The Art of Concurrency

并行开发艺术 (影印版)

O'REILLY®
東南大學出版社

Clay Breshears 著

并行开发艺术(影印版)

The Art of Concurrency

Clay Breshears

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Taipei · Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

并行开发艺术：英文/（美）布雷谢斯（Breshears, C.）著. —影印本. —南京：东南大学出版社，2010.1

书名原文：The Art of Concurrency

ISBN 978-7-5641-1929-4

Ⅰ. 并… Ⅱ. 布… Ⅲ. 软件开发－英文 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2009）第 206908 号

江苏省版权局著作权合同登记

图字：10-2009-248 号

并行开发艺术（影印版）

出版发行：东南大学出版社

地　　址：南京四牌楼 2 号　　邮编：210096

出 版 人：江　汉

网　　址：http://press.seu.edu.cn

电子邮件：press@seu.edu.cn

印　　刷：扬中市印刷有限公司

开　　本：787 毫米 × 980 毫米　16 开本

印　　张：19 印张

字　　数：319 千字

版　　次：2010 年 1 月第 1 版

印　　次：2010 年 1 月第 1 次印刷

书　　号：ISBN 978-7-5641-1929-4

印　　数：1~1600 册

定　　价：48.00 元（册）

本社图书若有印装质量问题，请直接与读者服务部联系。电话（传真）：025-83792328

To my parents, for all their love, guidance,
and support.

PREFACE

Why Should You Read This Book?

MULTICORE PROCESSORS MADE A BIG SPLASH WHEN THEY WERE FIRST INTRODUCED. Bowing to the physics of heat and power, processor clock speeds could not keep doubling every 18 months as they had been doing for the past three decades or more. In order to keep increasing the processing power of the next generation over the current generation, processor manufacturers began producing chips with multiple processor cores. More processors running at a reduced speed generate less heat and consume less power than single-processor chips continuing on the path of simply doubling clock speeds.

But how can we use those extra cores? We can run more than one application at a time, and each program could have a separate processor core devoted to the execution. This would give us truly parallel execution. However, there are only so many apps that we can run simultaneously. If those apps aren't very compute-intensive, we're probably wasting compute cycles, but now we're doing it in more than one processor.

Another option is to write applications that will utilize the additional cores to execute portions of the code that have a need to perform lots of calculations and whose computations are independent of each other. Writing such programs is known as *concurrent programming*. With any programming language or methodology, there are techniques, tricks, traps, and tools to design and implement such programs. I've always found that there is more "art" than "science" to programming. So, this book is going to give you the knowledge and one or two of the "secret handshakes" you need to successfully practice the art of concurrent programming.

In the past, parallel and concurrent programming was the domain of a very small set of programmers who were typically involved in scientific and technical computing arenas. From now on, concurrent programming is going to be mainstream. Parallel programming will eventually become synonymous with "programming." Now is *your* time to get in on the ground floor, or at least somewhere near the start of the concurrent programming evolution.

Who Is This Book For?

This book is for programmers everywhere.

I work for a computer technology company, but I'm the only computer science degree-holder on my team. There is only one other person in the office within the sound of my voice who would know what I was talking about if I said I wanted to parse an LR(1) grammar with a deterministic pushdown automata. So, CS students and graduates aren't likely to make up the bulk of the interested readership for this text. For that reason, I've tried to keep the geeky CS material to a minimum. I assume that readers have some basic knowledge of data structures and algorithms and asymptotic efficiency of algorithms (Big-Oh notation) that is typically taught in an undergraduate computer science curriculum. For whatever else I've covered, I've tried to include enough of an explanation to get the idea across. If you've been coding for more than a year, you should do just fine.

I've written all the codes using C. Meaning no disrespect, I figured this was the lowest common denominator of programming languages that supports threads. Other languages, like Java and C#, support threads, but if I wrote this book using one of those languages and you didn't code with the one I picked, you wouldn't read my book. I think most programmers who will be able to write concurrent programs will be able to at least "read" C code. Understanding the concurrency methods illustrated is going to be more important than being able to write code in one particular language. You can take these ideas back to C# or Java and implement them there.

I'm going to assume that you have read a book on at least one threaded programming method. There are many available, and I don't want to cover the mechanics and detailed syntax of multithreaded programming here (since it would take a whole other book or two). I'm not going to focus on using one programming paradigm here, since, for the most part, the functionality of these overlap. I will present a revolving usage of threading implementations across the wide spectrum of algorithms that are featured in the latter portion of the book. If there are circumstances where one method might differ significantly from the method used, these differences will be noted.

