Data-Intensive Computing Modules for Teaching Parallel and Distributed Computing

Michael Gowanlock
School of Informatics, Computing, & Cyber Systems
Northern Arizona University
Flagstaff, AZ, U.S.A.
michael.gowanlock@nau.edu

Benoît Gallet
School of Informatics, Computing, & Cyber Systems
Northern Arizona University
Flagstaff, AZ, U.S.A.
benoit.gallet@nau.edu

***Abstract*—Parallel and distributed computing (PDC) has found a broad audience that exceeds the traditional fields of computer science. This is largely due to the increasing computational demands of many engineering and domain science research objectives. Thus, there is a demonstrated need to train students with and without computer science backgrounds in core PDC concepts. Given the rise of data science and other data-enabled computational fields, we propose several data-intensive pedagogic modules that are used to teach PDC using message-passing programming with the Message Passing Interface (MPI). These modules employ activities that are common in database systems and scientific workflows that are likely to be employed by domain scientists. Our hypothesis is that using application-driven pedagogic materials facilitates student learning by providing the context needed to fully appreciate the goals of the activities.**

We evaluated the efficacy of using the data-intensive pedagogic modules to teach core PDC concepts using a sample of graduate students enrolled in a high performance computing course at Northern Arizona University. In the sample, only 30% of students have a traditional computer science background. We found that the hands-on application-driven approach was generally successful at helping students learn core PDC concepts.

***Index Terms*—Computer Science Education, Data-Intensive Computing, High Performance Computing, Parallel and Distributed Computing**

I. Introduction

There are numerous challenges to teaching parallel and distributed computing (PDC). While there is little consensus on the best way to teach PDC concepts, numerous studies have found that hands-on approaches are highly effective [1]–[4]. Hands-on approaches allow students to design and write software to be executed on a given hardware platform. The execution of programs are analyzed through various performance measures, and through this exercise, students learn key PDC concepts and skills.

Many scientific enterprises require analyzing large volumes of data. There is an increased demand for PDC to be employed for solving data-intensive problems. High performance computing (HPC) is not just a topic studied by computer scientists that touch on fields such as systems, systems software, architecture, algorithms, and other areas. PDC is exploited by domain scientists and other users such that they can carry out their research. Thus, many scientists and engineers need skills in PDC which are motivated by real-world problems. In response, computer science departments have developed curriculum for the fields of Big Data, data science, machine learning, and other data-focused areas [5], [6]. Furthermore, the Big Data topical area has been incorporated into the NSF/IEEE-TCPP PDC curriculum initiative [7].

We outline the motivating context for this work. The School of Informatics, Computing, and Cyber Systems at Northern Arizona University (NAU) has four MS graduate programs, and has a multidisciplinary PhD program in Informatics and Computing. The PhD program is very diverse as it primarily trains computer scientists, and electrical engineers in addition to domain scientists in biology, ecology, and other subfields of these disciplines. Consequently, there is a demonstrated need to train domain scientists in PDC [5], such that they obtain the necessary skills to tackle cutting-edge research questions.

There is much debate regarding whether to use a cluster for pedagogic purposes. Some research in high performance computing pedagogy has motivated using simulation [3], [4], [8]–[11] to avoid drawbacks of using clusters, such as platform-specific configurations. Also, as described in Tanaka et. al. [4], instructors may also spend a significant amount of time troubleshooting student job script submissions to clusters, working with the cluster administrator and performing other duties that are orthogonal to teaching core PDC concepts. In short, using the cluster environment has several drawbacks.

