Codeless PDC Modules for Early Computing Curriculum

Chris Bourke
School of Computing
University of Nebraska–Lincoln
Lincoln, NE USA 68588-0115
chris.bourke@unl.edu

Justin Firestone
School of Computing
University of Nebraska–Lincoln
Lincoln, NE USA 68588-0115
justin.firestone@unl.edu

***Abstract*—PDC at UNL is a series of "codeless" modules consisting of visualizations, simulations, and demonstrations which introduce Parallel and Distributed Computing (PDC) concepts in early computing courses. These materials are codeless because they do not require students to write or understand code. Instead, students read a short introduction to a PDC concept and then engage with a web-based visualization and/or (code-based) demonstration reinforcing the concept. The codeless nature of these modules makes them suitable for computing and non-computing majors.**

To test the effectiveness of our modules we introduced them into two CS1 courses and designed and administered a pre/posttest. Our results show statistically significant results: those who engaged with our modules substantially improved their knowledge and understanding of PDC concepts. Our modules also improved student attitudes, confidence and self-efficacy with respect to PDC topics. We also provide some qualitative observations of our study and identify common misconceptions students have about PDC.

***Index Terms*—parallel computing, distributed computing, computer science education**

I. Introduction

It has been more than a decade since the initial release of the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing (PDC) guidelines [1], [2] and there have been many initiatives and efforts to introduce basic PDC concepts at various levels of computing curricula [3], [4]. Similarly, there are a growing number of research papers, workshops, and innovative assignments being developed and shared. However, the main focus of these efforts has centered around intermediate and upper-level courses and related electives. The PDC 2012 guidelines themselves only identify a small number of topics and subtopics as being appropriate for a Computer Science 1 (CS1) course. Furthermore, they do not contemplate non-computing majors who could also benefit from early exposure to PDC concepts.

This approach is understandable because practical application of PDC knowledge requires a relatively high-level programming maturity and understanding of system operations, concepts which early computing students are unlikely to have encountered in their first or second year of studies. Nevertheless, we believe there is an opportunity to introduce basic PDC concepts earlier in the computing curriculum and even to non-computing majors, but only if the material is thoughtfully designed to be accessible to such students. Early computing courses such as CS0 or CS0.5 can serve as an entry point for computing majors or also as a recruiting tool for new majors or minors. This suggests there could be a significant impact from introducing PDC concepts in earlier courses.

To test this possibility, we have developed a series of "codeless" modules designed to introduce basic PDC concepts to introductory computing students whether they are majors or not. We describe our approach as "codeless" because the activities do not require students to write or even understand any code, making the activities accessible regardless of programming knowledge or computing background. The modules rely heavily on interactive visualizations and simulations which demonstrate PDC concepts that can run in any web browser without installing special or additional software.

Other module activities include practical demonstrations using fully implemented programs in two different languages: C and Java. The different language versions should make the materials useful for a variety of courses. Full instructions on compiling and running the programs is provided, written with a focus on clarity and readability. Software dependencies have been minimized and no specialized runtime environment or system is necessary. Compilation will necessarily require a development environment with a compiler and a runtime environment on a multicore system. We designed the modules to be modular so instructors could pick different subsets of the activities depending on available resources and which courses they are teaching.

Our overall goal in designing these modules is to make PDC concepts accessible as early as possible. To evaluate the effectiveness of our modules in introducing PDC concepts, we designed a pre/post test evaluation and administered them in two introductory computing (CS1) courses. At the center of this study are the following research questions.

- Q1: Are our codeless modules effective in introducing PDC concepts in introductory or even intermediate computing courses?
- Q2: Does introducing PDC concepts early in the curriculum effect student attitudes and awareness of PDC concepts?

In the next section, we discuss prior related work. Section III describes each of our modules in detail along with their

structure. Finally, Section IV outlines our research design and its results.

II. Related Work

A large number of publications have documented various challenges and successes of teaching PDC in undergraduate curricula including multiple survey papers [5]–[7]. However, the bulk of these efforts have focused on designing assignments and strategies for teaching PDC to more advanced students. Saule [8] details a series of scaffolded programming assignments designed for a junior-level class but suggests some of them could be adapted for use in earlier courses such as CS1. Liu [9] describes their longterm experiences teaching PDC to senior-level students.

At the junior-level, there have been several different innovative approaches to teaching PDC which depart from a traditional student experience. For example, several instructors have applied a flipped model which leverages pre-class video lectures to maximize classroom time devoted to practical exercises [10], [11]. Other authors have advocated for embedding smaller PDC modules throughout their entire computing curriculum [12], [13] to reinforce concepts.

