Integrating Parallel and Distributed Concepts into the Undergraduate Curriculum: Introducing Parallel Concepts through Sorting

Eileen Kraemer
Computer Science Department
The University of Georgia
Athens, GA USA
email: eileen@cs.uga.edu

Keywords-parallel, distributed, curriculum, concurrency

I. INTRODUCTION

The goal of the NSF-TCPP Workshop on Parallel and Distributed Computing Education is to address the challenges faced by educators who seek to incorporate into the undergraduate curriculum the concepts, tools, and skill sets required for parallel and distributed computing (PDC). The Computer Science Department at the University of Georgia will incorporate these concepts into the undergraduate curriculum using an integrated approach that spans 4 core undergraduate courses and will be phased in over 3 semesters. The University of Georgia is a research-intensive university; the CS department has a faculty of 21 and enrolls roughly 250 undergraduate majors, 60 MS students and 60 PhD students.

Adaptation of the courses to include concepts in parallel and distributed computing will involve a major revision of the curriculum in the C++ and Unix Systems Programming class, and the introduction of several guest lectures in the other three courses. Our goal is to have both the revised course and the content of the guest lectures become incorporated into the regular curriculum.

II. COURSE DESCRIPTIONS AND PDC TOPICS

A. Introduction to Computing and Programming

This introductory course for majors also fulfills the core “Quantitative Reasoning” requirement for all undergraduate students, with the result that we see a mix of roughly half CS majors and half non-majors. The course addresses fundamental techniques of program development, focuses on the object-oriented programming paradigm, introduces students to the use of supporting software tools, and uses the Java programming language. The course also includes some basic data structures (arrays and linked lists) and algorithms (searching and sorting).

In the spring semester of 2011, guest lectures were presented in all three sections of the CSCI 1301 course, which enrolls approximately 175 students. The lectures took place near the end of the semester (week 14 of 15) and used a demonstration in which the students were asked to act out several sorting algorithms. A hands-on analysis of the cost of the sorting algorithms motivated the notion of parallel computing. Parallel and distributed computing was defined, and costs and benefits were evaluated. Details of the approach, materials and evaluation follow in sections III and IV.

B. Software Development

Beginning in the fall semester of 2011, additional PDC topics will be integrated into the Software Development (CSCI 1302) course, the second course for majors. This course focuses on software development techniques in an object-oriented programming language (Java) and emphasizes systems methods, top-down design, testing, modularity, and structured techniques. PDC topics to be introduced include multi-threading, synchronization and mutual exclusion, implementation of parallel and distributed searching and sorting algorithms, design and implementation of a simple client-server system, and discussions of cluster computing, cloud/grid computing, and peer-to-peer computing.

Online course materials have been developed and the current version can be found at: http://www.cs.uga.edu/~eileen/Concurrency_tutorials

The online modules on “Concurrency Concepts” and “Implementing Concurrency” are relevant for this course and are being refined for use in the fall 2011 offering. One lecture will introduce the materials and discuss a programming project to implement a parallel sort algorithm. Another lecture will discuss client-server, p2p, and cluster and grid computing and introduce a programming project involving a simple-client server system.

C. Data Structures

The Data Structures (CSCI 2720) course enrolls approximately 45 sophomore-level CS majors. The course deals with the design, analysis, implementation, and evaluation of the fundamental structures for representing and manipulating data. Abstract data types and algorithm analysis of searching and sorting algorithms are emphasized.

In the spring semester of 2012 we will integrate guest lectures on the complexity analysis of parallel and distributed
algorithms. This more formal complexity analysis will build on the informal approach employed in the CSCI 1301 course in the spring of 2011, which relied on sketches and manual counts of operations.