While we are cognizant of the abovementioned challenges with using the cluster environment, these drawbacks must be weighed against the benefits of using clusters. We outline some of these benefits as follows:

1) Graduate students are the target demographic of the pedagogic materials. In our context at NAU, a substantial fraction of graduate students in the Informatics & Computing PhD program (and other domain science graduate programs) will need to use the university's cluster, `Monsoon`, to carry out their research. Consequently, many students have experience using the cluster, or will be required to use it to complete their academic programs.
2) Many research-intensive universities have access to clusters, and institutions in the United States have access to computational resources that can be used for teaching

This material is based upon work supported by the National Science Foundation under Grant No. 1849559.

purposes, such as XSEDE [12]. Hands-on experience with clusters is very valuable for students that require parallel processing to carry out their research. Furthermore, having PDC skills in addition to experience using the cluster environment can lead to alternative career paths for both computer and domain scientists.

3) The drawbacks of using a cluster are not insurmountable when the course scope is limited to core concepts, such as parallelism, scalability, basic communication patterns, and locality (i.e., concepts most useful to students).

Students enrolling in HPC courses may have diverse educational backgrounds (in this paper, this refers to students with and without traditional computer science backgrounds). To address this population, this paper presents data-intensive pedagogic modules that teach distributed-memory computing using MPI [13]. This paper makes the following contributions:

- We motivate the use of data-intensive pedagogic modules in PDC graduate-level education.
- We present five pedagogic modules that focus on data-intensive applications. These applications are used to capture the interest of students with and without a computer science background, and serve to teach core PDC concepts.
- We evaluate the efficacy of the pedagogic modules using students in an HPC class at Northern Arizona University.

Paper organization: Section II presents related work, Section III presents an overview of the modules, Section IV evaluates module efficacy, and Section V concludes the paper.

II. Related Work

PDC courses are often developed around an instructor's teaching and research interests and the goals of an institution's academic programs. Many instructors have reported that hands-on approaches which incorporate programming activities into curricula are key to mastering PDC concepts [1]–[4]. Some authors advocate for using compute clusters in computer science curriculum [14]–[16]. To avoid the drawbacks of using clusters (e.g., they are a shared resource), some instructors have proposed requiring each student use their own personal low-cost portable cluster [17]–[19]. Other scholars have suggested employing simulation instead of clusters to teach PDC [3], [4], [8]–[11].

In addition to the debate around using clusters, there are other concerns related to teaching PDC. For instance, emerging fields such as Big Data require practitioners to use new tools and technologies such as Hadoop and Spark [20]. Cloud computing is another area that is relevant for data scientists [21]. PDC courses may incorporate these emerging topics into their curriculum. Other efforts have examined teaching PDC to domain science students who do not have a traditional computer science background, but require proficiency in PDC to carry out their research objectives [5], [22].

Similarly to the abovementioned related work, we target the cluster environment. Additionally, our pedagogic materials are designed to target the interests of both computer scientists and those in other fields, such as the domain sciences.

III. Data-Intensive Pedagogic Modules

At the time of writing, we have released five modules that are publicly available online [23]. The modules are scaffolded and should be completed in sequence. The modules focus on teaching distributed-memory parallel computing using MPI [13]. We describe the overall goals and the modules below, but refer the reader to the website for more detail [23].

A. Overall Goals

The major topics of the modules are as follows: parallelism, scalability, locality of reference, basic communication patterns, and reasoning about performance beyond asymptotic time complexity. The modules use MPI, as it is the workhorse of HPC. This provides exposure to the low-level details of distributed-memory computing that are beneficial for both computer and domain science students. The module activities require a minimal to moderate amount of time to compute on a cluster, such that students are able to quickly obtain resources and do not need to wait long before obtaining results.

Table I summarizes the student learning outcomes of each of the five pedagogic modules. The learning outcomes are classified into one of three Bloom taxonomic levels [24] that characterize the transition from concrete to abstract concepts.

Table II summarizes MPI primitives used in the modules. We note that these are a basic guideline, as some modules leave aspects of communication to the discretion of the student. This allows for developing creative solutions to the problems and bolsters in-class discussions regarding different algorithmic design decisions.

B. Module 1: MPI Communication

The first module describes basics of MPI communication, and students are introduced to the following MPI primitives: `MPI_Send`, `MPI_Recv` and some of their variants, `MPI_Isend`, `MPI_Wait`, and depending on how they develop their solutions, they may use `MPI_Bcast`.