We are not the first to suggest possible methods and benefits of introducing PDC concepts earlier in the computing curriculum as well as for non-computing majors. Rague [14] developed an early visualization tool to reinforce parallel thinking for CS1 students. Gil [15] developed a data science curriculum intended for non-computing majors that did not expect or require programming skills. This approach implemented a drag-and-drop visual programming tool to manage data workflows.

Others have investigated different ways to actively engage students without significant coding experience. Neeman *et al.* [16] outline a strategy for teaching PDC concepts using natural-language analogies instead of specific technologies. Bogaerts [17] introduced parallel programming through Scratch, a block-based visual educational tool intended for students from ages 8 to 16. Bruce *et al.* [18] introduced concurrency into a CS1 course in the context of event-based programming through graphics and animation. Srivastava *et al.* [19] developed and analyzed how to use "unplugged" activities which do not require students to use computers at all.

Lastly, Tanaka *et al.* [20] discuss the challenges of needing access to, and familiarity with high-performance computing (HPC) resources to teach PDC concepts. Interestingly, their proposed framework, WRENCH, leverages containerization and a web-based application to allow students to simulate five different pedagogic activities related to networking, distributed continuous integration workflows, data locality, parallelism, and resource provisioning. Although the authors suggest their WRENCH framework could be used to introduce PDC concepts to first-year students, their initial participants were juniors in an operating systems course.

III. Codeless Modules

We developed three "codeless" modules to introduce PDC concepts in early computing courses. Each module is structured as follows: students are asked to read a short introductory text explaining the concepts which will be demonstrated through simple browser-based activities. They then are asked to engage in interactive visualizations or simulations which do not require them to develop or understand any code. Some of the modules include an additional demonstration which requires the student to compile and run a program but does not require them to write, edit, or read any code. Finally, students are asked to reflect on the concepts by answering a series of questions.

A live version of the modules is available at https://go.unl.edu/PDCatUNL and the code with other artifacts are hosted at https://github.com/cbourke/PDCatUNL.

A. Structure

The reading material has been written at a level that does not require a deep understanding of PDC concepts. The material deliberately omits many details that would be necessary to gain expertise, and instead favors describing concepts in a more abstract manner which should make them more accessible to a wider audience. For example, we do not distinguish between data-parallel and task-parallel models, nor do we provide rigorous definitions of cores, threads, or processes. However, we do mention these concepts in a way that early computing students should be able to relate to and understand.

The simulations include both web-based visualizations and pre-written, code-based simulations. The web-based visualizations are interactive applications inspired by the highly successful utilization of similar technologies introduced to visualize programs [21], data structures and algorithms [22], and automata [23]. They can be run on any web browser without special environments or software installations. The code-based simulations include both C and Java versions to accommodate students in different introductory courses. However, we still describe them as "codeless" because students are not required to write or design any code. They are required to compile and run the programs, but we include complete, minimal instructions similar to what they should have already seen in other classes with programming assignments. It is unavoidable to require a proper C or Java environment for compiling and running these simulations, but we have tried to minimize external dependencies as much as possible.

At the end of each module, students are prompted to reflect on the reading and their interaction with the browser-based simulation and demonstration by answering a series of questions designed to ensure that: 1) they ran a sufficient number of use cases in each simulation; and 2) their simulation experience directly traces back to the reading which is intended to reinforce understanding. For example, they are asked how much time a simulation took to execute with different numbers of threads and are then asked to deduce specific *reasons* for trends they saw (*i.e.*, seeing a roughly double speed-up when doubling the threads, but not seeing an improvement beyond the number of cores available on the system). To help understanding and avoid misconceptions, suggested answers are provided based on our own simulations.

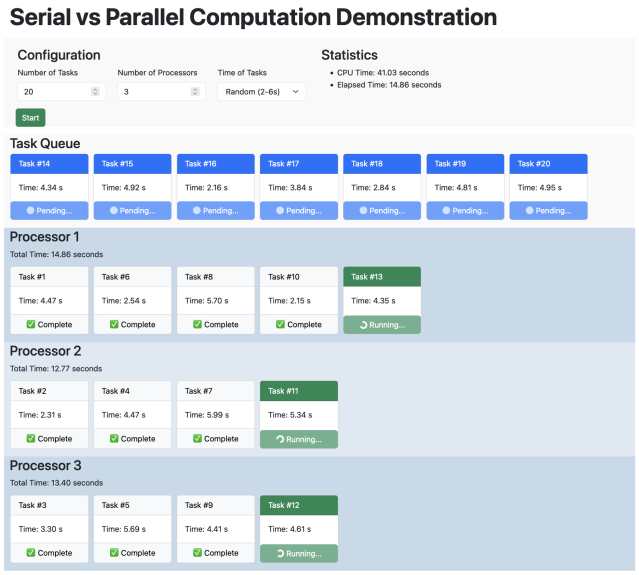

Fig. 1. A screenshot of Module 1.0's Parallel Computing Demo

This reflects our intent for the modules to be a formative, rather than summative, assessment.

