Trying to do it all in a single course: a surprisingly good idea

Tia Newhall
Computer Science Department, Swarthmore College
Swarthmore, PA USA

***Abstract*—We present the curricular design and learning goals of an upper-level undergraduate course that covers a wide breadth of topics in parallel and distributed computing (PDC), while also providing students with depth of experience and development of problem solving, programming, and analysis skills. We discuss lessons learned from our experiences teaching this course over the past 10 years, and discuss changes and improvements we have made in its offerings, as well as choices and trade-offs we made to achieve a balance, in a single course, between breadth and depth of topic across these two huge fields. Evaluations from students support that our approach works well meeting the goals of exposing students to a broad range of PDC topics, building important PDC thinking and programming skills, and meeting other pedagogical goals of an advance upper-level undergraduate CS course. Although our single course design was created due to constraints common to smaller schools that have fewer faculty resources, smaller curricula, and often fewer required courses for their majors, our experiences with this course lead us to conclude that it is a good approach for an advanced undergraduate course on PDC at any institution.**

***Keywords*-parallel and distributed computing; CS Education; curriculum design.**

I. Introduction

The ubiquity of multi-core processors, accelerators, cloud computing, clusters, and other parallel and distributed platforms, and the explosion of applications of parallel and distributed computing solutions to an increasingly diverse range of disciplines, has resulted in PDC knowledge and skills being a core part of undergraduate computing curricula. Recent ACM and IEEE curricular guidelines highlight the importance of PDC topics: the ACM-IEEE CS2013 Curriculum [1] added a new knowledge area in Parallel and Distributed computing; and the NSF/IEEE-TCPP Curriculum Guidelines [2] further specify expanded details on PDC curricula, and provide guidance and support for incorporating these topics into the undergraduate computing curriculum.

These efforts highlight the importance of incorporating PDC topics throughout the curriculum, as well as expanding upper-level undergraduate course offerings whose primary focus is on parallel and distributed computing.

Smaller institutions, such as liberal arts colleges, have more limited faculty resources, fewer curricular offerings, and often a shallower prerequisite structure to their CS major than departments at larger institutions. Large universities often have several course offerings on parallel and distributed systems, architecture, languages, or algorithms as part of their curricula. At smaller institutions, there is often a single upper-level course devoted to a parallel or distributed computing topic, limiting students' exposure to this increasingly important field. As a result, it is important to expose undergraduate students to a large breadth of PDC topics in what may be their only course option devoted to advanced study in PDC. Our course was developed in this context, under these curricular and teaching constraints common to smaller departments.

In this paper we present the design of a single course on parallel and distributed computing for undergraduates [3]. It has an overall goal of exposing students to a large breadth of parallel and distributed computing topics. Our course is designed as an advanced upper-level undergraduate course, with curricular and pedagogical goals of preparing students for graduate studies in CS, and specific PDC curricular goals of teaching students programming and problem solving skills that will help prepare them for PDC related careers. The ideal student in our course is a senior major who has completed most of the CS major requirements. In practice, we often have juniors and a handful of sophomore pre-majors in the course, which means that we need to provide some extra scaffolding to support a broad range of student backgrounds in CS.

We structure the course as a mixed lecture-style and seminar-style course. The seminar portion includes weekly in-class discussions of PDC research papers, a large independent course project, and multiple writing and oral presentations. The lecture portion includes traditional lectures on PDC systems, languages, algorithms and analyses, in-class problem solving activities, lab assignments on a wide range of parallel and distributed programming languages and tools, and practice designing and running large-scale experiments.

Prior to this course, we offered a traditional distributed systems class run purely in seminar-style. As PDC has become more pervasive, we were less satisfied with a course that focused only on distributed systems, and we wanted to expand coverage of parallel computing topics, including languages, tools, architectures, systems, algorithms, applications, and broader analyses. Our desire to include more of this content was a significant driver of the development of this course.

On the surface, a single course that tries to cover the breadth of PDC is a bad idea. These two huge field span

most of CS, and are increasingly growing, pervasive, and complex. Trying to "do it all" is impossible and even trying to get close is likely to result in an unsatisfactory learning experience.

In designing a course like ours, it is important to keep in mind that you can't do it all, and instead seek to provide a balance between a large degree of breadth of topic coverage and problem solving, while also including many opportunities for in-depth coverage and developing deeper problem solving, programming, analysis, and research skills. Our course is designed to attempt this balance.

To date, we have taught our course 5 times over the past 10 years. We have learned many lessons along the way for effectively designing and implementing a single course curriculum that covers a huge range of topics and learning goals. And, although our course was initially designed due to constraints of CS at a small liberal arts college, we like the breadth of coverage in this single course, and find its design to be a good undergraduate-level course introducing these fields. As a result, even in a larger and more resource-rich department that has a large number of course offerings in PDC topics, we think this is a good design for an undergraduate level course in PDC. It provides a good balance between breadth of exposure to topic and ideas with depth of understanding, and we recommend our course for any undergraduate curriculum, regardless of the constraints of a particular institution.

In Section II we present related work in PDC curriculum, tools and learning. In Section III we present details of our curriculum and its learning goals, discuss trade-offs, and present details of lab, writing, and active learning assignments we use. In Section IV we present evaluations by students who have taken the course. And in Sections V and VI we present our evaluation of the course, and discuss changes we have made and lessons we have learned.