There are three activities: ping-pong communication, communication in a ring, and random communication. While the first two activities are straightforward, random communication requires students determine how to receive from an unknown sender without using `MPI_ANY_SOURCE`. Then the module requires implementing a solution using `MPI_ANY_SOURCE`. Students compare the implementations, and reflect on differences, such as ease of programmability and efficiency.

C. Module 2: Distance Matrix

Application motivation: Computing the distances between pairs of points (or feature vectors) is common in many data-intensive applications. The DBSCAN [25] clustering algorithm, k-nearest neighbor searches [26], and performing join operations in databases [27] are a few example applications that require computing distances between pairs of points. Also, a variant of the problem is used in the k-means algorithm [28], which is the topic of Module 5. Given the broad range of uses for computing a distance matrix, many students may find utility in this exercise outside of the course.

TABLE I
SUMMARY OF STUDENT LEARNING OUTCOMES FOR EACH OF THE FIVE PEDAGOGIC MODULES. THE OUTCOMES ARE ASSIGNED ONE OF THREE BLOOM TAXONOMIC LEVELS: A-APPLY, E-EVALUATE, AND C-CREATE.

	Student Learning Outcome	Module				
		1	2	3	4	5
1	Implement several canonical MPI communication patterns.	A	-	-	-	-
2	Understand blocking and non-blocking message passing.	A	-	-	-	-
3	Examine how blocking message passing may lead to deadlock.	A	-	-	-	-
4	Understand MPI collective communication primitives.	-	A	E	E	E
5	Understand how how data locality can be exploited to improve performance through the use of tiling.	-	E	-	-	-
6	Understand the performance trade-offs between small and large tile sizes.	-	E	-	-	-
7	Utilize a performance tool to measure cache misses.	-	A	-	-	-
8	Understand how various algorithm components scale as a function of the number of process ranks.	-	E	E	E	C
9	Understand how different input data distributions may impact load balancing.	-	-	E	-	-
10	Discover how compute-bound and memory-bound algorithms vary in their scalability.	-	E	E	E	E
11	Understand common patterns in distributed-memory programs (e.g., alternating phases of computation and communication).	A	A	E	A	C
12	Reason about performance based on algorithm characteristics (i.e., beyond asymptotic performance).	-	-	E	E	E
13	Reason about performance based on communication patterns and volumes.	-	-	E	-	E
14	Reason about resource allocation alternatives.	-	-	A	E	C
15	Reason about how the algorithms can be improved beyond the scope of the module.	-	-	C	C	C

TABLE II
USE OF MPI PRIMITIVES IN THE MODULES. SINCE MODULES ARE OPEN-ENDED, SOME PRIMITIVES MAY OR MAY NOT BE EMPLOYED IN A GIVEN MODULE. R-REQUIRED, N-NOT REQUIRED BUT MAY BE EMPLOYED.

MPI Primitive	Module				
	1	2	3	4	5
MPI_Send	R	-	N	-	-
MPI_Recv	R	-	N	-	-
MPI_Isend	R	-	-	-	-
MPI_Wait	R	-	-	-	-
MPI_Bcast	N	-	-	-	-
MPI_Send and MPI_Recv variants	N	-	N	-	-
MPI_Scatter	-	R	-	-	N
MPI_Reduce	-	R	R	R	-
MPI_Get_count	-	-	N	-	-
MPI_Allreduce	-	-	-	-	N

In this module, students compute the $N \times N$ distance matrix on 90 dimensional data points, where N is the number of points. The students are exposed to the `MPI_Scatter` and `MPI_Reduce` primitives. After implementing the standard distance matrix that uses a row-wise data access pattern, the module requires implementing a solution that uses tiling. The students compare the performance of their row-wise vs. tiled solutions, where the latter should outperform the former due to better locality which yields higher cache hit rates. Finally, the students measure cache miss rates using the `perf` performance tool and reason about the performance of their solutions.

