Preparing an Online Java Parallel Computing Course

Vivek Sarkar
Rice University
Houston, TX, USA
vsarkar@rice.edu

Max Grossman
Rice University
Houston, TX, USA
jmg3@rice.edu

Zoran Budimlić
Rice University
Houston, TX, USA

Shams Imam
Two Sigma
Houston, TX, USA

***Abstract*—While multi-core platforms are now ubiquitous in all areas of information technology, from enterprise software engineering to mobile app development, parallel computing education is still lagging behind the demand for skilled parallel programmers. At many universities today, parallel and concurrent computing is still not part of the core curriculum because of resistance to major curriculum changes. Many other universities lack the necessary educators or infrastructure to teach a comprehensive parallel computing course. Furthermore, even addressing these issues would do nothing towards supporting software professionals who have already entered the work force and have no plans to return to school.**

To address this broad need for a standalone, publically available, comprehensive, and easily accessible course on parallel computing, we have developed an online offering packaged as a Coursera Specialization on Parallel, Concurrent, and Distributing Computing in Java.

In this paper, we describe the preparations for this online course and the unique challenges we encountered in terms of both curriculum development and technical infrastructure. We describe how lessons learned from an on-campus parallelism course at Rice University helped to shape the Coursera specialization, and summarize our experience with implementing this specialization on the Coursera platform at scale.

***Keywords*-component; formatting; style; styling;**

I. Motivation

Over the last two decades, the number of cores available in commodity hardware has grown significantly. Software engineers across the industry have had to learn to use these multi-core platforms, often while on the job. While there has been a significant push in recent years to improve the state of parallel computing education at the undergraduate level [1][2], these efforts at isolated undergraduate institutions do nothing to support students at 1) other educational institutions without the resources to create their own parallel programming course, or 2) professionals already working in industry who want to broaden their skills.

Recently, Massive Open Online Courses have evolved as an affordable and accessible way to gain a first-class education. They have been particularly popular and effective in teaching programming skils. Some course enrollments have exceeded one million students [3], with the courses offered today on MOOC platforms like Coursera [4], edX [5], and Udacity [6] covering a wide spectrum of Computer Science topics.

We see MOOCs as the best option for quickly addressing the current lack of parallel computing pedagogy. A MOOC on parallel computing allows professionals to retroactively gain an education in parallel computing, while offering a framework and a template on which on-campus parallel computing courses can be based. Despite this observation, there has been little existing work in teaching parallel programming to a general, online audience and no prior publications describing techniques in implementing a parallel computing MOOC. Most existing offerings are highly specialized in their scope.

The first known instance of a parallel computing MOOC was offered in 2012 on Coursera and titled "Heterogeneous Parallel Programming" [7], taught by Professor Wen-Mei Hwu of the University of Illinois Urbana-Champaign. This course focused on teaching the essential parallel programming concepts for natively programming multi-core CPUs and GPUs using OpenCL or CUDA. As such, it was targeted at a specialized and low-level audience looking to improve their practical skills on multi-core hardware platforms. Unfortunately, this course is no longer offered.

In the Spring of 2013, Udacity introduced a similar course specifically on CUDA programming [8]. Like Professor Hwu's course on Coursera [7], this course focused specifically on teaching how to program with CUDA without much discussion of the fundamentals of parallelism. The lessons learned would for the most part not be reusable across parallel programming models or hardware platforms. Again, this course is low-level and focused on the skills needed to become a CUDA programmer, rather than a parallel programmer. This course is still available today.

In the Spring of 2015, Coursera and the University of Illinois Urbana-Champaign created a specialization on "Clouds, Distributed Systems, and Networking" [9]. This specialization differentiates itself from the previous courses in that it focused on teaching both the abstract concepts and practical aspects of cloud computing. While its focus on cloud computing is arguably less special-purpose than the emphasis on CUDA/OpenCL that the previous online offerings exhibited, it still focuses on a relatively niche topic rather than on parallel programming as a whole.