II. Related Work

Motivated by the increasing importance of PDC knowledge as a part of the core undergraduate CS curricula, as well as by growing demand for PDC skills by the work force, there have been numerous contributions to help educators incorporate and expand coverage of PDC in their curricula. Some of these include model curricula and pedagogical guides [4]–[10], and numerous teaching tools, training, workshops, and other resources [11]–[13].

There has also been work related to active learning and collections of unplugged classroom activities for teaching PDC concepts [14]–[17]. Most of these are aimed at introductory sequence CS courses introducing parallel concepts and parallel thinking. Our use of in-class activities is aimed at upper-level undergraduate courses.

To our knowledge, ours is the only course that balances covering a large breadth of PDC topics with some in-depth study and investigation, that mixes a seminar-style paper reading and discussion and project-based experience with more traditional lecture and active learning experiences, that includes many written and oral presentation components, that fits at the level of an advanced upper-level undergraduate course, and that is particularly designed for institutions with room for only one upper-level course devoted exclusively to PDC.

III. Course Details

The topics we cover span the two huge fields of parallel and distributed computing, each of which touch most parts of CS, including architecture, systems, algorithms, applications, languages, and tools. Any sub-area alone could fill an entire course. Thus, our single course curriculum must be broad enough to touch on much of this vast range of topics. A single course cannot cover everything, nor cover all topics in the same amount of depth that a course with a narrower focus could, such as a course on parallel algorithms. However, to be a successful learning experience it must provide multiple opportunities for both depth of coverage and depth in development of problem solving and PDC thinking skills.

Our approach is to focus on fundamental topics across a broad range of areas in this field, through research paper reading and discussion, lecture, and lab activities. We select a specific theme to unify topics and to develop deeper analytic skills. And we also provide an opportunity for deep investigation into a specific topic through an independent course project.

Each week, there is a seminar-style paper discussion meeting and a lecture-style class meeting. Additionally, our course has a scheduled lab meeting each week that we use to introduce resources and tools necessary for assigned lab work and course projects.

The primary learning goals of the course are to:

- understand the fundamental questions in parallel and distributed computing and analyze different solutions to these questions.
- understand different parallel and distributed programming paradigms and algorithms, and gain practice in implementing and testing solutions using these.
- analyze and critically discuss research papers, both in writing and orally.
- formulate and evaluate a hypothesis by proposing, implementing and testing a project, and relating one's project to prior research via a review of related literature.
- write a coherent, complete paper describing and evaluating a project.
- orally present a clear and accessible summary of a research work.

Additionally, our course satisfies the college's requirements of a writing intensive course.

A. Student Preparation

Our institution, like most liberal arts colleges, has a shallow pre-requisite structure. We start with a traditional CS1 course, followed by two intermediate-level courses (data structures and algorithms, and introduction to computer systems), which serve as the only prerequisites to all upper-level courses. Upper-level courses are divided into three groups: theory and algorithms; systems; and applications. Majors are required to take 5 upper-level courses, at least one from each group.

Our intermediate-level introduction to systems course was first added in 2012. It introduces computer organization, operating systems, and parallel computing, focusing on shared memory parallelism and threads. Students in this course learn C, assembly, and pthreads programming. This class is the keystone in a redesign of our curriculum [7] that introduces parallelism early, allowing room to add and expand parallel and distributed topics in many of our upper-level courses that are not focused on PDC.

Because all students now enter our PDC course with experience in shared memory parallelism and pthreads programming, we are able to immediately start with reviewing what they know about parallel computing rather than introducing parallelism. This allows us to expand breadth and depth of PDC coverage in our course.

B. Paper Reading and Discussion

Each week students are assigned one or two papers to read for class discussion. We split the larger classes into two sections to foster a smaller seminar experience and to ensure all students have appropriate opportunity to participate. We provide students with instruction on how to read CS research papers, and we use reading groups and reaction notes assignments with each reading.

1) Reading Groups and Reaction Notes: To help prepare students for in-class discussion we use paper reading groups and assign reaction notes that are due before the in-class discussion of the weekly paper reading assignment. Every student is assigned to a reading group that is required to meet before the in-class discussion and after each group member has read the paper(s). Reading groups discuss the assigned papers and draft their group's reaction notes.

There are three parts to reaction notes: a high-level summary of the main idea of the paper; an answer to a focused question about the paper; and a list of 2-4 questions from the group's discussion of the paper, ones they feel are most interesting or unresolved. Group members take turns finalizing and submitting the write-up of the group's reaction notes, which are due before class. Instructors usually have time to skim through reaction notes prior to in-class discussion; this helps us to call on students who are reluctant participators to answer a question we know their group discussed, or to see if there are some common misconceptions, interesting analyses, or questions that we should include.

The reading group meetings ensure that students are better prepared for in-class discussion and result in better participation and deeper discussions in class. They also seem to put students more at ease participating in class discussions.

2) Selecting Papers and Paper Topics: The focus of paper topics differs in the first and second halves of the course. In the first half, papers focus on languages, tools, and libraries for parallel and distributed computing. This gives students background on tools they will use in labs and course projects. In the second half, papers focus more broadly on PDC topics such as cloud computing, grid computing, distributed file systems, peer-to-peer systems, security, green computing, consensus, and fault tolerance. We choose more systems topics because this course satisfies a Systems requirement for our major.

In addition to the split of paper topics, earlier papers must be more accessible than papers assigned later in the semester, after students know more of the field and have more practice reading, analyzing, and discussing papers. In general, we find that foundational papers on a topic, and summary papers of a field, tend to be more accessible. The former generally do not assume the reader has a large background in the field, and the latter tend to be written for a more general audience.