Module 1.0: The first module introduces students to asynchronous and parallel computing through several web-based visualizations which contrast with sequential and synchronous computing. The consequences of each type of computing are highlighted in the visualizations and demonstrations. In one example, students are given a browser-based GUI which executes a synchronous, blocking action that freezes the UI, and an asynchronous action which does not.

The main demonstration is a parallel computing simulation and visualization in which the user can specify a number of generic computing tasks, a number of processors to execute them, and how long each task will take (*e.g.*, 2-6 seconds). The simulation then queues the tasks for each processor to execute in order. The simulation reports the status of each task as well as total wall-clock time compared to processor time. Figure 1 contains an example screenshot of the visualization.

Module 2.0: The second module introduces students to the concept of threads. It is based on a brute-force, dictionary-based, password-cracking scheme which uses an "embarrassingly parallel" approach. We give students a basic explanation how password hashing works to motivate the activity as well as provide early exposure to security issues such as the importance of strong passwords. They are provided both sequential and parallel password-cracking programs with a C version using `pthreads` and a Java version using `Callables`. We include full instructions on cloning, compiling, and running the programs, as well as how to read and interpret the results. No actual coding is necessary to run and observe the simulations.

The demonstration exposes students to the potential performance improvements of parallel computation and its limits. They observe (roughly) proportional speed ups until the number of threads exceeds the number of processors after

```
creating 4 threads...DONE
Starting threads...
Starting thread 1...
Starting thread 2...
Thread 2 hashing [61851, 123702) = [down, mallorca]...
Starting thread 3...
Thread 1 hashing [0, 61851) = [a, downscaled]...
Starting thread 4...
Thread 3 hashing [123702, 185553) = [mallorcan, rotc]...
Thread 4 hashing [185553, 247406) = [rotch, zzz]...
Thread 1 did not crack password
Thread 4 cracked password: 0x618e0fcd...b0b => zzz42
Signaling other threads to terminate...
All threads DONE
Time:
218.541u 0.015s 0:54.82 398.6% 0+0k 4888+0io 0pf+0w
```

Fig. 2. Example run of parallel password cracking. The contents of a dictionary are split among a specified number of threads ("down" to "mallorca" for example).

which no further improvement is observed. When they run the program, they can specify how many threads to create. They are instructed to run the program multiple times with 1, 2, 4, *etc.* threads and observe the trends. An example run of the C version's output can be found in Figure 2.

We note that the demonstration is data-parallel because the dictionary listings are split evenly among each thread, so we chose password hashes which ensures the program does not get "lucky" by finding the hash early in any particular data slice, regardless of how many threads are specified.

Module 3.0: The final module introduces students to a producer-consumer pattern. Students are provided another browser-based visualization as well as a C and Java implementation to investigate. In all versions, generic tasks represent requests or work which are simulated through HTTPS requests to a web-service that causes a delay before responding. The length of the delay can be specified or randomized.

All of the simulations allow students to specify the number of producers and of consumers. Once started, each producer will generate tasks at random intervals and place them into a task queue for processing. When a consumer is available, it takes the next available task from the queue and executes it. The simulation will continue until stopped by the user. An example screenshot of the web-based visualization can be found in Figure 3.

As with the other modules, the questions point students to observe several scenarios, such as having equal or unbalanced numbers of producers or consumers. Students are then asked to predict the consequences of those different scenarios.

B. PDC 2012 Mapping

We designed the modules to target early computing (CS1) and non-computing major (CS0 or CS0.5) students. Our intention was to make the materials and PDC concepts accessible to a wide variety of students, which is why we use a "codeless" approach. We also wanted to explain the topics from a higher, abstract level for simplicity.

