<<并行开发艺术>>

图书基本信息

书名：<<并行开发艺术>>

13位ISBN编号：9787564119294

10位ISBN编号：7564119292

出版时间：2010-1

出版时间：东南大学出版社

作者：Clay Breshears

页数：285

前言

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 "sdence" 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 yourtime 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 [or 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 Pre/ix ScaB, 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.

<<并行开发艺术>>

内容概要

如果你想利用并发程序设计充分发挥多核心处理器的性能，《并行开发艺术(影印版)》就为你提供了所需要的实践知识和亲身体验。
《并行开发艺术》是一份难得的材料，专注于多核心处理器的共享内存模型，而不只是理论模型或分布式内存架构。
《并行开发艺术(影印版)》提供了详尽的解释和可用的示例，帮助你将算法从串行代码转化为并行代码，此外还包括建议和分析，避免程序员的典型错误。
作者为Intel资深工程师，拥有超过20年的并行和并发编程经验，《并行开发艺术(影印版)》将会帮助你：· 探索共享内存与分布式内存编程间的区别· 学习设计多线程程序的指导方针，包括测试和调整· 研究如何善用不同的线程库，包括Windows线程、POSIX线程、OpenMP和Intel Threading Building Blocks· 研究如何实现排序、查找、图形和其他实用计算的并行算法《并行开发艺术》向你展示如何扩展算法，以从新型的多核处理器中获益。
对于开发并行算法和并发编程来说，《并行开发艺术(影印版)》不可或缺。

作者简介

Clay Breshears博士，是Intel公司的课程架构师，专攻多核心及多线程程序设计与培训。

<<并行开发艺术>>

书籍目录

章节摘录

插图：Two types of dependencies can occur between tasks. The first is order dependency, where sometask relies on the completed results of the computations from another task. This reliance canbe a direct need to use the computed values as input to the succeeding task, or it may simplybe the case that the task that follows will be updating the same memory locations as theprevious task and you must ensure that all of the previous updates have been completed beforeproceeding. Both of these cases describe a potential data race, which we need to avoid.For example, if you are building a house, putting up the roof involves attaching the rafters tothe walls, laying down the decking, and applying the shingles. The dependence between thesethree tasks is one of execution order. You can't put down shingles until the decking is there,and you can't nail down the decking unless you have the rafters in place. So, instead of hiringthree teams to do these three tasks in parallel, you can hire one roofing crew to do all three inthe order required （there is parallelism within each of the roofing steps, plus the act of puttingon the roof is independent of installing the electrical wiring, the plumbing, and putting updrywall）.To satisfy an execution order constraint, you can schedule tasks that have an order dependencyonto the same thread and ensure that the thread executes the tasks in the proper sequence.The serial code was written with the order dependency already taken care of. So, the serialalgorithm should guide the correct decomposition of the computations into tasks andassignment of those tasks to threads. Still, even after grouping tasks to execute on threads,there may be order constraints between threads. If regrouping tasks to threads is not an optionor will severely hurt performance, you will be forced to insert some form of synchronizationto ensure correct execution order.The second type of dependency is data dependency. Identifying potential data dependenciescan be straightforward: look for those variables that are featured on the left side of theassignment operator. Data races require that the variable in question have at least one threadthat is writing to that variable. Check for any assignment of values to the same variable thatmight be done concurrently as well as any updates to a variable that could be read concurrently.Of course, using pointers to reference memory locations can make the identification processtrickier. There are tools （covered in Chapter 11） that can assist in finding nonobvious datadependencies in your code.

媒体关注与评论

“这本书紧扣题目，令人赏心悦目。
艺术无法传授，但追随大师的脚印便可觅得踪迹。
作者在超级计算机公司工作30余年，而后又在学术界浸淫10年，我可以毫无疑问地说，本书实至名归。”
——Tom Murphy，Contra Costa学院，计算机科学程序委员会主席“终于，一本专注于并发的书籍出现了。
它包含诸多现实世界的非凡算法，利用并行编程技术对其进行分析，提升它们的性能。”
——Mike Pearce，Intel Software Network，并行计算体系主管

编辑推荐

《并行开发艺术(影印版)》是由东南大学出版社出版的。

版权说明