Because students do not read a paper and prepare for in-class discussion before the first week of class, we often use the first week discussion meeting for a group building Guild exercise [18]. We find this helps students see the different strengths individuals bring to the group, and this often helps alleviate group and partnership difficulties.

Additionally, we assign as the first "paper" an oral presentation by reading groups on a particular parallel or distributed system. Each group is assigned a different type of system (e.g. MPP, cluster, gpu, cloud, SMP), and we suggest particular examples of each type on which they can focus their presentation, including many taken from the Top500 and Green500 lists [19]. Groups prepare an oral presentation of their system and deliver it at the weekly discussion session in the second week of class. This first "paper" assignment has several goals including: effectively working with their reading group; investigating a system; and delivering an oral presentation early in the semester. Students enjoy this exercise learning details about a particular system. It also helps later in the semester to expose students to a range of systems early.

As an example, listed below are the set of weekly paper assignments from the most recent offering of the course in the spring of 2020 (note that the last two paper assignments were cut due to changes in the semester resulting from the COVID-19 pandemic). Additionally, the course webpage [3] includes links to webpages from previous offerings of the course, each of which includes the list of assigned papers from that semester. In weeks with two assigned papers, often one is the main focus paper and the other is a "quick read"

paper that provides additional background or breadth on a topic.

- Oral presentation of a parallel or distributed system.
- "End-to-end arguments in system design" [20], and "The design philosophy of the DARPA Internet Protocols" [21].
- "MPI: a message passing standard for MPP and workstations" [22] and "OpenMP: An Industry Standard API for Shared Memory Programming" [23].
- "Scalable Parallel Programming with CUDA" [24], and "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems" [25].
- "TreadMarks: Shared Memory Computing on Networks of Workstations" [26].
- "MapReduce: Simplified Data Processing on Large Clusters" [27].
- "Heterogeneous Computing: Here to Stay: Hardware and Software Perspectives" [28], and one of our research papers as an introduction to project ideas related to our work.
- "The Google File System" [29].
- "A Guided Tour through Data-Center Networking" [30], and "A View of Cloud Computing" [31].
- "The Byzantine Generals Problem" [32], and "Simple Testing Can Prevent Most Critical Failures" [33]
- "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications" [34], and "What Is This Peer-to-Peer About?" [35].
- "Computer Security in the Real World" [36] and "The Internet Worm" [37].

C. Lecture

We use the lecture portion of the course to provide background in parallel and distributed computing topics, and to provide context for weekly paper readings. We include details of specific algorithms, languages, and systems, and introduce metrics for comparison and analyses. We use scalability as the pervasive theme throughout the semester. This theme is a good way to link the disparate set of topics and techniques we discuss, and to provide a common metric for analysis across topics.

Over time, we have expanded our use of group problem solving activities in place of more traditional lecture. These exercises benefit learning and help develop skills in group problem solving and presentation. We designed a set of exercises that span a range of PDC topics and focus on analysis and comparison. The activities use a common structure and include designating specific roles to individuals, namely:

- Group Spokesperson: presents the groups answers/ideas/solutions to the class.
- Group Moderator: runs the group discussion, ensuring that everyone gets a chance to share their ideas and understands the group's solution.
- Group Note Taker: takes notes on the group's ideas to help with discussion and presentation.
- Group Timer: ensures the group keeps moving appropriately and stays on task.

All group members participate in group discussion and problem solving. and we ensure that individual students take turns in different roles.

We developed the following problems for these in-class group activities (in a given semester we use most of these):

- Questions on a paper that they read before class. This is given very early in the semester, before the first paper reading and in-class discussion. The goal is to provide a structured practice of some of the types of questions we want them to discuss in their paper reading groups, using an example paper that is very accessible (we use [38]).
- Questions comparing two different types of parallel systems (usually a shared memory and a distributed memory) on several metrics related to parallel computation and overheads, scalability, and types of parallelism each support well. This exercise serves as a good introduction to a number of topics and themes that we revisit throughout the course.
- Developing a parallel algorithm for computing an operation or function (e.g. sum). This includes big O analysis and evaluating parallel overheads and scalability expectations and limits of their algorithm. Students often come up with unexpected techniques, resulting in lively discussions and often providing a lead-in to techniques we present in later lectures.
- Developing an algorithm to determine an ordering of events in a distributed system. This exercise replaces a lecture on time and event ordering in distributed systems. We have been surprised that students come up with ideas similar to Lamport's Happens-Before [39] relation for partial ordering, and centralized (and sometimes decentralized) solutions for agreeing to a total ordering.
- Shorter assignments analyzing different algorithms presented in lecture. They compute both big O complexity in terms of N and P, but also consider parallel and distributed overheads, including those due to remote memory access, and synchronization and communication. We typically do several of these each semester.

Over the years we have replaced more pure lecture with these group problem solving activities, resulting in noticeable improvements in students problem solving, engagement, and oral presentation skills, and with their comfort level participating in class. We still use more traditional lecture, and feel it is important and has its place, but we intersperse lecture with these group activities to achieve a good balance of topic exposure and to build important skills through practice and application.

D. Lab Assignments

The scheduled lab meeting each week has two purposes: in the first half of the semester, we assign several programming assignments to teach students a wide set of PDC languages and tools; in the second half of the semester, we help with, and monitor progress of, course projects, and introduce tools for experimentation and for writing reports.