To help determine whether we met our goals, we utilized

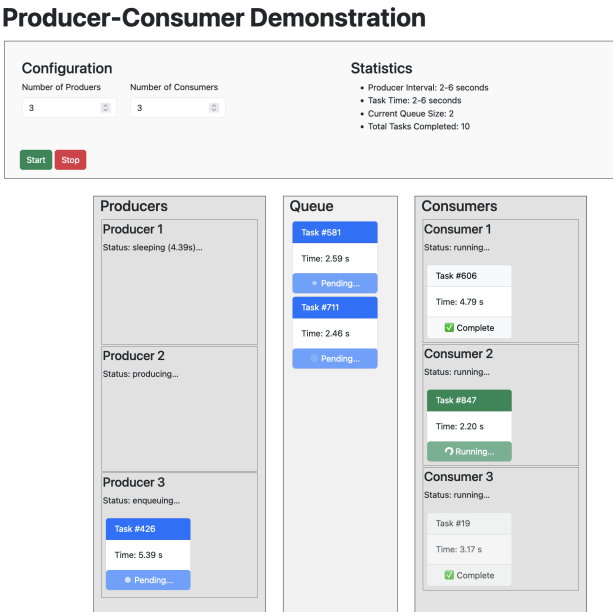

Fig. 3. A screenshot of Module 3.0's Producer-Consumer Demo

the CS Materials[1] application [24], [25] which allows CS educators to categorize materials using the ACM-CSC 2013 or PDC 2012 curriculum [2] guidelines and visualize and compare materials.

When we used this tool, we identified two primary topics from PDC2012 covered by our modules:

- "Why and what is parallel or distributed computing?" (Cross Cutting and Advanced Topics::High level themes, Bloom: Knowledge, CS1/2)
- Speedup (Parallel and Distributed Models and Complexity, Bloom: Comprehend, DS/A)

We observe that the two topics are on the lower end of Bloom's taxonomy which makes them appropriate for a primer on PDC topics. The first topic is clearly designed for the targeted course levels, and although the second topic is designated for a Data Structures & Algorithms course, we believe it is appropriate for the target course because of the accessible and "codeless" approach.

Our modules do not cover the other 7 CS1 topics designated by PDC2012, but we view this as sufficient coverage for PDC. Among the other 7 topics, 3 involve floating-point representation and 3 address more general computing literacy such as P2P systems and web searching, which are generally or implicitly covered in a typical introductory course (or earlier) and not necessarily within the context of PDC.

As the usage of this tool becomes more common, it will be possible to compare our modules to other PDC materials.

IV. Results & Analysis

To address our research questions and analyze how effectively our modules introduced PDC concepts, we incorporated them into two 15-week CS1 courses at a medium sized R1 university. Both courses consisted primarily of computing-majors (CS, CE). One was a larger "main" section (about 100 students) that used C while the second was a smaller honors section (25 students) that used C and Java.

All students in both sections were required to complete a pre/post test that was graded based on completion and constituted about 4% of their final grade. The pretest was administered in week 3. Students were then directed to go through the modules at their own pace by week 12 and the posttest was administered and due at the end of that week.

Since this study involved human subjects, we received approval from our institution's Institutional Review Board (IRB). Students were given the option of opting in or out of their data being used, although their participation was required as part of the course grade. After filtering students who opted out, as well as those who withdrew or failed to submit either test, we had $n = 88$ participants in our data set (69 from the main section and 19 from the honors section).

A. Pre/Posttest Design

The pretest and posttest featured the same questions which were intended to gauge students' attitudes toward computing, PDC concepts, and their basic knowledge and understanding of those concepts. The posttest contained additional questions to measure students' self-efficacy and confidence in their answers.

To measure attitudes, students were first asked to agree or disagree with the following statements:

A1 In general, my interest in computing is high
A2 My interest in learning about parallel and distributed computing is high
A3 In general, I believe learning about parallel and distributed computing is important
A4 I believe I have a good understanding of computing
A5 I believe I have a good understanding of parallel and distributed computing

Student answers were collected using a 7-point Likert scale (7 = strongly agree, ... 4 = neutral, ... 1 = strongly disagree).

To measure students' knowledge and understanding of PDC concepts, they were asked to provide free-form text answers to the following questions. For each of the four identified PDC concepts: sequential (S), asynchronous (A), parallel (P), distributed (D) computing, students were asked to provide:

1) a definition (D) of the concept; and
2) an example (E) of the concept

We'll refer to these 8 questions as Q-SD, Q-SE, Q-AD, Q-AE, Q-PD, Q-PE, Q-DD, Q-DE. For example, Q-PD refers to the question: "Provide a definition for parallel computing."

To encourage honest feedback, students were clearly explained that correctness of their answers would *not* impact their grade, which was based entirely on completing the pre/posttests.

[1] https://cs-materials.herokuapp.com/, specifically https://cs-materials.herokuapp.com/material/1784

B. Assessment

Since student answers were free-form text, they needed to be manually assessed. Both authors independently scored each answer based on the following 4-point Likert scale.

