Enhancing Parallel and Distributed Programming Through Software Engineering

Xinghui Zhao
School of Engineering and Computer Science
Washington State University
Vancouver, WA 98686

WSU Vancouver is a small regional campus and part of the greater Washington State University system. As of 2015, there were over 3300 students enrolled in WSUV, of which 54% are women, and 23% are students of color. Our computer science department consists of seven full time faculty and offers ABET-accredited BS and MS degree programs. Starting from Fall 2013, we have been gradually integrating parallel and distributed computing (PDC) topics into our curriculum, at multiple levels. Among them three are offered Fall semesters (CS 320 Fundamentals of Software Engineering, CS 447 Computer Game Design, CS 580 Concurrent Programming), and two are offered in Spring semesters (CS 122 Data Structures, CS 580 High Performance Computing). In this poster, we present our recent efforts in integrating parallel and distributed programming and software engineering concepts in relevant courses, aiming to provide hands-on experiences for applying PDC techniques in substantial software systems.

I. Integrating Parallel and Distributed Programming in Software Engineering Courses

With the growing ubiquity of multicore architectures, distributed systems, and cloud computing, parallel and distributed programming is considered a required skill to write modern software applications. As a result, software engineering becomes the main course where PDC topics can be naturally introduced.

A. CS320: Fundamentals of Software Engineering

CS320 is the first software engineering course in our curriculum. In this course, students are required to develop user-interactive software in teams. We encourage students to choose to implement software which has distributed components, such as client-server type of software, and software with a remote database. This enables us to naturally integrate PDC topics along with the progress of the projects. In addition, the students will have more incentives to learn these topics, because they find the knowledge useful for their projects. Specifically, when requirements engineering is taught, we introduce non-functional requirements which are related to distributed systems, such as scalability, reliability, and security. Then we explain why these requirements are important for a software that is distributed on multiple computers. In architecture design, we discuss architectural patterns for distributed systems, client-server and peer-to-peer communication, and software-as-a-service (cloud computing). When students are working on implementation, we introduce basic concepts of multi-threaded programming, which most of the students find useful not only for their projects, but in general. In the software testing/debugging phase, we discuss key issues and hard-to-debug errors in concurrent programming, such as race condition, deadlock, livelock etc. We believe naturally integrating PDC topics into every phase of their software development process helps students to learn these topics through practice.

B. CS 420: Software Design Projects

CS420 is the capstone course in our curriculum. This capstone course brings together all of the computer science coursework the student has taken, along with writing and communication skills acquired in non-CS courses, and gives the students an opportunity in which they can demonstrate their skills and creatively design a substantial software system. All the programming language knowledge, algorithms, data structures, and application domain knowledge the students have acquired are brought to bear on a team project. The overall intent of the course is to give students an experience that is comparable to conducting a software development project in a professional setting. The capstone projects provide a great opportunity for the students to practice their skills in parallel and distributed programming. Since 2013, all capstone projects require some level of PDC knowledge. A typical capstone project consists a front-end application, and a back-end server. The front-end can be either a mobile application or a web application, and the back-end server usually involves a database which provides remote data access for the front-end. These relatively large projects enable the students to develop parallel and distributed software in a team setting.

II. Integrating Software Engineering in Parallel and Distributed Programming Courses

Besides integrating PDC topics in software engineering courses, we also utilize software engineering projects in PDC focused courses, so that students can have hands-on experiences in practicing the skills they learn from the courses.

A. CS 580: Concurrent Programming

Concurrent Programming is a graduate level course, which specifically focuses on multi-threaded, parallel, and distributed

programming. This course covers many PDC topics, such as parallel programming for multicores (shared memory), and distributed programming (message passing). It also covers some advanced topics, such as theory of concurrency, power consumption of multicores, and cloud/grid computing. The course begins with an introduction to multicore architectures. We discuss Moore's law, as well as its impacts on architecture design. The fact that computer architects are shifting their design to multicores implicitly requires programmers to write concurrent programs. We found introducing concurrent programming in such a context better motivates students to learn this technique. Then we continue to discuss programming on multicores, and introduce key concepts in shared-memory programming, such as critical sections, atomic actions, and synchronization techniques, such as locks, barriers, semaphores, and monitors. We then encourage students to picture a system in a larger scale, which is a distributed system. We discuss message passing, RPC, Rendezvous, and introduced programming tools like MPI. We also cover some theoretical topics in concurrency, such as Actor model, and CSP. In this course, students are asked to propose, design, and develop a software system in which multiple concurrent programming techniques are used.