The first half labs include:

1) A scalability study of a reworked version of their pthreads GOL program from the introduction to systems course. The goal is to gain practice designing hypotheses and experiments to evaluate hypotheses, and to analyze test results and convey them in written form. It is also a reminder of parallelism, pthreads, and C programming.
2) Implementing a distributed client/server application. The goal is to learn TCP/IP socket programming. We have used two versions of this assignment: a multi-threaded web server and client, and a multi-threaded chat server and client.
3) Implementing a Cuda [40] application using the ParaVis [41] library to visualize its parallel computation. The goal is to learn GPGPU programming, making use of visualization to help with debugging and with finding errors in their parallelization. The particular assignment is a 2D fire simulator, similar to GOL, chosen so students focus more on the CUDA parallelization than on the 2D discrete event simulation.
4) Implementing an MPI [22] application (usually odd-even sort), and testing it on an XSEDE [42] cluster. The goal is to learn message passing programming in MPI, to implement an algorithm discussed in class, and to gain practice using XSEDE resources.
5) A shorter OpenMP [23] assignment parallelizing a sequential application. The goal is to learn how to use OpenMP to incrementally parallelize loops given a sequential application starting point.

We cannot cover every language and system in the first half lab assignments, but we design labs to span a range of parallel and distributed computing, choosing examples of commonly used languages and tools. This provides experience with languages and systems that they may use in their course project. With more time, we would add a lab on cloud computing using some implementation of MapReduce [27], and possibly another using a PGAS language [43].

E. Course Project

The independent project component of the course is introduced mid-semester. At this point students have been exposed to a breadth of topics, to programing in different languages and paradigms, to reading and analyzing primary research papers, and to using many tools and resources.

The project must have a main focus on parallel or distributed computing, must include some implementation and experimentation and evaluation, and should be structured around a general question that they are trying to answer. Some groups focus more on implementation, while other groups may have smaller implementation and focus much more on experimentation and analyses (often scalability studies).

We give students some ideas for project topics and invite students to look for projects that combine something they learned in other CS classes with parallel or distributed computing. Students enjoy the open-endedness of this and the opportunity to define their particular project in a broad way.

To help students stay on track for successfully completing their course project, we assign four deliverable parts, including:

1) **Project Proposal and Annotated Bibliography**. A structured investigation, reading, and writing exercise to help them define their course project and develop a plan to carry it out. The annotated bibliography requires investigative background reading of related work. The proposal includes details of what they are going to do and how they are going to do it. It allows us to give early feedback on their project plan, and tips for tools and resources or related projects that may be helpful.
2) **Midway Project Report and Oral Presentation to class.** A smaller writing requirement that follows Project Work Week, a full week for students to solely work on their course project. Midway requirements include meeting with the professor and updating their project schedule with changes to their plan and timeline. Following Project Work Week, each group presents their project to the class. It is the first time that groups learn details of other's projects, and is an opportunity to get feedback from the class. The presentation is also motivation for students to make significant progress on their project during Project Work Week.
3) **Final Oral Presentation.** Project groups prepare and present their project to the class and the larger CS community. These presentations are similar to CS conference talks in design and content,.
4) **Written Final Project Report and Demo.** Students submit a final project report that is structured like a CS conference or journal paper. We provide guidelines for the structure and focus, links to writing resources, and include a "revise and resubmit" of their report introduction. Students submit their project code and give a demo of their project. The demo is an opportunity for them to show off some of the implementation details, to talk through difficulties they had along the

way, and to discuss features they would add next given more time. It is a nice way to end the course.

Currently, we require that course projects are done in pairs or small groups, and we no longer allow individual course projects. This helps the instructor manage projects as our class enrollments have significantly increased. More importantly, we've found that group projects are stronger projects, are much better experiences for our students, and are more successful at achieving the learning goals we have for this experience.

IV. Student Evaluation

We present results of students' assessments of our course over its most recent two offerings that include all of the added learning exercises and support and changes we have made to the course over 10 years. Student evaluations from earlier offerings are similar to the ones we present here, but include assessment of parts that we have removed or changed, and miss assessment of parts we have added. The results are from course surveys asking students to evaluate parts of the course, what they learned, and their experiences and feedback about the class.

We asked students about the structure of the class. Overall, students like the balance of paper discussion and lecture (Figure 1a). Among students who would like a different distribution, there was more preference for increasing paper reading and discussion (33%), than for increasing lecture (14%). Almost all students mentioned that they like having both, even those who preferred a different balance. A common theme in responses was summarized by one student who said "both the lecture and discussion taught me a lot and complemented one another." Another said "I liked this split actually because we were engaging in each section in a different way and were learning content through a variety of means." A student who strongly preferred more paper reading and discussion explained their reasoning "because actively having to explain and discuss topics forces us to understand the material so much more", another describes class discussions as "invaluable in deepening my understanding. I loved most of the papers!" Students who wanted more lecture indicated that this was because they wanted to cover more material and background and to "delve into greater details about different topics of the course". Overall, paper reading and discussion is the part of the course students like best.

