1. Introduction

Multicore processors have become the mainstream computing platform. However, multicore processors cannot automatically improve the single-threaded performance. Thus, it becomes increasingly important for programmers to learn to develop multithreaded programs to extract thread-level parallelism from applications, so that software programs can harness the full potential of multicore chips. Nevertheless, undergraduate students in computer science or computer engineering programs are used to develop single-threaded programs targeting uniprocessor, which does not sufficiently prepare them to develop correct and efficient multithreaded programs that can fully utilize the power of multicore computing.

Recently a few institutions including MIT [1] and Georgia Tech [2] have begun to offer multicore programming courses. However, both these courses [1, 2] focus on studying and using a special multicore architecture, i.e., Cell processor [3] used in the Sony PlayStation 3, which may limit the wide adoption of these courses by other institutions intending to educate their students in multithreaded software development for general-purpose multicore processors.

In this paper, I present my recent effort to develop a course on programming for general-purpose multicore processors. Specifically, the course development and project is centered on Intel Core 2 dual-core architecture for general-purpose computing. As one of the few earliest courses on multicore programming for general-purpose computers, this course is expected to benefit students in our institution and beyond. This project is partially supported by the National Science Foundation (NSF). The developed course materials will be made freely available by the end of the project, which are expected to benefit more students and instructors interested in multicore programming. A temporary course website can be found at http://www.people.vcu.edu/~wzhang4/491.htm.

2. Basic Course Information

This course was offered at the Electrical and Computer Engineering Department of Southern Illinois University Carbondale for the first time in Spring 2010 by applying the problem-based learning approach [4]. This course will be offered again at the Electrical and Computer Engineering Department of Virginia Commonwealth University in Spring 2011.
The prerequisites of this course include a junior level computer architecture course (ECE 329) and a junior level operating system course (CS 306). Students are also expected to be familiar with C/C++ programming, which is already required before they can take the operating system course CS 309.

While there are many books on multithreaded programming for uniprocessor, to the best of my knowledge, there are only few books on programming for multicore processors, not to mention textbooks. Given the limited options, I selected a recent book entitled “Multi-Core Programming: Increasing Performance through Software Multithreading”, which was written by Intel engineers Shameem Akhter and Jason Roberts and Published by Intel Press. It should be noted that this book was not written as a textbook for classroom teaching and it only focuses on introducing Intel multicore architecture and software development tools. Nevertheless, I found this book, combined with the lectures and labs I developed based on open-source software can provide students’ rich resources for learning programming for general-purpose multicore processors.

3. Lectures Developed

While this course focuses on teaching programming for Intel Core-2 Duo processor, I also try to introduce general principles of multicore architecture and multithreaded programming so that students can extend their skills to other multicore programming tasks. In order to help students organize their learning and to facilitate easy adoption and reusability, I divide this course into 8 modules, which are organized around 4 themes as described below.

Theme1: Multicore Architecture
- Module 1 – Introduction to the Intel Core 2 Quad-Core and generally Multicore Architectures: Technology trends and challenges of future microprocessor design; architectural and microarchitectural features of Intel Core 2 quad-core processor; comparison of single-core, hyper-threading, multi-processors, and multicore architectures; and the differences between multicore processors and parallel/distributed computers.

Theme2: Find Concurrency and Extract Threads
- Module 2 – Fundamentals about Threads: The meaning of threads, differences between processes and threads, creating and destroying threads, thread execution models, thread state diagram, thread priority and scheduling, kernel vs. user threads.
- Module 3 – Finding Concurrency: Introduction of task and data decomposition, dependence analysis, concurrent program and data structuring patterns.
- Module 4 – Multithreaded Programming Models: Pthreads or OpenMP, introduction of automatic multithreading by parallel compilation, and the tradeoffs among different approaches.

Theme3: Ensure the Correctness of Multithreaded Programs
Module 5 – Synchronization: Race conditions, critical sections, reader/writer locks, spin locks, semaphores, and condition variables.

Module 6 – Software Tools for Debugging Multithreaded Applications: Tutorials on the advanced graphical debugger (dbx) from Sun Studio 12, and GDB/DDD for thread examination.

Theme 4: Maximize Multithreaded Performance on Multicore Chips

Module 7 – Performance Measurement and Load Balance: Throughput vs. latency, Amdahl’s law, mapping software concurrency to the parallel hardware, code transformation, data layout optimization, load balance, and thread migration.

Module 8 – Software Tools for Maximizing Performance on Multicore Processors: Sun Studio performance analyzer, and utilities to profile programs, i.e., prof, gprof.

4. Labs

To provide students’ hands-on experience in developing, debugging and optimizing multithreaded programs on general-purpose multicore computers, I have developed a set of six labs as the following. All the labs are based on either OpenMP or Pthread, both of them are free.

