Preparing Computer Science Students for an Increasingly Parallel World: Teaching Parallel Computing Early and Often

Martin Burtscher
Texas State University
burtscher@txstate.edu

Wuxu Peng
Texas State University
wuxu@txstate.edu

Apan Qasem
Texas State University
apan@txstate.edu

Hongchi Shi
Texas State University
hs15@txstate.edu

Dan Tamir
Texas State University
dt19@txstate.edu

1. THE PROBLEM

The ubiquity of parallel computing resources presents a pressing challenge to the entire computer science discipline. The thrust of this challenge is to find ways to better equip computer science students with skills to face an increasingly parallel world. Although there is general agreement that undergraduates should learn parallel computing concepts, there is debate about when parallel programming should be taught and to what extent. Recently, the NSF/IEEE-TCPP PDC committee and ACM have emphasized the need for integrating parallel computing topics across the curriculum [1, 3]. Although these initiatives have garnered strong support from the community, there remain key challenges in realizing this vision. The pedagogy of teaching current PDC topics to undergraduates is yet to mature and major curriculum revisions are problematic, particularly for departments in larger universities where revisions to the curriculum require significant planning and effort including training of faculty teaching lower-level classes, complying with administrative policies of the university curriculum board and tracking graduation credits for majors under the revised curriculum.

2. OUR APPROACH

This paper presents a systematic approach of integrating parallel computing into the curriculum through a series of short, self-contained modules. Modeled after the early-and-often strategy advocated by the CSinParallel project [2], our approach provides several pedagogical advantages. We discuss three key advantages here.

Introduce parallel topics at the right level of abstraction: To gain mastery in parallel programming (and sequential programming, for that matter), students need to learn how to think about problems at different levels of abstraction and acquire the ability to switch between levels rapidly. It is very important to determine the right level of abstraction for introducing different aspects of parallel problem solving. Exposing students to multiple levels all at once is likely to create confusion. Moreover, choosing too low a level may hide some of the natural parallelism available in an algorithm and result in lost opportunity. We advocate an approach that starts with the most abstract forms of concurrency and parallelism, and progressively reveals lower-level mechanisms required for more complex forms of process interaction. For example, students can learn about Amdahl's law for parallel programs without being able to program in parallel or having knowledge of synchronization and communication primitives. Concepts that students can grasp at a higher level of abstraction are introduced first and reinforced in subsequent years as students are gradually exposed to lower-level concepts. Finally, we tie these ideas together in the form of a capstone course at the senior level. Some topics, such as performance of parallel programs, span multiple levels of abstraction and are therefore part of several course modules.

Provide "parallel context" to key topics in the existing curriculum: Many theories and concepts covered throughout the CS curriculum can enhance a student's comprehension of parallel computing principles. However, such topics are often not taught in a parallel context. For example, almost all data structures courses introduce recursion, and in many cases, a divide-and-conquer algorithm is used as a primary example. Yet, the fact that divide-and-conquer algorithms naturally lend themselves to parallelism is rarely emphasized. Similarly, in later algorithms courses, the complexity analysis of divide-and-conquer traditionally ignores parallel implementations. Our approach incorporates parallel context to key ideas such as divide-and-conquer algorithms and recursion. Since the modules are dispersed over several courses and do not introduce completely new concepts but rather extend topics that are already being covered, we do not expect these modules to cause significant strain on covering the original content. What material, if any, to de-emphasize in a course to make room for the module is decided by the individual instructor.

Encourage adoption across different institutions: The modules we are developing are self-contained with lecture notes, assignments, exercises, exam questions, and solutions. The estimated duration for each module is between one to four 1.25-hour lectures. Some modules include lab time. Al-

SC'13 Nov 17-22, Denver, Colorado, USA

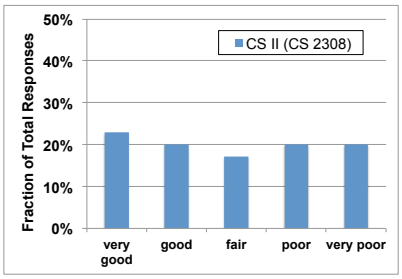

Figure 1: Student Learning Outcome in CS II (CS2308)

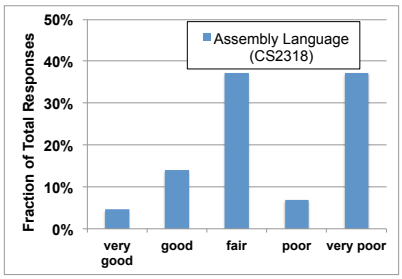

Figure 2: Student Learning Outcome in Assembly Language (CS2318)

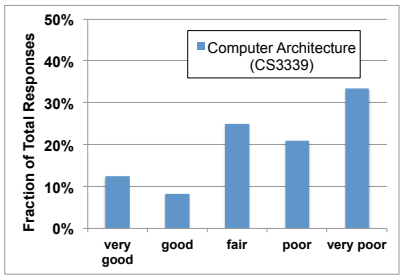

Figure 3: Student Learning Outcome in Computer Architecture (CS3339)

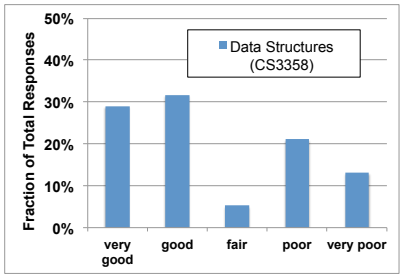

Figure 4: Student Learning Outcome in Data Structures (CS3358)

though the modules provide a textbook treatment of the material, they are not tied to any specific textbook or lab manual. This enables straightforward adoption of the modules at other institutions. The course modules are designed so that they are mostly language independent. Of course, some modules require the use of specific parallel languages or APIs. In these cases, we develop multiple modules for the same concepts using alternate language interfaces. Generally, modules do not entail any prerequisites other than the ones prescribed for the course in which the module is taught. Nevertheless, introduction of some modules may need to be postponed until later in the semester when the requisite material has been covered in the course.

3. STATUS

To date, we have developed six course modules covering topics of parallelization techniques, intra-core parallel architecture, inter-core parallel architecture, task orchestration - synchronization and communication, thread scheduling and mapping, and parallel performance. A web site has been set up where all module-related material is made freely available [4]. Five modules were introduced in undergraduate classes at Texas State University in Spring 2013. Another five are being taught in Fall 2013.

We instated two forms of evaluation during the first year. The assessment plan for student learning outcome was designed by the involved faculty whereas teaching effectiveness and student engagement was evaluated through an independent external evaluator. Additionally, we started collecting and compiling data for a longitudinal study of student understanding of parallel concepts. The initial results of that study are expected in 2015.

Figs. 1-4 present student performance on final exam questions for each course where a module was implemented. Evaluation of student learning outcomes shows that 60% of the students who were exposed to the PDC modules received a passing grade on the final exam question. This number indicates that our initial implementations of the modules were relatively successful. However, the individual course-based breakdown of student performance identifies problem areas that will need to be addressed in future semesters.

4. ACKNOWLEDGEMENTS

This work was funded by the National Science Foundation under grants DUE-1141022, CNS-1253292 and CNS-1217231.

5. REFERENCES

[1] ACM IEEE Joint Task Force on Computing Curricula, Computer Science Curricula 2013 Strawman Draft. 2012.

[2] CS in Parallel Project. `http://csinparallel.org`, 2013.

[3] NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing. `http://www.cs.gsu.edu/~tcpp/curriculum`, 2013.

[4] Parallel Computing in the Undergraduate Curriculum : the Early-and-Often Approach. `http://tues.cs.txstate.edu`, 2013.