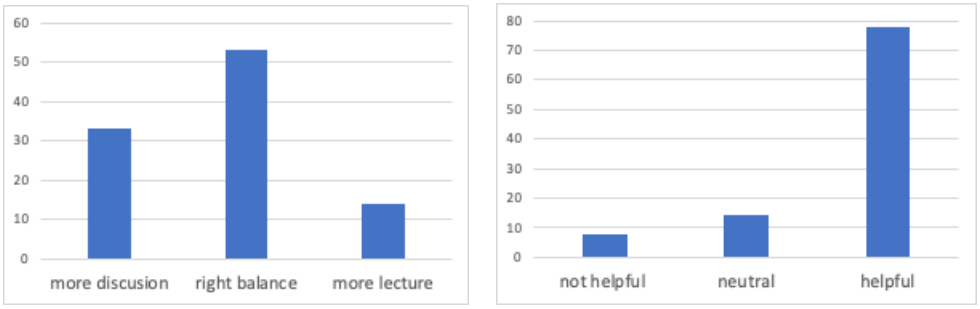
We asked students to evaluate how helpful in-lecture group problems were to their understanding of PDC. Overall, we received many "very helpful!" responses to this question (78% in Figure 1b). One student noted that "figuring out/discussing the problems in small groups allowed for more ideas/info than working alone." Another noted that they were "quite helpful to gain intuition on the various problems we explored." Students particularly liked problems that were more open-ended. A common theme in responses was summarized by one student who said "discussing ideas with peer[s] was a good way to bootstrap thinking/problem solving."

Roughly 14% had a neutral opinion about group problems, and 8% found them less helpful than lecture. Those who found them less helpful generally wanted more lecture in order to cover more topics in class. A couple students thought the problems were too easy, while a few more said some were too difficult. Overall, the most common complaint was that some problems were not completed in a single class meeting. We continue refining to avoid this problem, which is exacerbated by the fact that we only have one lecture meeting per week.

Responses to questions about how well reading groups and reaction notes prepared them for in-class discussion were unanimously positive. Students found the reading groups "very helpful" and valued the experience of discussing papers before class. Many comments were similar to a student who said that it "was nice being able to hear the kind of takeaways and questions my classmates got from the same reading", and others said "the groups ensured I spent more time really trying to understand the papers" and "I felt very prepared for in-class discussion."

Students similarly felt that reaction notes helped prepare them for paper discussions in class. A common theme in responses was summarized by one student: "I found the group reaction papers helpful in making sure each of us understood the paper, filling in gaps in our understandings, and gave us the opportunity to discuss/prepare for class." Another student summarized group discussion and reaction notes as "very helpful to help me synthesize information, discuss questions about the papers, and make sure that I kept up with the readings" another said that they resulted in "forcing me to read more carefully and feeling more confident to contribute in class." The only complaints about reading groups and reaction notes were from earlier offerings of the course where students felt reaction notes grading was unclear. We have since shared our grading rubrics to resolve this issue.

We also asked students about the difficulty of the assigned labs in the first half of the course, and if labs were useful in preparing for course projects. Overall, students found the labs to be very helpful and appropriately challenging, although a bit more challenging than other CS courses. A common theme was summarized by a student: "They were somewhat challenging, but at a reasonable level. They definitely helped prepare us for the course project as we learned how to use various tools for implementing parallel and distributed systems." Several students mentioned difficulties with relearning C and C++ programming after several semesters using Python in upper-level courses. Most felt that the lab assignment instructions and supporting materials were very helpful, and most felt that they were very good preparation for their course projects. Almost all students



(a) Balance of Lecture and Paper Discussion (b) Helpfulness of in-class exercises to understanding

Figure 1: Students responses to questions about course structure (results as percentages).

really enjoyed the labs, even those who found some very challenging, and stated that they learned a lot from them. One student summarized the common response well as: "Great labs, always a good degree of challenge, was great to have knowledge of different P&D techniques going into the final project."

Another question asked how well prepared students felt to do the final project. Students almost uniformally felt very well prepared for the project, naming the lab assignments and paper reading, discussions, and reaction notes as being particularly helpful. One student summarized student responses well: "I think the lead up from the first half of the semester to the project was really nice, especially because it seemed like the project was a culmination of what we've learned in the first half of the semester. The instructions for each part were very thorough, and it was really helpful guidance! So I feel we were very prepared to do each of the parts of the final project." Only a couple students felt not as well prepared, but all of these responses were about unexpected time commitments and constraints. In particular, they wished they had more time for their midway project deliverables. Several students who felt well prepared for the project also mentioned that the midway report was a bit too much, and we plan to cut it down some in the future as a result of this feedback. A few students thought there were too many deliverables along the way, but most appreciated the guidance and checkpoints. One student's comment summarized a common response: "[the professor] provided the necessary resources and guidance to complete each step. It was nice to have built in time to work and check ins throughout to make sure we kept on top of the work."

We also asked students three questions specific to course learning goals, paraphrased as how has this class affected your:

1) ability to analyze and critically read CS research papers?
2) ability to formulate a research question and implement an experiment to answer the question?
3) ability to write a clear and complete CS research paper?

Universally, students express large improvement in all of these abilities. The only exception was the second question, where some students stated that they already had a lot of experience with this, so their noted improvements were not as large.

Students reported the largest improvements in their ability to critically read and analyze papers. Many expressed sentiments like "I feel like I improved a ton in this regard" or "my ability to read CS research papers has improved dramatically!" All students reported significant gains in their abilities. For example, one student said: "I realized how lost I was reading these papers at first and how seamless it is by the time we were doing it for our final project." Another said "I feel much more equipped to read and intelligently discuss papers."