I've included a review of the threaded programming methods that are utilized in this book to refresh your memory or to be used as a reference for any methods you have not had the chance to study. I'm not implying that you need to know all the different ways to program with threads. Knowing one should be sufficient. However, if you change jobs or find that what you know about programming with threads cannot easily solve a programming problem you have been assigned, it's always good to have some awareness of what else is available—this may help you learn and apply a new method quickly.

What's in This Book?

Chapter 1, *Want to Go Faster? Raise Your Hands if You Want to Go Faster!*, anticipates and answers some of the questions you might have about concurrent programming. This chapter explains the differences between parallel and concurrent, and describes the four-step threading methodology. The chapter ends with a bit of background on concurrent programming and some of the differences and similarities between distributed-memory and shared-memory programming and execution models.

Chapter 2, *Concurrent or Not Concurrent?*, contains a lot of information about designing concurrent solutions from serial algorithms. Two concurrent design models—task decomposition and data decomposition—are each given a thorough elucidation. This chapter gives examples of serial coding that you may not be able to make concurrent. In cases where there is a way around this, I've given some hints and tricks to find ways to transform the serial code into a more amenable form.

Chapter 3, *Proving Correctness and Measuring Performance*, first deals with ways to demonstrate that your concurrent algorithms won't encounter common threading errors and

to point out what problems you might see (so you can fix them). The second part of this chapter gives you ways to judge how much faster your concurrent implementations are running compared to the original serial execution. At the very end, since it didn't seem to fit anywhere else, is a brief retrospective of how hardware has progressed to support the current multicore processors.

Chapter 4, *Eight Simple Rules for Designing Multithreaded Applications*, says it all in the title. Use of these simple rules is pointed out at various points in the text.

Chapter 5, *Threading Libraries*, is a review of OpenMP, Intel Threading Building Blocks, POSIX threads, and Windows Threads libraries. Some words on domain-specific libraries that have been threaded are given at the end.

Chapter 6, *Parallel Sum and Prefix Scan*, details two concurrent algorithms. This chapter also leads you through a concurrent version of a selection algorithm that uses both of the titular algorithms as components.

Chapter 7, *MapReduce*, examines the MapReduce algorithmic framework; how to implement a handcoded, fully concurrent reduction operation; and finishes with an application of the MapReduce framework in a code to identify friendly numbers.

Chapter 8, *Sorting*, demonstrates some of the ins and outs of concurrent versions of Bubblesort, odd-even transposition sort, Shellsort, Quicksort, and two variations of radix sort algorithms.

Chapter 9, *Searching*, covers concurrent designs of search algorithms to use when your data is unsorted and when it is sorted.

Chapter 10, *Graph Algorithms*, looks at depth-first and breadth-first search algorithms. Also included is a discussion of computing all-pairs shortest path and the minimum spanning tree concurrently.

Chapter 11, *Threading Tools*, gives you an introduction to software tools that are available and on the horizon to assist you in finding threading errors and performance bottlenecks in your concurrent programs. As your concurrent code gets more complex, you will find these tools invaluable in diagnosing problems in minutes instead of days or weeks.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities.

`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events,

event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*The Art of Concurrency* by Clay Breshears. Copyright 2009 Clay Breshears, 978-0-596-52153-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9780596521530

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

http://www.oreilly.com

Safari® Books Online

When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://my.safaribooksonline.com/*.

Acknowledgments

I want to give my thanks to the following people for their influences on my career and support in the writing of this book. Without all of them, you wouldn't be reading this and I'd probably be flipping burgers for a living.

To **Joseph Sargent** and **Stanley Chase** for bringing *Colossus: The Forbin Project* to the big screen in 1970. This movie was probably the biggest influence in my early years in getting me interested in computer programming and instilling within me the curiosity to figure out what cool and wondrous things computers could do.

To **Roger Wink** for fanning the flame of my interest in computers, and for his 30-plus years of friendship and technical knowledge. He taught me Bubblesort in COBOL and is always working on something new and interesting that he can show off whenever we get the chance to meet up.

To **Bill Magro** and **Tom Cortese** for being my first manager at Intel and one of my first teammates at the Intel Parallel Applications Center. Working at the PAC gave me the chance to get my "hands dirty" with lots of different parallel codes, to interact with applications and customers from many different technical and commercial areas, and to learn new methods and new threading libraries. It was a "dream come true" job for me.

