

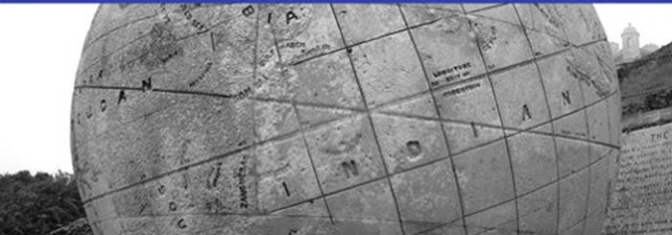


Darryl Gove

# Multicore Application Programming

For Windows, Linux, and  
Oracle® Solaris

**Developer's Library**



# Multicore Application Programming

---

# Multicore Application Programming: for Windows, Linux, and Oracle Solaris

## Table of Contents

Contents

Preface

Acknowledgments

About the Author

### 1 Hardware, Processes, and Threads

Examining the Insides of a Computer

The Motivation for Multicore Processors

Supporting Multiple Threads on a Single Chip

Increasing Instruction Issue Rate with Pipelined Processor Cores

Using Caches to Hold Recently Used Data

Using Virtual Memory to Store Data

Translating from Virtual Addresses to Physical Addresses

The Characteristics of Multiprocessor Systems

How Latency and Bandwidth Impact Performance

The Translation of Source Code to Assembly Language

The Performance of 32-Bit versus 64-Bit Code

Ensuring the Correct Order of Memory Operations

The Differences Between Processes and Threads

Summary

### 2 Coding for Performance

Defining Performance

# **Table of Contents**

## Understanding Algorithmic Complexity

- Examples of Algorithmic Complexity
- Why Algorithmic Complexity Is Important
- Using Algorithmic Complexity with Care

## How Structure Impacts Performance

- Performance and Convenience Trade-Offs in Source Code and Build Structures
- Using Libraries to Structure Applications
- The Impact of Data Structures on Performance

## The Role of the Compiler

- The Two Types of Compiler Optimization
- Selecting Appropriate Compiler Options
- How Cross-File Optimization Can Be Used to Improve Performance
- Using Profile Feedback
- How Potential Pointer Aliasing Can Inhibit Compiler Optimizations

## Identifying Where Time Is Spent Using Profiling

- Commonly Available Profiling Tools

## How Not to Optimize

## Performance by Design

## Summary

## 3 Identifying Opportunities for Parallelism

- Using Multiple Processes to Improve System Productivity
- Multiple Users Utilizing a Single System
- Improving Machine Efficiency Through Consolidation
  - Using Containers to Isolate Applications Sharing a Single System
  - Hosting Multiple Operating Systems Using Hypervisors
- Using Parallelism to Improve the Performance of a Single Task

# **Table of Contents**

- One Approach to Visualizing Parallel Applications
- How Parallelism Can Change the Choice of Algorithms
- Amdahls Law
- Determining the Maximum Practical Threads
- How Synchronization Costs Reduce Scaling

## **Parallelization Patterns**

- Data Parallelism Using SIMD Instructions
- Parallelization Using Processes or Threads
- Multiple Independent Tasks
- Multiple Loosely Coupled Tasks
- Multiple Copies of the Same Task
- Single Task Split Over Multiple Threads
- Using a Pipeline of Tasks to Work on a Single Item
- Division of Work into a Client and a Server
- Splitting Responsibility into a Producer and a Consumer
- Combining Parallelization Strategies

## **How Dependencies Influence the Ability Run Code in Parallel**

- Antidependencies and Output Dependencies
- Using Speculation to Break Dependencies
- Critical Paths