Students also felt the course helped with formulating research questions. Students commonly mentioned improvements in experimentation and hypothesis formulation, such as in "formulating questions to test functionality of a system/program", in "conducting experiments" and "especially when experiments failed our first few trials". Students with some prior experience with CS research all mentioned improvements in their abilities. One student summed up a common theme with: "I think the project definitely helped with my skills for formulating research questions! Although I've done summer research, the research questions that we had weren't really mine, so I think this time I had more of an experience in formulating a research question, which was really nice."

Finally, all students felt their ability to write a CS research paper greatly improved, many using terms like "tremendously" and "dramatically" in their responses. Most who had little to no research and writing experience prior to this course commented similarly to the student who said "I had limited experience writing a computer science research paper and feel my ability has improved significantly." Another expressed that "I feel like I learned a lot about what's expected of scientific writing in this class." Some mentioned how helpful the paper writing guidelines were to them: the

"outlines you provided for us for our project reports were really helpful in starting a foundation for how to write papers like this for computer science." Students also reported that paper reading and discussion helped with writing their own paper.

Finally, we asked students why they took the course, if it met their expectations, and if they had any additional suggestions and feedback.

The two most common reasons for taking the course were to satisfy the Systems requirement for the major and to learn more about PDC. Universally, students enjoyed the course and felt like they got a lot out of it. "I learned a ton and really enjoyed the class." Students who took it just to satisfy a requirement were often surprised by how much they enjoyed the course and how much they learned: "I actually ended up learning a lot more than I expected and understanding a field that I previously hadn't given much thought to. This was by far my most enjoyable CS class yet, even if I did find it quite challenging." Students who took it because of an interest in PDC topics felt like they learned many useful skills, and that it "did a great job of helping me understand these concepts."

Students unanimously enjoyed the class, felt challenged by it, and felt that they learned a lot from it. Paper reading and discussion was the most popular part, one student summarizing a common theme: "I loved the opportunity to have a seminar type CS course, which is a type of course I wished the CS department had more of". The PDC topics were also mentioned as important and "great to learn", and the labs as providing useful practice in PDC.

Overall, students found the course workload a bit high, but manageable and very worthwhile. Over the years we have added more guidelines and help, and reduced some content, but we feel there is still room to tweak the lab and final project workload a little, while still maintaining the primary purpose of these activities.

A. Remote Class Experience

In the spring 2020 offering of the course, we moved to all-remote teaching halfway through the semester due to the COVID-19 pandemic. We asked students about their experiences with the on-line transition and what worked well and not so well. Their responses matched our impressions well. Students thought the lecture and paper discussions transitioned pretty well to on-line. One student noted that the "small interactive seminars are much more conducive for remote learning than larger lectures". Dividing the class in half for paper discussions helped to make this transition to on-line easier than in a large lecture.

Students felt that labs and projects were the most difficult to do remotely, some expressing they were "much more difficult" to do in this setting. In response to moving to all-remote instruction we offered students a second course project option that was an in-depth research report with little to no implementation and testing. Most groups chose this option, and those that did felt that the original project requirements would have been much more difficult to satisfy working remotely.

Our observations agree with students. Overall, the course transitioned fairly well to on-line. We had the advantage of the first half of the semester being in-person, so students were already used to working in groups. The more seminar-style parts of the course transitioned better, as did the more pure lecture content. Final project presentations also worked very well remotely. In-class group problem solving was the least satisfying on-line. We used Zoom break-out sessions, which functionally worked well, but were very isolating, with no easy way for us to read the room, suggest hints or directions to the entire class, listen into group's problem solving in a non-intrusive way, or easily identify groups that were stuck or not functioning well. We had students use google docs in Zoom break-out sessions as a way to partially address these challenges, but it was the part of the class where both students and instructors noticed a significant loss from the in-person experience.

A few groups attempted the original course project that includes implementation and testing, and were successful in completing strong projects. Most groups, however, chose the alternate option. The groups that attempted the original project expressed how difficult it was, and felt they were not able to make as much progress as they would have in-person. We also found it more difficult to help students on these types of projects in an all-remote setting. Overall, we were surprised by the high quality of final projects this semester, under the circumstances of an unexpected remote semester, and also with the college going to an all pass/fail grading for the semester, which may have removed some motivation for devoting a lot of time and effort to projects. The final presentation may have helped motivate them some, but we believe that the fact that students pick a project topic that they are interested in learning more about was a primary motivation for them to work hard.

Due to its structure, our course may be one that is easier to transition to all on-line than a more traditional course with lecture, labs, and exams. We also have no doubt that the transition was easier due to having students in-person for the first half of the semester. An only remote offering would have been much more difficult for both students and faculty, and given our observations of the transition, we are certain that students would not have learned as much in an all on-line offering.

V. Instructor Evaluation

Overall, we feel that our course is designed well to meet its learning goals, and to cover much of the breadth of PDC while still providing depth. Students like the mix of lecture, labs, and paper reading and discussion, and we plan to keep this mix as it is. One planned improvement to in-class problems is to work on the timing to better ensure that

they are completed within a single class meeting; particularly because the lecture portion meets only once per week, it is difficult for groups to pick up from where they left off. Reducing the number of questions with each exercise, and being a bit more mindful of timing, will help us with this in the future.

The paper readings and discussions and final projects are the most satisfying parts of the course, both to the students and the instructor. We observe large improvements in students' reading, writing, analyzing and oral presentation in this class. Although some students wish for all seminar style, most appreciate the mix of lecture and paper discussion, and we also feel that this works well for an undergraduate level course, particularly one in a CS curriculum without a deep prerequisite hierarchy.