To **Jerry Baugh**, **Bob Chesebrough**, **Jeff Gallagher**, **Ravi Manohar**, **Mike Pearce**, **Michael Wrinn**, and **Hua (Selwyn) You** for being fantastic colleagues at Intel, past and present, and for reviewing chapters of my book for technical content. I've relied on every one of these guys for their wide range of technical expertise; for their support, patience, and willingness to help me with my projects and goals; for their informed opinions; and for their continuing camaraderie throughout my years at Intel.

To my editor, **Mike Loukides**, and the rest of the staff at O'Reilly who had a finger in this project. I couldn't have done anything like this without their help and advice and nagging me about my deadlines.

To **Gergana Slavova**, who posed as my "target audience" and reviewed the book from cover to cover. Besides keeping me honest to my readers by making me explain complex ideas in simple terms and adding examples when I'd put too many details in a single paragraph, she peppered her comments with humorous asides that broke up the monotony of the tedium of the revision process (and she throws a slammin' tea party, too).

To **Henry Gabb** for his knowledge of parallel and multithreaded programming, for convincing me to apply for a PAC job and join him at Intel back in 2000, and for his devotion to SEC football and the Chicago Cubs. During the almost 15 years we've known each other, we've worked together on many different projects and we've each been able to consult with the other on technical questions. His knowledge and proficiency as a technical reviewer of this text, and many other papers of mine he has so kindly agreed to review over the years, have improved my written communication skills by an order of magnitude.

And finally, a big heartfelt "thank you" to my patient and loving wife, **Lorna**, who now has her husband back.

About the Author

Dr. Clay Breshears has been with Intel Corporation since September 2000. He started as a senior parallel application engineer at the Intel Parallel Applications Center in Champaign, Illinois, implementing multithreaded and distributed solutions in customer applications. Clay is currently a courseware architect, specializing in multicore and multithreaded programming and training. Before joining Intel, Clay was a research scientist at Rice University helping Department of Defense researchers make the best use of the latest High Performance Computing (HPC) platforms and resources. Clay received his Ph.D. in computer science from the University of Tennessee, Knoxville, in 1996, but he has been involved with parallel computation and programming for over 20 years; six of those years were spent in academia at Eastern Washington University and the University of Southern Mississippi.

Colophon

The cover image is an aerial view of wheat-harvesting combines from Getty Images. The cover fonts are Akzidenz Grotesk and Orator. The text font is Adobe's Meridien; the heading font is ITC Bailey.

O'Reilly Media, Inc. 介绍

O'Reilly Media, Inc. 是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的 50 本书之一）到 GNN（最早的 Internet 门户和商业网站），再到 WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc. 一直处于 Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc. 是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc. 具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc. 依靠他们及时地推出图书。因为 O'Reilly Media, Inc. 紧密地与计算机业界联系着，所以 O'Reilly Media, Inc. 知道市场上真正需要什么图书。

出版说明

随着计算机技术的成熟和广泛应用，人类正在步入一个技术迅猛发展的新时期。计算机技术的发展给人们的工业生产、商业活动和日常生活都带来了巨大的影响。然而，计算机领域的技术更新速度之快也是众所周知的，为了帮助国内技术人员在第一时间了解国外最新的技术，东南大学出版社和美国 O'Reilly Meida, Inc. 达成协议，将陆续引进该公司的代表前沿技术或者在某专项领域享有盛名的著作，以影印版或者简体中文版的形式呈献给读者。其中，影印版书籍力求与国外图书“同步”出版，并且“原汁原味”展现给读者。

我们真诚地希望，所引进的书籍能对国内相关行业的技术人员、科研机构的研究人员和高校师生的学习和工作有所帮助，对国内计算机技术的发展有所促进。也衷心期望读者提出宝贵的意见和建议。

最新出版的影印版图书，包括：

- 《真实世界的 Haskell》（影印版）
- 《卓有成效的程序员》（影印版）
- 《Java Web 服务：建构与运行》（影印版）
- 《并行开发艺术》（影印版）
- 《使用 Perl 实现系统管理自动化 第二版》（影印版）
- 《Java 消息服务 第二版》（影印版）
- 《深入浅出网络管理》（影印版）
- 《Ruby 最佳实践》（影印版）
- 《更快速网站》（影印版）
- 《正则表达式 Cookbook》（影印版）
- 《flex 与 bison》（影印版）

CONTENTS

CHAPTER ONE

Want to Go Faster? Raise Your Hands if You Want to Go Faster!