Most recently, in May of 2016, Coursera and the École Polytechnique Fédérale de Lausanne launched a course

on "Parallel programming" [10] using Scala. This course focuses on shared-memory task parallelism, data parallelism, and concurrent data structures in Scala. Functional programming is used to simplify writing correct parallel programs, but student solutions are not graded on the implementation performance. However, this is the first example of a parallel programming course taught from the fundamentals up and using a portable, general-purpose platform.

From the earliest parallel computing MOOC in 2012 to the most recent in 2016, there is a clear demand for both low-level and specialized courses and more fundamental and generalized courses. There is a growing demand for a comprehensive online education in parallel computing, but it is still important to students that the lessons learned be relevant to real world software engineering. Each of these existing courses has its own limitations regarding of how general-purpose are the models used to teach the courses (e.g. CUDA, Scala), how special-purpose are the hardware platforms that the courses focus on (e.g. cloud, GPUs), and to which the course teaches both useful fundamentals and specialized practical skills. Additionally, to the best of our knowledge, there are no publications describing the creation process for these courses.

In this paper, we present the design and implementation of a parallel computing curriculum, implemented as a Coursera Specialization. We emphasize broad applicability by teaching fundamentals of parallelism using common tools (i.e., the Java programming language and standard libraries), while motivating course lessons with problems relevant in real world software engineering. As such, we aim to provide professional software engineers with the tools that will make them immediately comfortable in a multi-core environment while teaching fundamentals that will be useful regardless of which programming model or language they use. This specialization is titled "Parallel, Concurrent, and Distributed Programming in Java", and shortened to PCDP for the remainder of this paper.

This paper is structured as follows. Section II summarizes the authors' past experience teaching on-campus parallel programming courses, and how that background contributed to the design of PCDP. Section III describes in detail the curriculum and structure of the PCDP specialization. Section IV describes the technical infrastructure that had to be constructed to support PCDP at scale. Finally, Section V concludes the paper.

II. Background

In previous publications [1][11], we have described our approach to integrating parallel programming into an on-campus undergraduate curriculum at Rice University. In particular, the on-campus offering, called COMP 322 [12], includes:

1) An initial focus on building a foundation of fundamental and general concepts in parallelism (e.g. parallel algorithms, data races, deadlocks, livelocks, critical path length, etc.)
2) Reinforcing those fundamental concepts with practical experience using both standard parallel Java programming constructs and a custom parallel programming library [13] to express parallel programs. This includes reasoning about the abstract and real performance of these parallel programs.
3) Using a partially flipped classroom by assigning online content to review prior to class, and spending half of in-class time in a class lecture format, with the other half spent solving in-class problems.
4) Using an auto-grading system to improve the automation and transparency of student evaluation.

COMP 322 has been offered nine times since Fall of 2009, and experiences gained teaching it contributed directly to the structure and design of the PCDP specialization.

In particular, COMP 322 is split into three modules: Parallelism, Concurrency, and Distribution. Parallelism covers the creation and coordination of parallelism, as well as common parallel algorithms and techniques for reasoning about parallel performance. Concurrency focuses on critical sections and concurrent data structures, i.e. how to share data among many parallel computations. Finally, distribution illustrates programming parallel machines when distribution of data or locality matters, e.g. on heterogeneous or distributed systems. Later, we will describe how the PCDP specialization's structure mirrors these modules.

COMP 322 uses Java to teach parallel programming for a number of reasons. First, the majority of students are likely to use Java in their future careers. Teaching practicum in Java improves the relevance of those lessons. Second, the majority of students will have already seen Java in past courses and be familiar with its syntax and conventions. Third, Java offers portability and reduces the challenges faced when supporting portable execution of a single code base across many different student laptops. Fourth, the Java Virtual Machine is designed to be a highly concurrent system (e.g. it has the `MONITORENTER` and `MONITOREXIT` bytecode instructions), making it a natural fit for parallel programming pedagogy.