1. Needs Improvement - The student answer did not clearly demonstrate that they had an understanding of the concept.
2. Developing - The student seems to have a beginner's grasp on the concept but there are key issues or omissions in their answer that indicates they do not yet fully understand the concept.
3. Adequate - The student has a good grasp on the concept. There may be some insufficiencies or omissions from their answer but overall, they have demonstrated sufficient understanding of the concept.
4. Above Average - The answer goes above basic understanding and indicates a deeper appreciation of the concept.

Student understanding was not expected to be anywhere close to the level of expert understanding or even an advanced undergraduate student.

After scoring all questions, a second evaluation was performed for any score which differed by more than 0.5 points. After this analysis we reached a consensus for nearly all scores. For our analysis, we used a mean between the two scores.

C. Performance

In the posttest students were asked to self-assess their level of engagement with the modules. We again emphasized that their answers would have no effect on their grade in an effort to encourage honesty. We asked whether they had completed all, most, some, not many, or none of the modules. It was our intention that (self-identified) lower-engagement students would provide a *de facto* control group with which to analyze the effectiveness of our modules. 75 students (experimental group) identified as high-engagement (having completed all or most of the modules) and 13 identified as low-engagement (control group).

Because our assessments used a Likert scale, it was unlikely to be normally distributed, a result we confirmed with a Shapiro-Wilks test. As a consequence, we used a Wilcoxon Signed-Rank test [26] with $\alpha = 0.05$ for all our analysis.

To determine whether our modules had an effect on student learning and understanding of PDC concepts, we compared the pretest and posttest scores for each of the 8 questions for all students (Table I) as well as a separate analysis for high-engagement (Table II) and low-engagement (Table III) students.

The results were generally very positive. Students improved in their understanding of all four PDC concepts as evidenced by significantly improved scores for all 8 questions. The Wilcoxon analysis indicates this improvement is statistically significant for all 8 questions (all p-values were $< \alpha = 0.05$). The most substantial improvements can be seen in both sequential and parallel computing. This arguably makes sense because the nature of the course in which the students participated focused on sequential computing, whereas the modules focused on parallel computing. The module related to distributed computing was designed to simply expose students to that topic, which could explain why there was a relatively modest improvement in that area.

The separate analysis comparing the experimental and control groups indicates even stronger results. Similar, or slightly better, improvements were observed for the experimental group and all are statistically significant. However, the control group (low-engagement) students show only moderate improvements in understanding and only 2 of the 8 questions are statistically significant (Q-SE and Q-AD). The sequential computing understanding gains (Q-SE representing the stronger gain among the two) is again arguably understandable

TABLE I
PERFORMANCE - ALL. THE PERFORMANCE ANALYSIS FOR ALL STUDENTS ($n = 88$) FOR EACH OF THE 8 QUESTIONS. THE (PRE)-TEST AND (POST)-TEST SCORES ARE MEANS. THE (DIFF)ERENCE IN MEANS IS REPORTED, WITH POSITIVE VALUES INDICATING IMPROVEMENT. THE WILCOXON STATISTIC (W.) IS REPORTED ALONG WITH ITS P-VALUE.

	Pre-	Post-	Diff	W.	p-value
Q-SD	1.639	2.565	0.926	224	2.13e−10
Q-SE	1.252	2.198	0.946	76	6.29e−13
Q-AD	1.218	2.159	0.940	110	3.08e−11
Q-AE	1.090	1.872	0.781	127	2.26e−10
Q-PD	1.250	2.409	1.159	126	7.20e−13
Q-PE	1.071	1.840	0.769	113	2.05e−12
Q-DD	1.329	2.014	0.684	359	3.74e−7
Q-DE	1.289	1.477	0.187	588	0.0149

TABLE II
PERFORMANCE - EXPERIMENTAL (HIGH-ENGAGEMENT). THE PERFORMANCE ANALYSIS FOR THE EXPERIMENTAL GROUP ($n = 75$; SELF-IDENTIFIED HIGH-ENGAGEMENT STUDENTS).

	Pre-	Post-	Diff	W.	p-value
Q-SD	1.63	2.63	1.00	130	1.12e−9
Q-SE	1.27	2.28	1.01	58	1.31e−11
Q-AD	1.21	2.19	0.98	81	4.82e−10
Q-AE	1.09	1.92	0.83	80	1.04e−9
Q-PD	1.27	2.49	1.22	42	4.30e−12
Q-PE	1.06	1.90	0.84	71	7.98e−12
Q-DD	1.32	2.03	0.71	247	2.29e−6
Q-DE	1.28	1.49	0.21	441	0.0138

TABLE III
PERFORMANCE - CONTROL (LOW-ENGAGEMENT). THE PERFORMANCE ANALYSIS FOR THE CONTROL GROUP ($n = 13$; SELF-IDENTIFIED LOW-ENGAGEMENT STUDENTS). ONLY Q-SE AND Q-AD SHOW STATISTICAL SIGNIFICANCE (P-VALUE $< \alpha = .05$).