In this module, students also learn about compute-bound algorithms that achieve high parallel efficiency. Additionally, students will need to think about locality, thereby introducing basic computer architecture concepts.

D. Module 3: Distribution Sort

Application motivation: Sorting is a subroutine in many algorithms [29], and data-intensive workloads. Sorting is used in database systems [30], such as computing top-k database queries [31], and is employed in scientific applications [32], [33]. Sorting is a good vehicle for teaching several fundamental PDC concepts, and is often used in introductory computer science curricula [34].

This module exposes students to a bucket sort [35], which is a sorting technique that performs well in distributed-memory. The module requires using `MPI_Send` and `MPI_Recv` (and/or its variants), `MPI_Reduce`, and possibly `MPI_Get_count`, depending on the solution.

The first activity has each MPI rank be assigned uniformly distributed unsorted data, where the data is assumed to be distributed on the ranks before any processing begins. Each rank is required to sort a single bucket corresponding to a data range, where buckets are of equal width. The ranks perform a communication phase to scatter their local unsorted data to the other ranks. After all ranks receive their data, they perform the sort operation. In the module, the data stays distributed at each rank to reflect that large datasets may exceed the main memory capacity of a single node.

The second activity is identical to the first activity, except the data is exponentially distributed. This means that some ranks will have more data to scatter and sort than others. From this activity, the students learn about data-dependent workloads and resulting load imbalance.

To remedy the load imbalance problem, the third activity requires using a histogram-based approach. In this approach, one rank generates a histogram of its local data and then divides the data into unequal-sized buckets (in contrast to the equal-sized buckets in the first and second activities). This remedies the load imbalance that occurs when using equal-sized buckets, and overall performance is similar to that in the first activity.

In summary, this module teaches students about data-dependent workloads and load imbalance. Students learn that the sequential program does not require scattering the data, and that parallelism incurs non-negligible communication overhead. Furthermore, because sorting is memory-bound, the scalability of the algorithm is not as high as the compute-

bound algorithm presented in Module 2.

E. Module 4: Range Queries

Application motivation: Range queries are used in database systems and in scientific applications [36]. An example 2-dimensional range query is as follows: "Return all asteroids with a light curve amplitude between 0.2–1.0 and a rotation period between 30–100 hours." Range queries are often used as a subroutine in data analysis workflows that require comparing the similarity of feature vectors.

Range queries search a space defined by a minimum bounding box which find all points/feature vectors within that region. This module is not focused on exposure to new MPI primitives, and requires the use of `MPI_Reduce`.

In the first activity, the module assumes that the input dataset and query dataset (the ranges that are searched) are stored on each rank before any processing begins. Each rank searches the input dataset using its assigned set of queries. The module requires implementing a brute-force implementation that does not use spatial indexing methods (e.g., kd-tree [37], R-tree [38], or quad-tree [39]) to prune the search. The module requires examining the strong scaling behavior of the algorithm using fixed input and query dataset sizes.

In the second activity, the module supplies an R-tree [38] data structure that is used to index the input dataset before querying begins. This data structure limits the search distance of the queries such that fewer distance calculations need to be computed, which improves the overall efficiency of the algorithm compared to the brute-force approach.

Comparing the first and second activities, the students learn that the brute-force algorithm has better scalability because it is inherently compute-bound. When employing the R-tree, scalability decreases compared to the brute force approach, as the algorithm has a higher ratio of memory accesses to distance calculations. Despite worse scalability, the R-tree implementation is much more efficient than the brute-force algorithm. Consequently, students learn that more efficient/sophisticated algorithms often have worse scalability than their simple (inefficient) counterparts.

The third activity requires experimenting with running the R-tree implementation using different node counts, and having students devise their own experiment. Their invented experiment may include using dedicated or shared compute nodes, varying the number of nodes, and changing the allocation of tasks to nodes. Ideally, students will learn that deploying p ranks on 1 node performs worse than deploying p ranks on 2 nodes, and that using multiple nodes exploits more aggregate *memory bandwidth* to improve performance. Students learn that memory bandwidth is an important resource.