COMP 322 primarily uses online quizzes and multi-week programming assignments to evaluate student comprehension. We have found quizzes to be a lightweight method for testing knowledge on very specific subjects, and identifying poor comprehension early on. Programming assignments are naturally crucial to giving students a chance to apply their theoretical knowledge in the real world. In general, programming assignments are graded on both the correctness and performance. Early assignments focus on abstract performance metrics (e.g total work, critical path length, ideal parallelism etc.) while later assignments focus on real world performance. Additionally, students are given midterm and final exams.

These characteristics of COMP 322 have developed over nearly a decade of parallel programming pedagogy. Naturally, these lessons helped to shape the PCDP specialization, as described in the next section.

III. Curriculum Development

The PCDP Coursera specialization consists of three separate courses: Parallelism, Concurrency, and Distribution. These courses mirror the structure of the on-campus course, COMP 322.

Each course consists of four weeks of content focused on different topics. Each week includes five lectures done in front of a light board, a demonstration video illustrating a practical use of the concepts from lectures, a quiz on the concepts taught, and a mini-project that gives students hands-on experience with the concepts from that week.

There are two goals in the design of the PCDP specialization which were often at odds with each other. On one hand, this content must be consumable by a programmer who is completely new to parallel programming, and so explanations should be simple and clear. At the same time, the specialization must leave that programmer with enough background and practice that their skills are immediately relevant. Striking a balance between providing enough detail to be useful, but not getting too deep into details as to lose students, was a crucial constraint on the content and structure of this specialization.

A. Parallelism

The Parallelism course is split into weeks on task parallelism, functional parallelism, loop parallelism, and data flow programming. As such, it provides a comprehensive overview of the most common parallel programming patterns in use today.

The Parallelism course starts by describing how to create parallelism using tasks, and how to abstractly reason about the perfomance of parallel programs using concepts like total work, critical path length, and Amdahl's Law.

With that foundation, this course then introduces the connection between functional and parallel programming using Java Streams. At the same time, the concurrency errors that are commmonly avoided when using functional programming (i.e. data races) are covered.

The course then introduces loop parallelism as a common parallel pattern similar to some functional parallelism patterns, and concludes with using constructs like phasers [14] to implement dataflow and pipeline parallelism.

Mini-projects in the parallelism course focus on illustrations of the real-world performance improvement possible with multi-threaded execution. While abstract performance metrics are taught, they are not used in the evaluation of student submissions. Instead, student submissions are evaluated on the real world speedup they are able to achieve. In particular, mini-projects used include reciprocal array sum, dense matrix-matrix multiplication, and one-dimensional iterative averaging (a 1D stencil code) to illustrate concepts taught in lectures.

B. Concurrency

The Concurrency course is split into weeks on threads/locks, critical sections, actors, and concurrent data structures. This course focuses on providing programmers with the tools necessary to efficiently manage concurrent accesses to shared data.

The Concurrency course starts by introducing two fundamental constructs: threads and locks. It discusses how they can be used safely together, but also explains how concurrency errors (e.g. deadlocks) are easily created with unstructured locking. The course then builds on that by discussing critical sections, and explaining the safety benefits (e.g. deadlock freedom) that result from using more structured forms of isolation.

Actors are then introduced as a different way of managing concurrent access to shared data without the need for explicit isolation on data structures. Finally, the Concurrency course concludes with an overview of low-level concurrent data structures (e.g. concurrent hash map, concurrent queue) as well as introducing linearizability as a property of these thread-safe data structures.

Mini-projects in the concurrency course include:

1) A performance comparison of Java locks, `synchronized`, and Java read-write locks when implementing a concurrent, read-heavy, sorted list.
2) Global isolation vs. object-based isolation for highly parallel applications, illustrated using a banking simulator.
3) Finding prime numbers using the Sieve of Eratosthenes algorithm, implemented using actors.
4) Implementing a parallel Boruvka's algorithm for finding the minimum spanning tree of a graph.

C. Distribution

The Distribution course is split into weeks on MapReduce, client-server programming, message passing programming, and effective combinations of distributed and multi-threaded parallelism. This course focuses on ways in which programmers can program distributed systems, including modern frameworks such as Hadoop MapReduce and Apache Spark.