## **Identifying Parallelization Opportunities**

### **Summary**

## **4 Synchronization and Data Sharing**

### **Data Races**

- Using Tools to Detect Data Races
- Avoiding Data Races

### **Synchronization Primitives**

# **Table of Contents**

Mutexes and Critical Regions

Spin Locks

Semaphores

Readers-Writer Locks

Barriers

Atomic Operations and Lock-Free Code

Deadlocks and Livelocks

Communication Between Threads and Processes

Memory, Shared Memory, and Memory-Mapped Files

Condition Variables

Signals and Events

Message Queues

Named Pipes

Communication Through the Network Stack

Other Approaches to Sharing Data Between Threads

Storing Thread-Private Data

Summary

## **5 Using POSIX Threads**

Creating Threads

Thread Termination

Passing Data to and from Child Threads

Detached Threads

Setting the Attributes for Pthreads

Compiling Multithreaded Code

Process Termination

Sharing Data Between Threads

Protecting Access Using Mutex Locks

# **Table of Contents**

Mutex Attributes

Using Spin Locks

Read-Write Locks

Barriers

Semaphores

Condition Variables

## **Variables and Memory**

## **Multiprocess Programming**

Sharing Memory Between Processes

Sharing Semaphores Between Processes

Message Queues

Pipes and Named Pipes

Using Signals to Communicate with a Process

## **Sockets**

## **Reentrant Code and Compiler Flags**

## **Summary**

## **6 Windows Threading**

### **Creating Native Windows Threads**

Terminating Threads

Creating and Resuming Suspended Threads

Using Handles to Kernel Resources

### **Methods of Synchronization and Resource Sharing**

An Example of Requiring Synchronization Between Threads

Protecting Access to Code with Critical Sections

Protecting Regions of Code with Mutexes

Slim Reader/Writer Locks

Semaphores

# **Table of Contents**

Condition Variables

Signaling Event Completion to Other Threads or Processes

Wide String Handling in Windows

Creating Processes

Sharing Memory Between Processes

Inheriting Handles in Child Processes

Naming Mutexes and Sharing Them Between Processes

Communicating with Pipes

Communicating Using Sockets

Atomic Updates of Variables

Allocating Thread-Local Storage

Setting Thread Priority

Summary

## **7 Using Automatic Parallelization and OpenMP**

Using Automatic Parallelization to Produce a Parallel Application

Identifying and Parallelizing Reductions

Automatic Parallelization of Codes Containing Calls

Assisting Compiler in Automatically Parallelizing Code

Using OpenMP to Produce a Parallel Application

Using OpenMP to Parallelize Loops

Runtime Behavior of an OpenMP Application

Variable Scoping Inside OpenMP Parallel Regions

Parallelizing Reductions Using OpenMP

Accessing Private Data Outside the Parallel Region

Improving Work Distribution Using Scheduling

Using Parallel Sections to Perform Independent Work

Nested Parallelism



# **Table of Contents**

Using OpenMP for Dynamically Defined Parallel Tasks

Keeping Data Private to Threads

Controlling the OpenMP Runtime Environment

Waiting for Work to Complete

Restricting the Threads That Execute a Region of Code

Ensuring That Code in a Parallel Region Is Executed in Order

Collapsing Loops to Improve Workload Balance

Enforcing Memory Consistency

An Example of Parallelization

Summary

## **8 Hand-Coded Synchronization and Sharing**

Atomic Operations

Using Compare and Swap Instructions to Form More Complex Atomic Operations

Enforcing Memory Ordering to Ensure Correct Operation

Compiler Support of Memory-Ordering Directives

Reordering of Operations by the Compiler

Volatile Variables

Operating System-Provided Atomics

Lockless Algorithms

Dekkers Algorithm

Producer-Consumer with a Circular Buffer

Scaling to Multiple Consumers or Producers

Scaling the Producer-Consumer to Multiple Threads

Modifying the Producer-Consumer Code to Use Atomics

The ABA Problem

Summary

# **Table of Contents**

## **9 Scaling with Multicore Processors**

### **Constraints to Application Scaling**

Performance Limited by Serial Code

Superlinear Scaling

Workload Imbalance

Hot Locks

Scaling of Library Code

Insufficient Work

Algorithmic Limit

### **Hardware Constraints to Scaling**

Bandwidth Sharing Between Cores

False Sharing

Cache Conflict and Capacity

Pipeline Resource Starvation

### **Operating System Constraints to Scaling**

Oversubscription

Using Processor Binding to Improve Memory Locality

Priority Inversion

### **Multicore Processors and Scaling**

### **Summary**

## **10 Other Parallelization Technologies**

### **GPU-Based Computing**

### **Language Extensions**

Threading Building Blocks

Cilk++

Grand Central Dispatch

Features Proposed for the Next C and C++ Standards

# **Table of Contents**

Microsoft's C++/CLI

Alternative Languages

Clustering Technologies

MPI

MapReduce as a Strategy for Scaling

Grids

Transactional Memory

Vectorization

Summary

## **11 Concluding Remarks**

Writing Parallel Applications

Identifying Tasks

Estimating Performance Gains

Determining Dependencies

Data Races and the Scaling Limitations of Mutex Locks

Locking Granularity

Parallel Code on Multicore Processors

Optimizing Programs for Multicore Processors

The Future

## **Bibliography**

Books

POSIX Threads

Windows

Algorithmic Complexity

Computer Architecture

Parallel Programming

OpenMP

# **Table of Contents**

Online Resources

Hardware

Developer Tools

Parallelization Approaches

Index