We have had difficulty finding an appropriate textbook for this course, as there is not one alone that covers the breadth of topics in our class. In the past we used Lin and Snyder [44] for its focus on analyzing scalability of parallel algorithms and including analyses of memory and synchronization costs. The difficulty is that this content fits only a small portion of our course's topics, and students are unhappy paying for a textbook that is not used on a weekly basis. Currently, we use a mix of our own and other on-line resources, textbooks, tutorials, and references [45]–[48].

We are often amazed by what students accomplish in this class. All students improve in their confidence and abilities participating in class, and in their abilities to problem solve, analyze solutions, and ask important questions in PDC. We also have had several students do further work on their course project and get CS research paper or poster publications from their efforts. For students pursuing graduate studies in particular, this course is good preparation with its focus on reading, writing, presentations, and designing and carrying out an independent project.

VI. Lessons Learned

This is not a course that we would have developed had it not been for some of the constraints imposed by a small liberal arts college. In fact, trying to do it all is a recipe for failure. However, there are a few keys to making this successful.

The first is being willing to give up a lot of important topics and interesting and cutting edge papers that you would like to discuss. It will hurt, but it is obviously impossible to do it all, and it works much better to focus on accessible papers across the breadth of topics. Often these are survey papers or more foundational papers on a topic. Fewer papers also ensures students have time to read research papers appropriately for understanding and to be prepared for in-depth discussion. Over the years we have cut the total number of papers assigned, and the result is that students are better at reading and discussing them.

The second is designing the course around an overarching theme to link the topics and provide a common depth of focus. We chose scalability as a common theme that we apply over and over to a broad range of topics and contexts.

The third is to ensure multiple opportunities for depth within the main breadth of topic coverage. A course that is all high-level survey will not develop the same parallel and distributed thinking and analysis skills. The course project is the primary way in which we achieve depth of coverage, but paper reading and discussion, in-class problem solving exercises, the focus of some lecture content, and lab assignments also provide depth.

The fourth is to give students practice with a wide range of PDC programming languages and tools. This helps students to problem solve in different contexts, and to compare different types of systems and applications. Teaching just a single language or paradigm does not as easily develop more broad PDC problem solving skills.

Finally, we suggest instructors tailor the course to their specialty or interests. Although a course like this includes broad coverage of systems, algorithms, tools, and languages, there is still an opportunity to focus more generally on areas of your expertise or ones in which you want to learn more. For us, systems has been a larger focus, but algorithms, languages, or applications could equally be the larger focus. We also try to incorporate one or two papers or lectures on topics of specific interest to us. This is a good way to provide depth in a subfield, to spark student interest in your own work, and to provide students with some background instruction to prepare them for projects based on your research area.

One of the most difficult parts of designing a course like this is giving up some content and important pedagogical experiences in order to maintain a good balance between breadth and depth. Identifying and focusing on the primary learning outcomes helps to prevent introducing a workload that is counter productive to these goals.

VII. Conclusions

We presented the design and implementation details of our undergraduate-level course on parallel and distributed computing. Our course covers a large breadth of these two immense fields, while also developing depth in PDC thinking, problem-solving, programming, and analysis skills. Its active learning exercises, paper reading and discussion, and large course project further develop important skills for students entering graduate school or the workforce.

Although our course was initially designed due to constraints of CS in a small institution, we like the breadth of coverage in this single course, and find its design to be a good undergraduate-level course introducing these fields. From our 10 years developing and teaching our "doing it all in one course" curriculum, we feel that a course like ours would work well at any institution because it provides a large

view of two huge fields that undergraduate students may not otherwise see, and its design includes many opportunities for depth in specific topics and learning; this is an undergraduate course we would want to teach regardless of the constraints of our institution.

REFERENCES

[1] ACM/IEEE-CS Joint Task Force, "Computer science curricula 2013," www.acm.org/education/CS2013-final-report.pdf, 2013.

[2] The NSF/IEEE-TCPP Curriculum Working Group, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing - core topics for undergraduates," http://www.cs.gsu.edu/~tcpp/curriculum/, 2012.

[3] CS87: Parallel and Distributed Computing Course Webpages, http://www.cs.swarthmore.edu/~newhall/cs87.

[4] C. M. Brown, Y.-H. Lu, and S. Midkiff, "Introducing parallel programming in undergraduate curriculum," in *Parallel and Distributed Processing Symposium Workshops and PhD Forum IPDPSW, 2013 IEEE 27th International*, May 2013.

[5] M. Grossman, M. Aziz, H. Chi, A. Tibrewal, S. Imam, and V. Sarkar, "Pedagogy and tools for teaching parallel computing at the sophomore undergraduate level," *J. Parallel Distrib. Comput.*, vol. 105, no. C, Jul. 2017.

[6] R. Muresano, D. Rexachs, and E. Luque, "Learning parallel programming: a challenge for university students," *Procedia Computer Science*, vol. 1, no. 1, 2010.

[7] T. Newhall, A. Danner, and K. C. Webb, "Pervasive parallel and distributed computing in a liberal arts college curriculum," in *Journal of Parallel and Distributed Computing*, vol. 105, July 2017.

[8] D. J. John and S. J. Thomas, "Parallel and distributed computing across the computer science curriculum," in *Parallel and Distributed Processing Symposium Workshops IPDPSW, 2014 IEEE International*, May 2014.