Because writing applications for distributed systems is naturally a more specialized skill than general parallel programming, the distribution course also offers the most specialized topics of the three courses. Discussion starts with writing distributed programs using Hadoop MapReduce and Apache Spark, as these are argubly the most widely used distributed programming models for analyzing large datasets in industry.

Client-server programming is introduced as another important distributed programming pattern that more relevant

to the web technology industry. After introducing client-server programming, the course discusses how to combine multiple parallel processes (e.g. multiple web server instances) with multi-threaded parallelism within each process.

Finally, message passing programming models are introduced as the variant of distributed programming that is most common in distributed, high-performance computing. The message passing portion of this course starts from the abstract concept of Single-Program Multiple-Data parallelism and progresses to specific MPI APIs. Analogies are also drawn to MapReduce programming in that some computational patterns are expressible in both (i.e. a Hadoop Reducer vs. `MPI_Reduce`).

Mini-projects in the Distribution course generally have a higher level of complexity than in previous courses, and include implementing the Page Rank algorithm using Apache Spark, writing a simple file server, extending that file server to be multi-threaded, and writing a parallel matrix-matrix multiplication using MPI.

D. Curriculum Review

All content developed for the PCDP specialization went through several rounds of review. First, all videos, quizzes, and mini-projects were reviewed by at least two undergraduate students that had taken Rice's on-campus parallelism course. These reviewers were helpful in checking for grammatical errors, clarity, and correctness.

Then, courses are sent to Coursera for a "beta" test where external Coursera learners take the course and offer feedback to the course staff. This feedback is addressed before the course goes live.

IV. Technical Infrastructure

In addition to challenges developing a curriculum appropriate for a MOOC (described in Section III), there are also unique technical challenges in supporting a large-scale parallel computing course, particularly one that evaluates students on the real-world performance of the solutions to their assignments. We highlight three of those challenges in this section: mini-project development, a custom parallel programming library, and automatic grading of student submissions.

A. Mini-Project Development

In the PCDP specialization, all mini-projects are provided as Maven [15] projects. This simplifies their deployment, compilation, dependency management, and IDE integration on student laptops.

Mini-project topics are generally chosen to be understandable, relatable, and topical to the parallelism concepts students are being taught. These mini-projects are not intended to be multi-day efforts, but rather brief hands-on experiences with the concepts students are learning. For example, the MPI mini-project parallelizes a dense matrix-matrix multiply across MPI ranks. While the topic matter is straightforward and understandable, this project still requires understanding of topics such as data distribution, send/receive APIs, barriers, and asynchronous communication.

Mini-projects are described in a one to two page project description page in the Coursera course. The description of each project includes at least the following four sections:

1) Project Goals & Outcomes: A high-level overview of the concepts students will use to implement the mini-project, the expected outcome, as well as pointers to related material in the lectures or demo videos.
2) Project Setup: Instructions on downloading and setting up the provided mini-project source code.
3) Project Instructions: More detailed instructions on completing the mini-project, often paired with helpful `TODO`s in the provided mini-project source code.
4) Project Evaluation: Instructions on how to test student code locally and in the Coursera autograder (details in Section IV-C), as well as a rubric that their submission will be evaluated on.

Mini-project development especially emphasises thorough and useful documentation of provided source code to aid student understanding and to focus their efforts on expressing parallelism, not on reading boiler plate code. Significant time was also spent ensuring that the provided tests were consistent and reliable across different platforms.

Optimally, mini-projects should be executable across all student platforms. In general, the choice of the Java Virtual Machine as an execution environment guarantees this. In only one case (the mini-project on MPI) are there third-party, native dependencies which may prevent some students from running the mini-project locally. However, students always have the ability to test their code on a shared auto-grading infrastructure (described in Section IV-C). While this may be a cumbersome development experience, we do not foresee it being a regular practice.

To help address student environment or platform issues early on, a "Mini-Project 0" is offered at the start of the course that asks them to compile and run a simple parallel program on their laptop. This enables early identification of any infrastructure issues for students.

