Integrating PDC Topics in Multiple Levels of CS Courses at WSU Vancouver

Xinghui Zhao, David Chiu, and Scott Wallace
School of Engineering and Computer Science
Washington State University
Vancouver, WA 98686

WSU Vancouver is a small (∼3000 students) regional campus and part of the greater Washington State University system. 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 multiple courses, at multiple levels in the curriculum. Among them three were offered in Fall 2013 (CS 320 Fundamentals of Software Engineering, CS 447 Computer Game Design, CS 580 Concurrent Programming), two are currently being offered in Spring 2014 (CS 122 Data Structures, CS 580 High Performance Computing), and in addition we also propose to integrate PDC topics in a course that is going to be offered in Spring 2015 (CS 453 Web Data Management).

I. Courses Taught in Fall 2013

A. CS 320: Software Engineering

Software Engineering is one of the core CS courses. 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. In this way, students are better motivated and also more interested in learning these topics.

B. CS 447: Computer Game Design

In Computer Game Design, we incorporated multi threading and synchronization as well as a discussion of parallel path planning. The discussion on multi threading takes place in the context of building a simple infrastructure to support networked games. We discuss the Java memory model and the guarantees of the synchronized and volatile keywords. In addition, students are introduced to the happens-before relationship and are asked to demonstrate an understanding about how synchronization and use of volatile affect potential instruction orderings. The discussion of parallel path planning comes at the end of classical (sequential) path planning ala Dijkstra's algorithm and A*. The discussion hinges mainly on potential methods for parallelizing path planning algorithms as discussed in the literature as well as potential issues (bottlenecks) with these implementations. In the latest offering, we discussed a parallel version of Dijkstra's algorithm along with "Probabilistic Roadmaps" which are trivially parallelizable.

C. 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.

Students are asked to work on several programming-intensive assignments, including programming with Actors and

message passing, programming with MPI and C, developing critical section solutions using locks and barriers, and writing a simulated multiprocessor kernel.

II. Courses Being Taught in Spring 2014

A. CS 122: Data Structures

Data Structures is required course for freshman, and it is the first course in our curriculum that is integrated with PDC topics. We believe that introducing these topics at an early stage helps students understand the concepts better. In this course we include additional treatment on floating point numbers including range, precision and the IEEE 754 representation. This includes new lecture materials and in-class demonstrations. In addition, we also include a treatment of parallel computation as related to divide and conquer algorithms and recursion. These topics are introduced in the context of sorting, and if time permits, selection.

B. CS 580: High Performance Computing

High-Performance Computing (HPC) is a graduate level course, but excellent undergrads were allowed to take this course.

The course begins with a very high-level overview of instruction level parallelism (ILP) including pipelining, out-of-order execution, superscalar execution. The goal of this overview is to raise awareness of these lower-level CPU issues, since our curriculum lacks an advanced architecture course, where these topics would be taught in detail. The overview also helps students understand the challenges of the memory wall and power wall, and how these motivate the multi-core architecture and next-generation systems. From ILP, the course moves on to a much deeper treatment of cache architectures, loop optimizations, loop dependence analysis, coherence protocols, and false sharing. Next, parallel architectures are introduced, which includes both tighly-coupled (multi-core, GPUs) and loosely-coupled (clusters), including the appropriate networks (torus, hypercube, omega, etc.) on which they are supported. Finally, we move towards programming these systems using OpenMP for shared-memory processing, and MPI and Hadoop for distributed memory processing.

Students are evaluated over three major components: homework assignments, term project, and a teaching plan. The assignments are mostly problem solving and programming over two of our cluster environments (an HPC cluster supports MPI over infiniband, and a high-memory Hadoop cluster). For the term projects, students must pitch an approved project idea, which uses HPC concepts. Students must generate and present results at a research poster session. Finally, students are asked to teach a 50 minute course on an HPC topic of their choice. Their teaching plan must be coherent, and their teaching will undergo peer-evaluation.

III. Courses to be Offered in Spring 2015

A. CS 453: Web Data Management

In Web Data Management, the focus of the course is to walk through the phases in creating a search engine for the web. The distinct PDC topics are taught and practiced include (1) Synchronization (producer-consumer) Problem, and (2) cloud/grid computing with Map-Reduce.

For synchronization, the topic was motivated by the application of a web crawler. We discuss the performance of a single-threaded web crawler, and the benefits and challenges of having the task-parallelism of a multi-threaded version. A global queue of URLs is stored, and threads are dispatched to download the web page. The PDC issue involves the synchronizing the classic producer-consumer problem. Students taking this course have not yet been exposed to the Operating Systems course, where they are introduced to synchronization problems. Students' implementations varied from C++ and Java, using semaphores and synchronized classes, respectively. Students are asked to generate performance plots (throughput, speedup) over an increasing number of threads.

Students are then asked to create inverted indices and page rank scores for the crawled data collection. We discuss the problem with Big Data, which causes single-threaded algorithms to thrash virtual memory. The distributed fork-join processing framework is next introduced, with an emphasis on Map-Reduce. Students upload their code on to the Amazon EC2 cloud, and write Map-Reduce algorithms (using Python) to create the required data structures. The students are not asked to analyze the performance for this assignment. Students are also evaluated heavily on producing Map-Reduce algorithms on exams.

IV. 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 Web Data Management is to Analyze or solve problems related to centralized and distributed indexing data structures and using suitable algorithms. 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.

The feedback from students in Fall 2013 was positive, which was encouraging. We will continue to work on the integration in the next academic year.

Acknowledgement

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