F. Module 5: k-means Clustering

Application motivation: The k-means clustering algorithm [28] is probably the most popular clustering algorithm given its simplicity and characteristics.

The naïve k-means algorithm [40] is an unsupervised machine learning technique that takes as input a dataset of points and a number of clusters, k, and as output assigns each point to one of k clusters. The algorithm is iterative in nature, and terminates when two centroids (cluster centers) do not change positions between two consecutive iterations. At each iteration, each point is assigned to its closest centroid by computing the distance between the point and all k centroids. Each centroid is then assigned a set of points, and its position is updated to reflect the mean positions of these points.

k-means is an interesting distributed-memory algorithm as it has alternating phases of synchronous computation and communication. Furthermore, depending on k and the data dimensionality, the algorithm may be bound by either computation or communication. The module prescribes a single 2-dimensional input dataset, and requires each rank to be assigned a subset of the data (e.g., each rank is assigned N/p points where N is the number of points and p is the number of processes/ranks). For a given iteration, each rank can independently compute the distances between its subset of the data points and the k centroids; however, the centroid locations need to be updated using global knowledge. In principle, this requires each rank to know the assignment of all points to their respective centroids.

To communicate this global information, the module presents two options that can be used. The first option explicitly describes the assignment of points to centroids, and the second option exploits weighted means. While the first option is more explicit, it requires significant communication overhead. The second (less explicit) option is more efficient and requires minimal communication.

The module requires students to understand how performance varies as a function of k by examining the time spent computing distances between points and centroids, and the time required for communication. Students learn that when k is large, the total time is dominated by computation, whereas on low values of k, the total time is dominated by communication. Also, students will discover that using multiple compute nodes is not advantageous when k is low.

G. Ancillary Modules

We also provide two ancillary modules. One module provides an introduction to the SLURM batch scheduler. The other module provides warmup exercises that gently introduce students to MPI primitives. These exercises can be used as in-class activities. For instance, during class, students can solve these exercises in groups, or the entire class can program the solution with the instructor.

IV. Evaluation

A. Methodology

The first author of this paper teaches a graduate level high performance computing course at Northern Arizona University. We evaluated the efficacy of the modules using pre and post module completion quizzes. The quizzes did not contribute to student grades. The students in the course were motivated to take the quizzes so that they could assess whether they understood the course material. The students

TABLE III
DEMOGRAPHICS OF THE STUDENTS IN THE GRADUATE LEVEL HIGH PERFORMANCE COMPUTING COURSE (CS- COMPUTER SCIENCE, EE- ELECTRICAL ENGINEERING).

Program	Number
Computer Science (BS)	1
Computer Science (MS)	1
Electrical Engineering (MS)	2
Astronomy & Planetary Science (PhD)	1
Informatics & Computing (PhD)	5 (1×bioinformatics, 1×CS, 1×ecoinformatics, 2×EE)

were informed that the results of the quizzes would appear in a study, and the instructor shared the results with the students at the end of the semester. Furthermore, the instructor collected responses to an anonymous free response survey.

We used no-stakes quizzes to understand module efficacy, because without pre-module completion quizzes, we would not have a baseline for each student. We elected not to directly use assignment scores in the course, because it does not capture baseline performance. Furthermore, since students are not expected to master knowledge before completing the modules, the pre quizzes are unfair by design; therefore, it would be unfair to grade these quizzes.

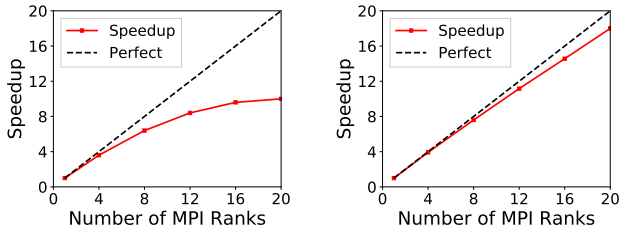
Some quiz scores were excluded from the study. Students that did not complete both pre and post quizzes for a given module had their quiz score removed for that module (if they completed a single quiz). Otherwise, this would not indicate whether their scores improved after completing a given module. Seven of ten students completed all quizzes, so the quiz completion rate is high.