B. Parallel Programming Library

While much of the PCDP specialization focuses on traditional parallel programming frameworks, the APIs of those frameworks can be verbose and cumbersome for newcomers to parallel programming. As a result, a lightweight parallel programming library for Java was developed and open sourced [16] as part of this work, based on past experience developing the pedagogical HJlib parallel programming library [13][1]. For the remainder of this paper, we refer to this lightweight library as PCDPlib.

PCDPlib is a Java parallel programming library that exposes lambda-based APIs. Using a library rather than a

```
// Waits for all nested tasks to complete
PCDP.finish(() -> {
  S1;

  // Spawns an asynchronous task
  async(() -> {
    S2;

    // Spawns a nested asynchronous task
    async(() -> {
      S3;
    });
  });

  // Spawn a sibling asynchronous task to S2 above
  async(() -> {
    S4;
  });
});
```

Figure 1. A simple example of the PCDP task-parallel APIs.

custom language has significant long-term maintainability and stability benefits. User logic is expressed in the body of Java lambdas, and the PCDPlib APIs are used to pass this logic to the runtime for scheduling and synchronization.

For example, Figure 1 offers a simple illustration of using PCDP's APIs to create asynchronous tasks (`async`) and perform bulk task synchronization (`finish`).

PCDPlib's implementation was deliberately kept simple while still supporting the parallel programming constructs needed for the PCDP specialization. In particular, PCDPlib supports the combination of the following parallel programming constructs in a single program and on a single runtime:

1) Asynchronous task parallelism
2) Bulk task synchronization
3) Multi-dimensional parallel loops (chunked and unchunked)
4) Futures
5) Isolated critical sections
6) Actor parallelism

A listing of all PCDPlib APIs can be found at the PCDPlib Javadocs [17].

A strong emphasis was placed on keeping PCDPlib's implementation simple for its pedagogic value. In particular, we believe that it is important for a student who is curious about the implementation of a parallel API to be able to read and understand the source code that implements that API. This was also part of the motivation for open sourcing the library. By keeping the implementation of PCDPlib approachable and open, we hope that students may use it as a supplementary material for learning about parallel systems.

Of course, not every mini-project in the PCDP specialization depends on PCDPlib. In general, as the specialization progresses to more advanced topics, fewer mini-projects make use of PCDPlib and instead rely on more traditional frameworks. For example, no mini-projects in the Distribution course of the specialization depend on PCDPlib.

C. Automatic Grading

Given that the "M" in MOOC stands for "Massive", manual grading of student submissions in the PCDP specialization is not feasible. In previous work [1], the authors have explored auto-grading techniques for Rice University's on-campus parallelism course. However, Coursera provides autograding infrastructure for courses hosted on its platform, including what are called "custom graders" [18]. A "custom grader" allows instructors to upload custom Docker [19] images which are passed a student submission as input, and which produce a grade for that submission as a JSON-formatted feedback object. This offers instructors wide flexibility in grading the assignments. Different Docker objects can be uploaded to Coursera for different assignments and used to add customized grading for each assignment. From a security point of view, the benefit of using custom Docker objects is that instructors can use root privileges to pre-install the necessary software on their images without having root privileges on the running instance while grading a student submission.

The Coursera custom grading system allows instructors to request variable amounts of CPU cores and memory for each grading instance. For the PCDP specialization, we always request the maximums for both: 4 CPU cores and 4GB of memory.

For this course, we developed a generic, JUnit-based testing infrastructure on top of the Coursera custom grader system. For each assignment, the instructor must provide a rubric which specifies the points for each JUnit test. Our custom testing infrastructure is then responsible for creating a complete project around the student submission, compiling it, running all JUnit tests at varying core counts, and generating a grade based on the provided rubric and observed correctness/performance of each test. This infrastructure is shared across all mini-projects in the specialization; only the rubric file needs to be changed from one mini-project to another. This greatly simplifies deployment of custom graders, and improves the robustness of the testing infrastructure.

One of the largest challenges in developing this shared testing infrastructure was supporting the full variety of mini-projects. For example, the custom testing infrastructure had to be flexible enough to support execution of multi-threaded Java, multi-threaded PCDPlib, MPI, Apache Spark, and multi-process/multi-threaded web server programs. This had to be accomplished from within Coursera's containerized environment.