B. CS 580: Distributed Systems

Distributed Systems[1] is a graduate level course, which focuses on concepts, theories, and techniques of building distributed systems. PDC topics are a major component in this course. It covers various characteristics and challenges of distributed systems, such as heterogeneity, distribution transparency, fault tolerance, security, and scalability. Students also learn practical programming techniques in this course, including socket programming, parallel programming, RPC, and MapReduce. In this course, students are asked to design a distributed software system from scratch, while they are learning corresponding concepts of distributed systems. These course projects usually utilize the client-server architecture, and have an underlying distributed file system to provide data service to the application. Through the projects, students gain practical, hands-on experiences in dealing with the challenges associated with distributed systems.

III. Evaluation

To maintain our programs ABET accreditation, we have installed a nuanced assessment plan for each of our courses. This assessment plan are leveraged to integrate an evaluation plan for PDC topics in the relevant courses.

Particularly, at the end of each term, the instructor must produce a Faculty Course Assessment Report (FCAR). In the FCAR, the faculty member associates each Measured Course Outcome listed on the syllabus with the method(s) of evaluation (e.g., a particular set of homework questions, specific exam questions, project requirements, etc.). For instance, a Measured Course Outcome in Distributed Systems is to Write distributed programs using different techniques / languages. This particular outcome is assessed by a course project, and questions on the homework and final exam. Based on the students' performance on the evaluation medium, the instructor can reflect on her/his teaching and also, offer an overall assessment of the specific course outcome.

Because we believe that such rigor in evaluating PDC outcomes must also be taken, we have created a set of outcomes which integrates software engineering knowledge and parallel and distributed programming techniques for the courses listed above. These outcomes are evaluated in the same manner as ABET course outcomes in the course offerings since 2013. The feedback from students in in the past three years was positive, which was encouraging. We will continue to work on the integration of software engineering and parallel and distributed programming in the future years.

IV. Conclusion

To curtail power consumption, chip design has shifted towards the multi- and many-core architecture. As a result, parallel and distributed computing (PDC) is no longer niche, but an essential ingredient in all aspects of computer science. Together with the need to solve increasingly data-intensive problems, the multi-core shift has created a surge in the demand for a parallel and distributed computing workforce. The computer science faculty at WSU Vancouver understand the value of adopting a PDC curriculum to best prepare our students for tackling next-generation problems.

In the past three years, we have been integrating PDC topics into multiple CS courses at different levels. In every semester, we provide at least one lower level and one upper level course containing PDC concepts. Along with this integration, we also investigate the possibilities of enhancing parallel and distributed programming through software engineering related courses and projects. PDC related topics are integrated in software engineering courses, including both the theoretical software engineering and practical capstone software design courses. In addition, in PDC focused courses, students are asked to develop substantial software systems in which they practice both PDC and software engineering techniques. Additional course outcomes related to this integration are added to relevant courses, and evaluated at each semester in the same manner as ABET course outcomes.

V. Travel Budget

The travel fund is essential for us to attend EduPar 2016 and share our experiences and results of the integration efforts. The proposed travel budget is as follows: 1) airfare from Portland to Chicago: $500; 2) Conference registration: $600; 3) Hotel: $1,200; 4) Local transportation: $200. The total amount of the travel budget is $2,500.

Acknowledgement

The work presented here is supported by the NSF/TCPP Curriculum Early Adopter Award.

[1]CS580 is a course number dedicated for special topics. Both Concurrent Programming and Distributed Systems used this course number in 2014-2015.