[9] C. T. Delistavrou and K. G. Margaritis, "Towards an integrated teaching environment for parallel programming," in *2011 15th Panhellenic Conference on Informatics*, Sept 2011.

[10] J. Adams, R. Brown, and E. Shoop, "Patterns and exemplars: Compelling strategies for teaching parallel and distributed computing to CS undergraduates," in *Parallel and Distributed Processing Symposium Workshops and PhD Forum IPDPSW, 2013 IEEE 27th International*, May 2013.

[11] Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER), https://tcpp.cs.gsu.edu/curriculum/?q=node/21183.

[12] R. Brown, L. Shoop, and J. Adams, "CSinParallel," https://csinparallel.org.

[13] D. Bunde and J. Mache, "Teaching parallel computing with higher-level languages and activity-based laboratories," http://faculty.knox.edu/dbunde/parallel.html.

[14] M. A. Kuhail, S. Cook, J. W. Neustrom, and P. Rao, "Teaching parallel programming with active learning," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018.

[15] S. Srivastava, M. Smith, A. Ghimire, and S. Gao, "Assessing the integration of parallel and distributed computing in early undergraduate computer science curriculum using unplugged activities," in *2019 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, 2019.

[16] S. Matthews, "PDCunplugged: A free repository of unplugged parallel and distributed computing activities," in *2020 IEEE/ACM Workshop on Parallel and Distributed Computing Education (EduPar-20)*, 2020.

[17] S. K. Ghafoor, D. W. Brown, M. Rogers, and T. Hines, "Unplugged activities to introduce parallel computing in introductory programming classes: An experience report," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education ITiCSE'19*, 2019.

[18] R. Wright and J. Boggs, "Learning cell biology as a team: A project-based approach to upper-division cell biology," *Cell biology education*, vol. 1, 2002.

[19] The Top 500 and Green 500 Lists, https://www.top500.org.

[20] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, Nov. 1984.

[21] D. D. Clark, "The design philosophy of the darpa internet protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, Jan. 1995.

[22] J. J. Dongarra, S. W. Otto, M. Snir, and D. Walker, "A message passing standard for mpp and workstations," *Commun. ACM*, vol. 39, no. 7, Jul. 1996.

[23] L. Dagum and R. Menon, "OpenMP: an industry standard api for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, 1998.

[24] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?" *ACM Queue*, vol. 6, no. 2, Mar. 2008.

[25] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *IEEE Computing in Science Engineering*, vol. 12, no. 3, 2010.

[26] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, Honghui Lu, R. Rajamony, Weimin Yu, and W. Zwaenepoel, "Treadmarks: shared memory computing on networks of workstations," *IEEE Computer*, vol. 29, no. 2, 1996.

[27] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting +++Systems Design & Implementation - Volume 6*, ser. OSDI'04. USENIX Association, 2004.

[28] M. Zahran, "Heterogeneous computing: Here to stay: Hardware and software perspectives," *ACM Queue*, vol. 14, no. 6, Dec. 2016.

[29] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[30] D. Abts and B. Felderman, "A guided tour through data-center networking: A good user experience depends on predictable performance within the data-center network." *ACM Queue*, vol. 10, no. 5, May 2012.

[31] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, Apr. 2010.

[32] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, Jul. 1982.

[33] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, and M. Stumm, "Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Oct. 2014.

[34] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, Feb. 2003.

[35] W. K. Steinmetz R., "Chapter 2: What is this peer-to-peer about?" https://link.springer.com/chapter/10.1007/11530657_2, 2005.

[36] B. W. Lampson, "Computer security in the real world," *IEEE Computer*, vol. 37, no. 06, jun 2004.

[37] E. H. Spafford, "The internet worm: Crisis and aftermath," *Commun. ACM*, vol. 32, no. 6, Jun. 1989.

[38] T. Mattson and M. Wrinn, "Parallel programming: Can we please get it right this time?" in *2008 45th ACM/IEEE Design Automation Conference*, 2008, pp. 7–11.

[39] L. Lamport, *Time, Clocks, and the Ordering of Events in a Distributed System*. Association for Computing Machinery, 2019.

[40] NVIDIA, "NVIDIA CUDA Compute Unified Device Architecture," https://developer.nvidia.com/about-cuda, 2018.

[41] A. Danner, T. Newhall, and K. Webb, "A Library for Visualizing and Debugging Parallel Applications," in *Proc. of 9th NSF/TCPP Workshop on Parallel and Distributed Education (EduPar-19)*, 2019.

[42] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. Scott, and N. Wilkins-Diehr, "Xsede: Accelerating scientific discovery," *Computing in Science and Engineering*, vol. 16, no. 05, sep 2014.

[43] M. De Wael, S. Marr, B. De Fraine, T. Van Cutsem, and W. De Meuter, "Partitioned global address space languages," *ACM Comput. Surv.*, vol. 47, no. 4, 2015.

[44] C. Lin and L. Snyder, *Principles of parallel Programming*. Addison-Wesley, 2008.

[45] Prasad, Gupta, Rosenberg, Sussman, and Weems, *Topics in Parallel and Distributed Computing: Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms*. Springer, 2018.

[46] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 1st ed. Arpaci-Dusseau Books, August 2018.

[47] S. J. Matthews, T. Newhall, and K. C. Webb, "Dive into Systems," https://diveintosystems.org/, 2020.

[48] B. Barney, "Introduction to Parallel Computing," https://computing.llnl.gov/tutorials/parallel_comp.