J. Parallel Distrib. Comput. 105 (2017) 18–30

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Pedagogy and tools for teaching parallel computing at the sophomore undergraduate level

Max Grossman *, Maha Aziz, Heng Chi, Anant Tibrewal, Shams Imam, Vivek Sarkar

Department of Computer Science, Rice University, 6100 Main St, Houston, TX 77005, United States

HIGHLIGHTS

- An overview of parallel computing pedagogy at Rice University, including a unique approach to incrementally teaching parallel programming: from abstract parallel concepts to hands-on experience with industry-standard frameworks.
- A description of the HJlib parallel programming library and its applicability to parallel programming education.
- A description of the motivation, design, and implementation of the Habanero Autograder, a tool for providing automated and immediate feedback to students on programming assignments.
- A discussion of unexpected benefits from using the Habanero Autograder as part of Rice University's core parallel computing curriculum.

ARTICLE INFO

Article history:
Received 15 June 2016
Received in revised form
5 December 2016
Accepted 30 December 2016
Available online 9 January 2017

Keywords:
Autograding
Parallel
MPI
Java
JVM
Education
Multi-threading
Tools
Pedagogy

ABSTRACT

As the need for multicore-aware programmers rises in both science and industry, Computer Science departments in universities around the USA are having to rethink their parallel computing curriculum. At Rice University, this rethinking took the shape of COMP 322, an introductory parallel programming course that is required for all Bachelors students. COMP 322 teaches students to reason about the behavior of parallel programs, educating them in both the high level abstractions of task-parallel programming as well as the nitty gritty details of working with threads in Java.

In this paper, we detail the structure, principles, and experiences of COMP 322, gained from 6 years of teaching parallel programming to second-year undergraduates. We describe in detail two particularly useful tools that have been integrated into the curriculum: the HJlibparallel programming library and the Habanero Autograder for parallel programs. We present this work with the hope that it will help augment improvements to parallel computing education at other universities.

1. Background

At Rice University, COMP 322: Fundamentals of Parallel Programming is the introductory parallel programming course for all undergraduates. As a required course for both the Bachelor of Arts and Bachelor of Science Computer Science degrees, COMP 322 sees approximately one hundred students each Spring semester from a wide range of backgrounds and interests. Additionally, COMP 322 is taken early in the curriculum, most commonly during the Spring semester of students' second year of the undergraduate program. Hence, many of the assumptions that later parallel programming courses make about prerequisites do not hold true for COMP 322.

In this paper we use our experiences teaching COMP 322, assisting in COMP 322 as teaching assistants, and taking COMP 322 as students to offer insights in to effectively teaching parallel programming to students early in the undergraduate CS curriculum. We also describe two tools that have become fundamental parts of COMP 322 since their introduction, and present qualitative and quantitative evidence of their effectiveness in teaching parallel programming to second-year students.

1.1. An overview of COMP 322

COMP 322 introduces students to several basic concepts of parallel programming and parallel algorithms. It follows a pedagogic approach that exposes students to intellectual challenges in

* Corresponding author.
E-mail address: jmaxg3@gmail.com (M. Grossman).

http://dx.doi.org/10.1016/j.jpdc.2016.12.026
0743-7315/

parallel software without enmeshing them in jargon and lower-level details of today's parallel systems.

The prerequisites for COMP 322 require that students have taken three introductory courses in Computer Science prior to enrolling.

The first, COMP 140: Computational Thinking, focuses on teaching students fundamental skills for writing programs and reasoning about their behavior. As an introductory course, COMP 140 focuses on Computer Science practicum using a case study-based approach. Programming constructs and techniques are introduced using simple but interesting algorithms and applying them to real-world problems (e.g. Markov chains to compose poetry). COMP 140 is taught in Python.

The second course, COMP 182: Algorithmic Thinking, teaches students a rigorous approach to computational and scientific problem solving by "(1) understanding the problem; (2) formulating the problem mathematically; (3) designing an algorithm; (4) implementing the algorithm; and (5) solving the original scientific problem" [20]. COMP 182 is students' first rigorous introduction to algorithms (e.g. dynamic programming, sorting algorithms) and algorithmic asymptotic complexity analysis (e.g. Big-O notation). COMP 182 is also taught in Python.

The third course, COMP 215: Introduction to Program Design, is students' first introduction into software engineering and software design. COMP 215 introduces students to functional programming with Java 8, lambdas, object-oriented programming, testing (e.g. unit tests and other tools for debugging software), performance optimization techniques, and additional algorithms not covered in COMP 182. COMP 215 is taught in Java.