Table III presents the population of 10 students in the high performance computing course offered in Spring 2020. Students in the Informatics & Computing PhD program were asked to report on their prior education, since the multi-disciplinary program trains students with a large range of backgrounds. Note that only three students have a traditional computer science background (one undergraduate, one MS, and one PhD). The other seven students are from electrical engineering, astronomy, bioinformatics, and ecoinformatics[1]. This diverse group of students demonstrates the motivation for these data-intensive pedagogic modules. Without motivating applications, the students may not understand why high performance computing may be important to their research. Furthermore, many students will be able to use the applications in their research.

B. Example Quiz Question

In what follows, we illustrate an example quiz question from Quiz 4 that was used to assess Module 4. **Question:** Figure 1 shows the speedup of two different MPI programs executed on two identical 32-core compute nodes. Both programs only use 20 of 32 cores. Also, both programs need to be executed

[1]A caveat here is that students without a computer science degree may have completed computer science courses during their education, particularly lower division programming courses. Therefore, the information in Table III does not fully capture the educational experiences of the cohort.



(a) Program 1/Compute Node 1 (b) Program 2/Compute Node 2

Fig. 1. Example quiz question. Speedup vs. number of cores/MPI ranks for (a) Program 1; and (b) Program 2.

continuously (on 20 cores) for the next week and will be run on the same two nodes (compute nodes 1 and 2). Another user wants to use one of the compute nodes that you are executing one of your programs on. Select the program and compute node that is most likely to minimize performance degradation to your program. Circle one answer: (1) Program 1/Compute Node 1, or (2) Program 2/Compute Node 2.

In Module 4, the students had to reflect on whether their range query implementation using an R-tree would perform best if they had to share resources with another user that was running either a memory-bound or a compute-bound program. Since CPU cores cannot be shared by users on our cluster (a typical cluster configuration), it is preferable to share resources with the user running a compute-bound program, since the R-tree implementation is memory-bound.

Similarly, the answer to the quiz question is program 2/compute node 2. If poor scalability (Program 1, Figure 1(a)) is due to a memory-bound algorithm, then sharing resources with another job that is memory-bound will likely degrade the performance of both jobs.

This module and quiz question gives students exposure to thinking about the co-scheduling of jobs. Assigning two (or more) identical jobs has been coined the *terrible twins* problem [41]. Co-scheduling the same (or similar) programs on the same resource leads to potentially severe performance degradation relative to executing the jobs on dedicated resources [42], [43]. Since CPU cores continue to be added to multi-core processors, contention for memory bandwidth is often the limiting resource [44].

C. Results

Figure 2 shows the pre and post module completion quiz scores for the ten students in the course. As described above, those students that did not complete both pre and post quizzes for a given module were removed from the sample. From Quiz 1 (Module 1), we observe that overall, the students understood the material very well even before completing the module. Since the students had exposure to MPI communication concepts in lecture, and because the basic communication concepts are less demanding in Quiz 1/Module 1 than future concepts, it was expected that the students were going to perform well on both the pre and post module completion quizzes. From the plots showing Quizzes 2–4, we observe that

