Introducing Parallel Computing in Undergraduate Curriculum (Poster)

Cordelia M. Brown, Yung-Hsiang Lu, and Samuel Midkiff
School of Electrical and Computer Engineering, Purdue University
West Lafayette, Indiana, USA
brown83, yunglu, smidkiff@purdue.edu

Abstract— **This paper reports our experiences and findings in teaching the concepts of parallel computing in three undergraduate courses (two programming courses and one hardware course). These three courses are parts of the modernization for integrating parallel computing into the curriculum. Instead of introducing a new course, we modify existing courses so that students learn different aspects of parallel computing throughout the four years, in both hardware and software. This paper describes the concepts that have been integrated into the courses, the assessments, and our observations.**

Keywords - parallelism; undergraduate curriculum; paarllel computing;practical experience

I. INTRODUCTION

The traditional approach teaches sequential programming, algorithms, and data structures in undergraduate courses, and then introduces parallel programming in graduate courses. Parallelism would be introduced in one or two undergraduate courses, such as operating systems. This approach would treat parallel computing as a specialized rather than a mainstream topic. In the last few years all computers have switched to multicore processors. Even tablets and smartphones utilize multicore processors. This shift in processor architectures has made understanding parallel computing essential for undergraduate students, partly due to the pressing needs from industry.

In response to this paradigm shift, the faculty at Purdue's School of Electrical and Computer Engineering have been integrating the concepts of parallel computing into the undergraduate curriculum so that students learn different aspects of parallel computing in multiple courses, both hardware and software, across all years. The changed courses are listed below. The numbers indicate the students' years when the courses are usually taken.

- Introduction
  - 1 Introduction to Computer Engineering
- Software
  - 2 Advanced C Programming
  - 3 Data Structures
  - 3 Script Programming
  - 3 Object-Oriented Programming
  - 4 Computer Networks
- Hardware
  - 2 Digital System Design
  - 3 Microprocessor and Assembly Programming
  - 4 Computer Organization

The authors are directly responsible for two programming courses and one hardware course; this paper will focus on these three courses:

1. Object-Oriented Programming. This is an elective course for Computer Engineering students.
2. Advanced C Programming. It is required for all students in Computer Engineering. It is also a popular elective course for students in Electrical Engineering and in the other departments in the College of Engineering.
3. Introduction to Digital System Design. It is required for all students in both Electrical and Computer Engineering.

Among the three courses, Object-Oriented Programming is modernized first because it is elective. Since year 2008, understanding parallel programming is one of the requirements to pass this course as one of the learning objectives (also called ABET Outcomes). Major changes in Advanced C Programming started in 2012 and introducing parallel computing is one of the changes. In Introduction to Digital System Design, students learn parallelism with a focus on hardware performance. We evaluate students' understanding through programming assignments and in-class exams. The following sections summarize the changes in these courses:

II. OBJECT-ORIENTED PROGRAMMING

This course was modernized first for teaching the concepts of parallel computing due to three reasons. First, it is an elective course and it is not a prerequisite of any other course. Hence, changes in this course would have less impact on the other courses. Second, this course had topics that overlapped with another course and these topics could be eliminated to accommodate the new topics for parallel computing. Third, threads are supported in Java directly and C++ by libraries. Before modernization, this course explained how to use and how to implement container classes (such as Vector, Queue, and Map). Since implementing these data types is already taught in Data Structures and Data Structures is a required course, we decided to eliminate the content about implementing container classes.