	Pre-	Post-	Diff	W.	p-value
Q-SD	1.69	2.17	0.48	15	0.108
Q-SE	1.15	1.75	0.60	1	0.013*
Q-AD	1.29	1.98	0.69	3	0.021*
Q-AE	1.12	1.60	0.48	7	0.123
Q-PD	1.33	1.94	0.61	20	0.134
Q-PE	1.12	1.50	0.38	3	0.075
Q-DD	1.39	1.94	0.55	14	0.091
Q-DE	1.37	1.42	0.05	11	0.610

because sequential computing was a focus of the course. The gains in understanding asynchronous computing may be because the modules covered asynchronous computing first, and students who completed "some" of the modules might have only engaged with the asynchronous computing module. Completing the subsequent modules could have helped clarify differences between the four overall topics, especially the differences between parallel and distributed computing.

The stark contrast between the experimental and control group indicates that high-engagement with our PDC modules can effectively improve student understanding of PDC concepts.

D. Confidence & Self-Efficacy

In the posttest, for each pair of questions for each topic, we asked students to gauge their confidence in the correctness of their answers on a 7-point Likert scale (7 = high confidence, 1 = low confidence).

Since the responses were ordinal and we didn't have pretest data to compare to, we ran a Spearman correlation with each question pair's means and self-identified confidence score. A high correlation (near +1) would indicate high-confidence is associated with more correct answers and that low-confidence is associated with more incorrect answers. A low correlation (near 0) would indicate that students are not able to accurately estimate their own understanding of a topic. The results of this analysis can be found in Table IV.

For sequential, asynchronous, and distributed computing topics, all three had a moderate positive correlation and were statistically significantly correlated (p-values < 0.05). This suggests that students have a (reasonably) accurate self-assessment of their own answers, whether correct or not.

There was one outlier for the topic of parallel computing, where students showed the lowest confidence. However, given that this was an emphasis of the modules coupled with the prior analysis that students showed a positive improvement in this topic suggests students might have suffered from an imposter syndrome effect (*i.e.*, that students were knowledgeable in the topic but less confident in their knowledge). The fact that the Spearman analysis shows that there is no statistically significant correlation between scores and confidence reinforces this. None of these results were *negatively* correlated, meaning the participants avoided the Dunning-Kruger effect (being overly confident in incorrect answers yet having low confidence in answers when competent).

We interpret this as a strong indication that our learning goals were achieved. Students gained a good introduction to PDC topics and learned enough to: 1) have a good foundation for future coverage; and 2) have self-awareness that their knowledge is limited and there is much more to learn.

E. Attitudes

Our modules also had a substantial positive effect on students' attitudes and interest in PDC topics. We applied the same Wilcoxon analysis ($\alpha = 0.05$) to student responses to attitude questions A1 - A5 from the pretest to the posttest. The results are presented in Table V.

TABLE IV
CONFIDENCE.

	Mean	S.Coeff.	p-value
Sequential	5.55	0.372	0.0003
Asynchronous	5.22	0.286	0.0069
Parallel	4.91	0.167	0.1189
Distributed	4.95	0.328	0.0017

TABLE V
ATTITUDES & INTEREST.

	Pre-	Post-	Diff	W.	p-value
A1	6.31	6.27	-0.03	600	0.8924
A2	5.19	5.51	0.32	582	0.0107
A3	5.44	5.94	0.50	470	0.0004
A4	5.35	5.07	-0.28	670	0.0649
A5	3.28	4.92	1.64	237	$6.89e{-10}$

Interestingly, students' interest in computing in general (A1) and self-reported understanding of computing in general (A4) remained unchanged (there was no statistically significant difference, p-value ≥ 0.05). However, for all 3 PDC-related questions, students showed substantial and statistically significant increases in interest (A2), belief in importance (A3), and understanding (A5).

F. Limitations & Risks

Our study is limited in its applicability by the usual factors and risks to validity that affect studies like this. Though our dataset size was substantial ($n = 88$), our intervention was limited to one semester and one institution. A larger and more widespread study would be needed to assess more general applicability.

The results may also be at risk due to our methodology of identifying a *de facto* control group. We do not doubt the truthfulness of those who identified as low-engagement because there would be no reason to misrepresent their participation. However, there could be a significant number of students who misidentified as high-engagement, either by overestimating their own participation or because they thought it would affect their grade in some way.