Lab 1: 1) You need to write a C/C++ program to find a number with 9 digits. Each digit can be any number between 1 and 9, and the number of all these 9 digits should be different (i.e. there is no duplicated number in these 9 digits). Also, the first digit can be divided by 1 without any remainder (from the most significant digit), the first two digits can be divided by 2 without any reminder, the first three digits can be divided by 3 without any reminder, … , the first 9 digits can be divided by 9 without any reminder. 2) In this Lab, we provide one benchmark for your test on real multi-core processors. This benchmark utilizes OpenMP for multi-threading programming and the number of threads can be easily set. First, you should run this benchmark with different number of threads setting, and record their execution time. Also, you need to run more times (e.g. 10 times or more) of this benchmark to get an average execution time. Second, you need to compare the results you get under different number of threads settings to indicate the performance improvement in your report with tables or figures.

The objectives of lab 1 are: 1) to warm up your C/C++ programming skills for the future Labs and the project of this class and 2) to test the multi-core programming benchmark on real multi-core processors, record and understand the performance improvement of multi-core computing.

Lab 2: In this Lab, we provide a framework codes for you to create processes and threads, including one program with two processes and another with two threads

---

1 OpenMP is supported by Microsoft Visual C++ 2005 or newer version and Pthread can be supported by open-source C compiler such as gcc.
in the main process. First, you need to add your codes to allocate some memory space from the heap for a string, and assign an initial content “siu” into this allocated space in the main function. Second, for the two-thread program, you need add your codes to change the content to “siu333” in the thread 1, display it and wait 10 seconds to display it again, while you need to change it to “siu888” in the thread 2 after waiting 3 seconds from the start of this thread. Third, for the two-process program, you need to add your codes to change content to “siu333” in the child process, display it and wait 10 seconds to display it again, while you need to change it to “siu888” in the parent process after waiting 3 seconds from the start of this process. Last, you need to compare the results in the two threads program and the two processes program, and give your conclusions.

The objective of lab 2 is to practice how to create processes and threads in the Unix system, and understand the relation and difference between processes and threads.

- **Lab 3:** In this Lab, you need to do two programming to fulfill the requirements below, respectively.
  
  **Program 1:** Please use OpenMP to parallel run the SOR benchmark and you just need to parallel execute the phase 2 of this benchmark (data processing phase). Also, you need to statically distribute the work among threads, and record the results under different number of threads settings. (50 points)
  
  **Program 2:** Please use OpenMP to create two sections to run two data processing functions in two different threads. Function A is to do SOR data processing, while function B is to do 2-D matrix transportation. Also, you need to print the thread id running a specific function.

The objective of lab 3 is to practice and be familiar with some basic features of OpenMP, and understand their differences, including “for loops”, “sections”, etc.

- **Lab 4:** In this Lab, we have a benchmark for you to use OpenMP to parallel execute it to improve its performance of execution time. First, you need to add OpenMP support into this benchmark, like dynamically set thread number. Second, you need to select the proper place in this benchmark to use OpenMP to improve the performance of execution time. Third, you need to add your codes to record the execution time of this benchmark to evaluate your codes. Third, you need to run your changed benchmark under different number of threads settings for several times (e.g. 10 times or more), record the data, calculate the average execution time, and show your performance improvement. Fourth, you need to choose one inappropriate parallel case, evaluate its performance under different number of threads settings, and compare the results with your right parallel case. Last, you need to use your data to explain the reason why your change can improve the performance, while another case degrades the performance.
The objective of lab 4 is to practice the task decomposition of parallel programming, and understand the inappropriate parallel execution will not improve but degrade the performance.

- **Lab 5:** In this Lab, you need to create two threads, and let them join. Then, you need to print 20 numbers from 1 to 20 to the screen **sequentially**. The second thread is responsible for printing 5, 10, 15, and 20, while the first thread is to print all other numbers. Also, you need to show the id of the thread which prints a specific number.

The objective of lab 5 is to practice and be familiar with some basic features of Pthread, like threads creating, threads joining, thread exiting, etc.

- **Lab 6:** We provide following codes framework to implement the producer/consumer models. The producer will generate random numbers between 0 and 999 and put the new generated number into working set queue. Consumers will fetch the number from the working set queue, and maintain a global sum. Also, both the producer and consumer stop work when the sum exceeds a given level (like 10,000). However, the codes are **NOT** finished yet. In the `do_work` function, the codes for the main thread (i.e. producer) and other threads (i.e. consumers) are not provided. You should finish the remaining work to satisfy the feature description above, and avoid any potential race condition.

The objective of lab 6 is to practice the OpenMP programming, understand the basic method of avoiding race condition (e.g. critical section, atomic, etc) in OpenMP, and be familiar with the **producer and consumer models**.

5. Concluding Remarks

This paper introduces the course development efforts of multicore programming, including the course lecture modules and six associated labs. The goal of this course is to familiarize students with developing multithreaded programs that can maximize performance on general-purpose multicore processors. Due to the importance of multicore programming skills [5, 6] and the lack of educational materials in this domain, I believe this paper can potentially benefit the computer science and engineering faculty who are interested in teaching courses on programming for general-purpose multicore processors.

**References:**
