve an arbitrary problem efficiently on a multiprocessor of arbitrary size. This section does not offer universal solutions, but tries to demonstrate some simple ways to get started. By sticking with some common "paradigms" for parallel algorithms and threads, you can avoid a lot of errors and aggravation.

Though it may seem simplistic, the most important step in writing a parallel program is to think carefully about the global structure of your program and the computing structures that threads offer. To speed up the program, we're looking for a way to divide the work into a set of *tasks* so that:

- The tasks interact little with each other;
- The data shared by separate tasks is contained in a minimum of simple data structures that can be protected by locking mechanisms to prevent unsynchronized access;
- The number of tasks can be varied so as to match the number of processors;
- All tasks have equal computing requirements, or, instead, are configured in such a way that the set of tasks can keep all the processors fairly busy.

As we've seen, Amdahl's Law (Figure 7-3 on page 103) sets limits on the scalability of parallelized computations. Scalability is limited due to three kinds of overheads: synchronization overhead, contention, and balance. The synchronization operations required for correct multithreaded operation take time to execute, even when there is no contention. When two or more threads share data or locks, the system slows down due to contention for shared resources. And finally, balance refers to the ability of the algorithm to divide work evenly among the available processors. In the "serial" sections of your program, where only a single processor is used, balance is worst. Poor balance is often the result of dividing the work into large, unequal chunks: when the smaller chunks finish, they leave processors idle. If there are always at least as many runnable threads as available processors, the thread scheduler can keep the processors busy. While balance can be improved by dividing work into many small chunks that can be easily scheduled onto idle processors, making the *grain size* of the processing chunks small is usually accompanied by an increase in synchronization overhead, which hurts scalability.

Thread Paradigms

There are many ways to make your program parallel. Some techniques are very complex, depend on complicated data structures and locking strategies, depend on clever non-obvious algorithms, or require compiler support. We present a different approach in this section: three simple control structures, or paradigms, that can be applied to a wide range of applications.

Each paradigm can be characterized by:

- How the work is divided among parallel threads, and whether each thread executes the same code;
- How the threads are synchronized;

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.