Another issue is the size and proportion of our control group. Although there are general guidelines for the minimum size or proportion of control groups versus experimental groups with parametric tests, we are unaware of any *general* guidelines for the tests we used. Nevertheless, the smaller sample size ($n = 13$) and proportion (14.7%) of our control group may be an additional risk warranting further investigation.

G. Qualitative Observations

As we assessed student answers to the pretest, we observed several common issues, themes, and misconceptions.

- Students' knowledge was significantly deficient in all topics; surprisingly so even for sequential computing. However, they did seem to have an intuition that sequential computing was the "default" approach of computing.

They often associated it with imperative-style programming language constructs (conditions, loops, *etc.*), but were generally unable to provide concrete definitions.
- PDC terms were often mistaken for or confused with software development processes. For example, "distributed" was taken to mean collaboration among a team of software developers working "in parallel."
- Students would often conflate multitasking (multiple different programs running at the same time) with parallel computing. The most common examples included running two user-facing programs at the same time such as a web browser and word processor.
- There seemed to be a higher familiarity with "parallel computing," or at least with the term, than with asynchronous or distributed computing. However, student definitions often conflated parallel with distributed computing. A common misconception was that parallel computing involved distinct machines. There was virtually no mention of terms such as "core" or 'thread."
- Distributed computing was frequently conflated with networking or simply transmitting data from one system to another with peer-to-peer protocols. However, a few students were familiar with projects such as Folding@Home or SETI@Home, suggesting some prior exposure to this topic in high school curricula or personal research.

We observed that these misconceptions were far less common in the posttest. Nevertheless, there remained some persistent issues and common themes.

- Students seemed to have a much better grasp on sequential computing overall, providing good definitions. However when asked for examples, few original ideas were expressed. Most examples were directly from course assignments with little or no explanation for how they were sequential.
- The most common examples students gave were gaming-related analogies. For example, students compared asynchronous computing to completing independent quests in a game, completing a few objectives of each and switching back and forth between quests if they got bored or stuck. Level generation or autosaving a game state on a separate thread were notable examples of parallel computing.
- Parallel computing was still conflated with multitasking, but this was far less common.
- Most students saw the benefits of parallel computing in that it "sped up" the real time required to complete computing tasks. The terms "multiple processors" or "cores" or terms such as "at the same time" were far more common in the posttest. However, very few used or expressed an understanding of the term "thread."
- Most examples were very abstract, referring to terms such as "task" or "work" instead of more concrete examples. This may be a consequence of the abstract, codeless way that modules presented these concepts.
- Although the modules did not focus on distributed computing, students seemed to have a better understanding about how it involves multiple distinct machines, computers, or systems. Far fewer answers conflated parallel programming with distributed computing, and far fewer conflated distributed computing with ancillary concepts such as networking.

V. Conclusion

While most PDC efforts have focused on programming strategies for upper-level and elective courses, we believe there is value and opportunity in introducing these concepts much earlier in the computing curriculum. Our "codeless" approach is a simple, accessible, and scalable way to introduce PDC concepts to both computing majors and non-majors as early as CS0.

Our present study has provided significant evidence that our approach is an effective tool for introducing PDC concepts and topics to novice computing students without introducing too much complexity. Our results show substantial improvements in understanding and attitudes toward PDC concepts. We hope that other educators can adopt or adapt our modules and can replicate our results and success.

Acknowledgments

The first author acknowledges and is thankful to the Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER) for their financial support during the summer 2022 PDC Curriculum Early Adopter Grant and Summer Training Program.

The first author also acknowledges and is thankful to the CS Materials group (National Science Foundation, OAC-1924057) for financial support and training during the summer 2023 CS Materials workshop at UNC-Charlotte.

References

[1] T. C. W. Group, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing – core topics for undergraduates," 12 2012.

[2] S. K. Prasad, A. Gupta, K. Kant, A. Lumsdaine, D. Padua, Y. Robert, A. Rosenberg, A. Sussman, and C. Weems, "Literacy for all in parallel and distributed computing: Guidelines for an undergraduate core curriculum," *CSI Journal of Computing*, vol. 1, no. 2, 2012.

[3] R. Brown and E. Shoop, "CSinParallel and synergy for rapid incremental addition of PDC into CS curricula," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, 2012, pp. 1329–1334.

[4] S. K. Prasad, A. Y. Chtchelkanova, S. K. Das, F. Dehne, M. G. Gouda, A. Gupta, J. JáJá, K. Kant, A. J. L. Salle, R. J. LeBlanc, M. Lumsdaine, D. A. Padua, M. Parashar, V. K. Prasanna, Y. Robert, A. L. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. C. Weems, and J. Wu, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: core topics for undergraduates," *Proceedings of the 42nd ACM technical symposium on Computer science education*, 2011.