Parallelism is introduced after the coverage of performance and profiling. In this fifteen-week course, parallel computing is introduced during the tenth, eleventh, the twelfth weeks. We cover the following topics: (A) Conceptual parallelism. We start with an example of doing laundry in an apartment laundry room: there are many washers and dryers (processing units). To keep these processing units busy, there must be enough loads of clothes

(data). Then, we explain that washers and dryers are heterogeneous computing units and the concept of pipelining is introduced. (B) Synchronization and deadlock. We use a real-life example, withdrawing cash from two ATM machines from the same bank account, to explain the concept of atomicity. The students are motivated to learn about synchronization. We then provide multiple examples showing coarse and fine-grained synchronization and describe deadlocks that can occur when locks are not acquired in correctly. (C) Amdahl's Law. We explain why the performance of sequential parts is important even in parallel programs. We evaluate students' understanding using programming assignments and exam questions. We have at least two assignments and one exam about parallel programming. Students write programs that create multiple threads and then compare the execution time using different numbers of threads on multi-core (4, 8, and 16 core) computers donated by Intel. Students also write programs using threads in networking: servers respond to multiple clients by creating one thread for each client.

III. Advanced C Programming

This is the second programming course for students in Computer Engineering, after taking an introductory programming course using C and MATLAB. This course is modernized more recently because it is the prerequisite of all other Computer Engineering courses. We have to ensure that the changes in this course would be properly integrated in the other courses. This course covers memory management and file operations. It then moves on to recursion and basic data structures such as linked lists and binary trees. This course used to have two credit hours, which limited the amount of material that can be covered. In the Fall of 2012, one credit hour was added. Some materials are moved from the follow-up course on Data Structures so that we can introduce parallelism in Data Structures in future semesters. We received an early adopter grant from NSF/TCPP for introducing parallel computing into this course. In the fall semester of 2012, we introduced the concepts of parallel computing using multiple processors and partitioned data. In the spring semester of 2013, we continue our modernization of this course to allow a greater focus on parallelism and a better evaluation of the students' grasp of important concepts. We plan to use two programming assignments:

1. Pixel-wise color inversion of images
2. Subset sum.

The first assignment partitions each bit-map image into non-overlapping blocks and each thread changes the pixels in a block. Since no data is shared among the threads, data protection is unnecessary. In order to obtain observable speedup by parallel computing, the images need to have high resolutions. Locality is not covered in this course and we will use the second assignment that requires large amounts of computation with small amounts of data. Many NP-complete problems have this property. This assignment also introduces the concept of a shared variable. This variable can be set by one or several threads if these threads find the subsets whose sums equal to the given number. In addition to programming assignments, we will also create non-programming assignments asking students conceptual questions, for example, the differences between exclusive resources vs. non-exclusive resources.

IV. Introduction to Digital System Design

This is the first hardware course for Electrical and Computer Engineering students. This course analyzes and designs digital logic using CMOS gates. Parallelism is introduced through a carry look-ahead (CLA) adder circuit. Its performance is compared with that of a ripple adder circuit. Students construct and examine the overall organization of a CLA, and determine the worst case propagation delay incurred by a practical (Programmable Logic Device-based) realization of a CLA.

V. Discussion

We discover that many students readily accept the concepts of parallel computing. Before we started adding parallel computing into the curriculum, we were concerned that students would have difficulty understanding the concepts after they have learned sequential programming in the first year. Fortunately, this problem was not observed. We observe that students already know multicore processors are prevalent and want to explore the potential of the hardware. Our second observation is that most students can understand the relationships between speedup, the number of threads, and the number of cores. We have different computers with different numbers of cores. Students notice that when they use more threads than the number of cores, programs' performance does not improve. Locality plays important roles in performance. Students learn the concept of locality later in Computer Organization (a senior-year course). Many students discover that performance does not improve proportionally as the number of threads if data sizes also grow. This is surprising and somewhat frustrating for some students because they do not understand the effects of locality yet. This also motivates the students to learn more about the architecture aspects of parallel computing.

Our approach in teaching parallel computing adopts curricular (multiple course) integration versus offering a single new course. This can provide a bigger picture of parallel computing as well as an opportunity to review and connect concepts from previous and upcoming courses. As part of the curriculum modernization, we have also developed a concept inventory for parallel computing. The purpose is to ask the same questions throughout their four-year studies at Purdue. For this purpose, we have to avoid using any terms that can be precisely defined but are taught later in the curriculum. Thus, to make the concept inventory applicable, we must be careful to make the questions clearly and precisely defined, while using words that are understandable for all undergraduate students.