V. Conclusions

As a result of recent hardware trends, parallel computing has been identified as an area of computing education in

need of significant improvements [2]. While existing efforts to improve parallel computing education have focused on undergraduate curriculum, those efforts are generally unscalable to the majority of programmers and leave many students unsupported (e.g. undergraduate students at understaffed institutions, professionals in industry).

This paper summarizes the existing work in this area in Section I, and differentiates our approach, the Coursera PCDP Specialization in both its generality and comprehensiveness. We describe the development of both the course curriculum and technical infrastructure for this course in Sections III and IV. The PCDP Specialiation is built to be scalable and accessible, offering both 1) a strong grounding in the general fundamentals of parallel computing, and 2) practical experience programming parallel systems.

ACKNOWLEDGMENT

We would like to thank Coursera, the Rice Center for Digital Learning and Scholarship, the teaching staff for COMP 322, and Rice University for their support in developing the PCDP Specialization.

REFERENCES

[1] Grossman, Max and Aziz, Maha and Chi, Heng and Tibrewal, Anant and Imam, Shams and Sarkar, Vivek, "Pedagogy and Tools for Teaching Parallel Computing at the Sophomore Undergraduate Level," *Journal of Parallel and Distributed Computing*, 2017.

[2] S. K. Prasad, A. Gupta, A. L. Rosenberg, A. Sussman, and C. C. Weems, *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*. Morgan Kaufmann, 2015.

[3] J. Markoff, "The most popular online course teaches you to learn," https://bits.blogs.nytimes.com/2015/12/29/the-most-popular-online-course-teaches-you-to-learn/?_r=0, 2015.

[4] Coursera, "Coursera," https://www.coursera.org/.

[5] edX, "edX," https://www.edx.org/.

[6] Udacity, "Udacity," https://www.udacity.com/.

[7] Wen-mei W. Hwu, "Heterogeneous Parallel Programming," http://academictorrents.com/details/8903d0871c652b96c7b29db738cea76902d65888.

[8] Luebke, David and Owens, John and Roberts, Mike and Lee, Cheng-Han, "Intro to Parallel Programming," https://www.udacity.com/course/intro-to-parallel-programming–cs344.

[9] Farivar, Reza and Singla, Ankit and Gupta, Indranil and Godfrey, P. Brighten and Campbell, Roy H., "Clouds, Distributed Systems, and Networking," https://www.coursera.org/specializations/cloud-computing.

[10] Kuncak, Viktor and Prokopec, Aleksandar, "Parallel Programming," https://www.coursera.org/learn/parprog1.

[11] Aziz, Maha and Chi, Heng and Tibrewal, Anant and Grossman, Max and Sarkar, Vivek, "Auto-Grading for Parallel Programs," in *Proceedings of the Workshop on Education for High-Performance Computing*. ACM, 2015, p. 3.

[12] "COMP 322: Fundamentals of Parallel Programming," 2014. [Online]. Available: https://wiki.rice.edu/confluence/display/PARPROG/COMP322

[13] S. Imam and V. Sarkar, "Habanero-java library: a java 8 framework for multicore programming," in *Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools*. ACM, 2014, pp. 75–86.

[14] J. Shirako, D. M. Peixotto, V. Sarkar, and W. N. Scherer, "Phasers: a Unified Deadlock-Free Construct for Collective and Point-to-Point Synchronization," in *Proceedings of the 22nd annual international conference on Supercomputing*. ACM, 2008, pp. 277–288.

[15] The Apache Software Foundation, "Apache Maven," https://maven.apache.org/.

[16] Imam, Shams and Grossman, Max, "PCDPlib Source Code," https://github.com/habanero-rice/PCDP.

[17] ——, "PCDPlib Javadocs," https://habanero-rice.github.io/PCDP/.

[18] Coursera, "Coursera Custom Graders," https://github.com/coursera/programming-assignments-demo/tree/master/custom-graders.

[19] Docker, "Docker," https://www.docker.com/.