D. C++ and Unix Systems Programming

The C++ and Unix Systems Programming course (CSCI 1730) serves the dual purpose of introducing students to a second programming language (C++) and providing a more detailed look at the operation of computing systems. The course typically enrolls approximately 45 sophomore-level CS majors. An initial unit on “C++ for Java programmers” familiarizes students with the use of C and C++ in systems programming and introduces templates and file processing. The “Systems Programming” unit covers the basics of UNIX systems programming, including file systems and directory structures, basic and advanced file i/o, process creation, and interprocess communication such as signals, pipes/fifos and sockets.

In the spring semester of 2012 we will conduct a version of this course in which the curriculum has been revised to focus on topics in parallel and distributed computing. We will make use of the online course materials we have already developed and will develop additional modules focused on the use of pthreads, ACE, and MPI. This course will place heavy emphasis on hands-on implementation of classic problems in synchronization and concurrency using low-level Unix systems primitives, C++ libraries, pthreads, and MPI. Students will implement programs using 1) threads and shared memory; 2) multiple processes on the same machine communicating via pipes and fifos; 3) multiple processes on different machines communicating via sockets or message-passing. Students will be introduced to tools for the detection of concurrency defects. A final project will involve evaluations of these different approaches for performance, and ease of implementation, testing, and debugging.

III. Report on Progress to Date

Lecture 1 in the CSCI 1301 course took place during week 14 of the spring 2011 semester, and was incorporated into the portion of the course focused on algorithms for searching and sorting. At the beginning of class, students were asked to complete a survey of their knowledge of parallelism, sorting, and time complexity concepts.

To motivate the notion of “cost” of search (time complexity) students were asked to engage in a “Human BubbleSort”. The algorithm for BubbleSort was written on the board. Students took up the roles of Processor, Array Element, Comparison counter, Swap counter, and Step counter. Eight Array Elements were each asked to hold up a sign with their element’s value and to stand up at the board. The Processor was tasked with directing the students to compare values and/or change places, according to the algorithm. Blue painter’s tape was used to mark a box on the floor, in which the processor stood. The Comparison counter was issued a bag of 50 tootsie rolls; one tootsie roll was paid out for each compare. The Swap counter was issued a bag of 50 chocolate eggs; one egg was paid out for each swap. The Step counter kept a tally of each step in the algorithm.

After the algorithm completed, we counted out how many tootsie rolls and chocolate eggs had been distributed and then constructed a sketch of the progress of the algorithm, using an X for each compare and a double-X for each compare-plus-swap that occurred, and placing these in one row for each iteration of the loop in the algorithm. We used the sketch to convince ourselves that BubbleSort is an $n^2$ algorithm.

Next, students acted out Mergesort in a similar fashion, and a complexity analysis was performed based on candy consumed and a sketch on the board. We found that Mergesort is an $n \log n$ algorithm.

Students observed that in Mergesort the Array Elements spent much of their time waiting around, and the idea to use more processors was generated. The Processor was given several rolls of blue tape, representing the ability to spawn another process (mark off a box on the floor and assign a new student to also take on a Processor role), to handle a selected sub-array. Parallel mergesort was born.

The parallel version was acted out. The Comparison counter, Swap counter, and Step counter performed as before. A post-demonstration analysis showed that the number of comparisons and swaps was the same for the parallel version of Mergesort as for the sequential version, but that the overall number of steps were reduced because multiple comparisons and/or swaps occurred at the same timestep.

A discussion of the significance of these findings followed. Students were then asked to complete a post-test.

IV. Evaluation on Progress to Date

Pre-test and post-test data has been collected on paper forms. Data entry of student responses is in progress. Analysis and reporting will follow shortly. However, early indications show that students gained new insights into both the nature of parallel computing and the limitations of additional parallelism.

Videos of the lecture and the students acting out the sorting algorithms were recorded. Portions of these videos will be incorporated into future lectures.

For each of the other other courses, several types of evaluation instruments are in development: 1) a pre-test/post-test evaluation of parallel and distributed concepts for each new module; 2) rubrics for evaluating the implementations produced in the programming projects, focusing on the level of mastery of PDC concepts; 3) online surveys for assessing subjective satisfaction and for soliciting input on how the material could be improved.