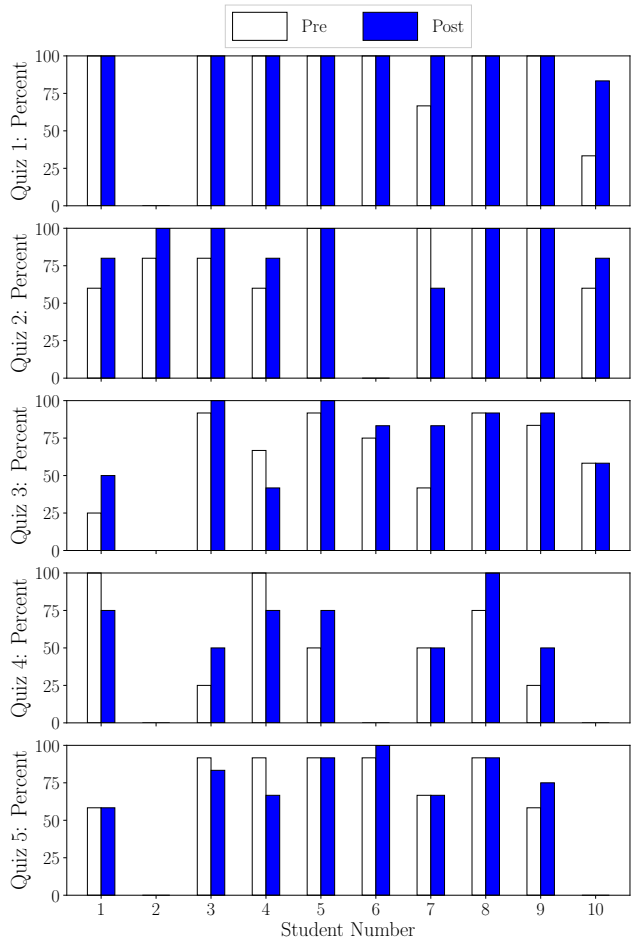


Fig. 2. Student quiz scores pre (white bar) and post (blue bar) module completion. Quizzes 1–5 are ordered from top to bottom, which correspond to modules 1–5, respectively.

there is much more variability in pre and post quiz scores. Also note that six students (#2, 5, 6, 8, 9, 10) had all of their quiz scores either stay the same or increase between the pre and post quizzes. Four students (#1, 3, 4, 7) had at least one score decrease between the pre and post quizzes.

Table IV summarizes several statistics derived from Figure 2. In total, there are 42 pre and post quiz pairs. There are 17 pairs which are equal in score, 19 pairs where scores increased after completing the module, and there are 6 instances where scores decreased after completing the module. Students either maintained the same quiz score or increased their score after completing the module in 85.7% of the instances. Furthermore, we note that in 4 of 5 quizzes, the mean quiz grades increased. Quiz 5 saw a modest decrease in mean quiz scores, from 80.21% to 79.17%. The reason for this minor decrease is not readily apparent from the collected data, so we do not speculate on this phenomena.

To better understand the magnitude of increasing and decreasing scores, we examine the mean relative performance increase (or decrease). The mean relative performance increase and decrease are computed as $\frac{1}{i} \cdot \sum_{j=1}^{i} \frac{|a_j - b_j|}{b_j}$, and

TABLE IV
STATISTICS DERIVED FROM FIGURE 2. SEE TEXT FOR DETAILS.

Statistic	Value
Total Pre & Post Quiz Pairs	42
Pre & Post: Equal in Score	17
Pre & Post: Increase in Score (i)	19
Pre & Post: Decrease in Score (d)	6
Mean Relative Performance Increase	47.86%
Mean Relative Performance Decrease	27.30%
Mean Quiz 1 Grade Pre (Post)	88.89% (98.15%)
Mean Quiz 2 Grade Pre (Post)	82.22% (88.89%)
Mean Quiz 3 Grade Pre (Post)	69.50% (77.78%)
Mean Quiz 4 Grade Pre (Post)	60.71% (67.86%)
Mean Quiz 5 Grade Pre (Post)	80.21% (79.17%)

$\frac{1}{d} \cdot \sum_{j=1}^{d} \frac{|a_j - b_j|}{b_j}$, respectively, where a_j and b_j refer to pre and post quiz scores for those quiz pairs with an increase ($i = 19$) or decrease ($d = 6$) in score. We find that when scores increase, the mean relative increase is 47.86%, but decreasing scores are relatively smaller, which decrease by 27.30% on average. This indicates that in cases where students improved between the two quizzes, the completion of the modules improves understanding of core concepts. We note that due to the small sample size, we abstain from drawing strong conclusions regarding these findings.