Additionally, it is common to take the introductory systems course (COMP 321: Introduction to Computer Systems) in parallel with COMP 322. One unfortunate side effect of this is that students often come in with little or no Unix shell experience, which presents a challenge when working with parallel compute clusters. Students also often have little understanding of the memory hierarchy or processing units of the hardware they use, topics which are closely related to parallel program performance.

In addition to these courses in Computer Science, students are likely to have taken introductory calculus and linear algebra courses before enrolling in COMP 322.

COMP 322 itself covers a variety of parallel programming patterns, including fork-join, futures, SIMD parallelism on accelerators, and SPMD parallelism. COMP 322 covers several abstract and concrete methods of synchronization, including barriers, mutexes, phasers, locks (reentrant and read–write), Java's `synchronized` statement, and Java concurrent collections. COMP 322 focuses on shared-memory parallel programming, but also touches on distributed memory programming using programming systems like MPI (for message-passing), UPC (for PGAS), and Apache Spark (for distributed functional programming) as examples. Of course, emphasis is placed on universal challenges in concurrency, such as deadlock, livelock, and data races. In addition to teaching students how to use parallel programming frameworks, COMP 322 also covers how many of those frameworks are implemented by describing different task scheduling policies (e.g. work-sharing, work-stealing, work-first vs. help-first). Because the prerequisites of the course will only have exposed students to Python and Java, all assignments are completed in Java.

COMP 322 is a 14 week course. Each week of COMP 322 includes three fifty-minute lecture sessions, with the last 10–20 min of each lecture dedicated to completing an in-class worksheet. Hence, COMP 322 is a partially flipped classroom. Students are required to complete several short quizzes each week based on videos available online, and the majority of weeks includes an evening 2-hour lab session during which students get immediate hands-on experience with the topics covered in class. In the Spring 2016 semester, there were a total of 13 labs. Two large exams are given, one in the middle of the semester and one at the end. In Spring 2016, the midterm exam was given during Week 7 and the final exam was given the week after Week 14.

For this past Spring 2016 semester, students completed five homework assignments each of which included both a written portion and a programming portion. In general, students are given between 2 and 5 weeks to complete each assignment. The programming portions for the Spring 2016 semester included:

1. Parallelizing a sorting algorithm.
2. Parallelizing a matrix multiply operation, with artificial overheads inserted to illustrate the concept of chunking work.
3. Parallelization of a genome alignment algorithm, with an emphasis on task-parallel programming.
4. Parallelization of minimum spanning tree construction, with an emphasis on programming with the Java Threads APIs.
5. Parallelization of pi estimation on a distributed system using OpenMPI's Java APIs.

Homeworks 3 and 4 included intermediate checkpoints to ensure students are making progress on the assignment. For example, HW 3's first checkpoint was a parallel implementation of a genome alignment algorithm but no grade was given on real-world performance, only on the abstract parallel metrics of their implementation (more details on abstract metrics are provided in Section 3). HW 3 Checkpoint 2 asked for a well-performing parallel implementation that demonstrated reasonable speedup. The third and final checkpoint for HW 3 asked for a parallel implementation that was space-efficient as well, essentially implementing wavefront-style parallelism across a grid without storing the whole grid at once.

In COMP 322's rubric, 40% of a student's grade goes to homeworks, 40% to exams (20% each), 10% to hands-on labs, and 10% to general participation (which includes attendance at labs, active participation in class, and discussion on the course forums).

Before Spring 2016, programming assignments in COMP 322 were entirely manually graded by teaching assistants (TAs) for correctness, performance, style, and documentation. Students received limited feedback as they progressed through the assignment because TAs graded their homework only after the submission of a final version. This process was inefficient, tedious, error-prone, and opaque to students. Manual grading takes a significant amount of time, which (1) increases the latency between a homework being submitted and being returned with feedback, resulting in students repeating the same mistakes in labs and subsequent assignments, and (2) reduces the time the teaching staff can spend on mentoring students.

1.2. COMP 322 pedagogy

COMP 322 strikes a balance between (1) providing students with the ability to reason about parallel execution in a way that crosses programming model boundaries, and (2) offering them hands-on experience with practical and popular frameworks for writing parallel programs. At a high level, COMP 322 accomplishes this in four stages. First, students are given theoretical tools for reasoning about the behavior and performance of their parallel programs. Then, students gain hands-on experience with a programming framework whose APIs closely match those abstractions. Next, students have the chance to move further away from these abstractions and use standard, closer-to-hardware programming frameworks which are more commonly used in practice. Finally, students are given a survey of alternative models and special-purpose frameworks as a taste of other fields in parallel computing.