[5] J. A. Carneiro Neto, A. J. Alves Neto, and E. D. Moreno, "A systematic review on teaching parallel programming," in *Proceedings of the 11th Euro American Conference on Telematics and Information Systems*, ser. EATIS '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3544538.3544659

[6] N. M. Mwasaga and M. Joy, "Using high-performance computing artifacts as a learning intervention: A systematic literature review," in *Proceedings of the 2nd International Conference on Intelligent and Innovative Computing Applications*, ser. ICONIC '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3415088.3415130

[7] X. Suo, O. Glebova, D. Liu, A. Lazar, and D. Bein, "A survey of teaching pdc content in undergraduate curriculum," *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1306–1312, 2021.

[8] E. Saule, "Experiences on teaching parallel and distributed computing for undergraduates," *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 361–368, 2018.

[9] J. Liu, "20 years of teaching parallel processing to computer science seniors," *2016 Workshop on Education for High-Performance Computing (EduHPC)*, pp. 7–13, 2016.

[10] S. V. Moore and S. R. Dunlop, "A flipped classroom approach to teaching concurrency and parallelism," *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 987–995, 2016.

[11] J. Zarestky and W. Bangerth, "Teaching high performance computing: Lessons from a flipped classroom, project-based course on finite element methods," in *2014 Workshop on Education for High Performance Computing*, 2014, pp. 34–41.

[12] R. Brown and E. Shoop, "Modules in community: Injecting more parallelism into computer science curricula," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 447–452. [Online]. Available: https://doi.org/10.1145/1953163.1953293

[13] D. J. John and S. J. Thomas, "Parallel and distributed computing across the computer science curriculum," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, 2014, pp. 1085–1090.

[14] B. Rague, "Teaching parallel thinking to the next generation of programmers," *Journal of Education, Informatics and Cybernetics*, vol. 1, no. 1, pp. 43—-48, 2009.

[15] Y. Gil, "Teaching parallelism without programming: A data science curriculum for non-CS students," in *Proceedings of the Workshop on Education for High-Performance Computing*, ser. EduHPC '14. IEEE Press, 2014, p. 42–48. [Online]. Available: https://doi.org/10.1109/EduHPC.2014.12

[16] H. Neeman, L. Lee, J. Mullen, and G. Newman, "Analogies for teaching parallel computing to inexperienced programmers," in *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '06. New York, NY, USA:

[24] A. Goncharow, M. Mcquaigue, E. Saule, K. Subramanian, P. Goolkasian, and J. Payton, "CS-Materials: A system for classifying and analyzing pedagogical materials to improve adoption of parallel and distributed computing topics in early CS courses," *Journal of* Association for Computing Machinery, 2006, p. 64–67. [Online]. Available: https://doi.org/10.1145/1189215.1189172

[17] S. Bogaerts, "Hands-on exploration of parallelism for absolute beginners with scratch," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, 2013, pp. 1263–1268.

[18] K. B. Bruce, A. Danyluk, and T. Murtagh, "Introducing concurrency in CS 1," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 224–228. [Online]. Available: https://doi.org/10.1145/1734263.1734341

[19] 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," *2019 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, pp. 17–24, 2019.

[20] R. Tanaka, R. Ferreira da Silva, and H. Casanova, "Teaching parallel and distributed computing concepts in simulation with wrench," in *2019 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, 2019, pp. 1–9.

[21] J. Urquiza-Fuentes and J. A. Velázquez-Iturbide, "A survey of successful evaluations of program visualization and algorithm animation systems," *ACM Trans. Comput. Educ.*, vol. 9, no. 2, jun 2009. [Online]. Available: https://doi.org/10.1145/1538234.1538236

[22] C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards, "Algorithm visualization: The state of the field," *ACM Trans. Comput. Educ.*, vol. 10, no. 3, aug 2010. [Online]. Available: https://doi.org/10.1145/1821996.1821997

[23] S. Rodger, "Learning automata and formal languages interactively with jflap," *SIGCSE Bull.*, vol. 38, no. 3, p. 360, jun 2006. [Online]. Available: https://doi.org/10.1145/1140123.1140270
Parallel and Distributed Computing, vol. 157, pp. 316–330, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S074373152100126X

[25] A. Goncharow, M. Mcquaigue, E. Saule, K. Subramanian, J. Payton, and P. Goolkasian, "Mapping materials to curriculum standards for design, alignment, audit, and search," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 295–301. [Online]. Available: https://doi.org/10.1145/3408877.3432388

[26] J. D. Gibbons and S. Chakraborti, *Nonparametric Statistical Inference*. CRC Press, 2011.