From Table IV, we observe that Quiz 4 had a lower score than the other quizzes. This quiz largely focused on analyzing several hypothetical performance evaluations that require students to synthesize their understanding of core performance metrics. Module 4 required students to create their own performance evaluation that uses different resource allocations (number of nodes and cores, dedicated vs. non-dedicated nodes, etc.). Since these module questions are open-ended, depending on how much effort students applied to these module questions, they may have not fully appreciated the hypothetical scenarios in the quiz questions. Therefore, we hypothesize that these scenarios may be more challenging for students than the other quizzes, as they are more conceptual in nature, and were dependent on the level of effort made on examining the impact of resource allocation in Module 4.

D. Student Free Response Surveys

The instructor asked students to comment on several aspects of the course. We present a selection of the responses to the anonymous free response surveys below (edited for spelling, grammar, and brevity).

- *Students were asked if they found the course easier or more difficult than other graduate level courses.* Reported results: 1 student- easier, 5 students- more difficult, and 4 students- much more difficult.
- *Students were asked what aspects of this class they found most challenging.* Students reported: (i) "Building a coding environment on my laptop and dealing with how the cluster works took more effort than I thought." (ii) "... designing a parallel algorithm and working with the cluster were challenging." (iii) "I was a bit overwhelmed in the beginning with trying new code and dealing with the cluster."

- *Students were asked what their favorite module was and why (if applicable).* Four students reported that they liked Module 5 (k-means). Students elaborated that they liked the module because the prior modules provided the scaffolding to do well on the assignment. They also liked the visualization aspect of the assignment, and that it was satisfying to see the data cluster correctly.
- *Students were asked what their least favorite module was and why (if applicable).* The responses were inconsistent: 2, 1, 1, 2, and 1 students found that modules 1, 2, 3, 4, and 5 were their least favorite, respectively.
- *Students were asked what the most challenging module was and why (if applicable).* Four students reported that Module 2 was the most difficult. One student reported that it was challenging because it was significantly more complicated than Module 1. Another student reported that they were still not confident in MPI when completing Module 2. Another student reported that they wanted more instruction on how to write blocking for loops.
- Additional responses from the free response surveys included: (i) "It was a great course, which taught me a new skill." (ii) "Of my classes this seemed like the most practical. . . . And learning how to use Monsoon will help me in other courses. HPC will only grow in importance." (iii) ". . . Furthermore, it is really good to be able to apply parallel programming approaches to speedup an algorithm. . . This knowledge will really help us in our academic life." (iv) "I like that all of the examples span a broad number of subjects and topics."

Interpretation and Discussion: Overall, the students felt that the course was more difficult than their other graduate level courses. Students reported that there are many new concepts and skills to learn related to both PDC and extraneous concepts such as the cluster and the command line environment which make the pedagogic activities more challenging than material in other courses. Few students in the class have a traditional computer science background (Table III), and the course may be more demanding for non-traditional computer science students.

Despite the reported challenges and the difficulty level relative to other graduate level courses, it is clear that core PDC concepts can be taught to students with diverse backgrounds. The pedagogic modules proposed here were largely successful in teaching student learning outcomes that are common in PDC curricula (Table I). We found that students were learning the material and it is clear that the hands-on experience is very beneficial to learning. Furthermore, the results imply that data-intensive applications may be a good vehicle to teach PDC.

There are several caveats to this study. First, we only examined a group of 10 graduate students, which is a small sample to draw broad conclusions regarding the efficacy of teaching PDC using the data-intensive pedagogic modules. Second, students were assigned the pre quizzes before the assignment was due, but not before they were exposed to concepts in lecture. Therefore, the improvement in quiz scores may also be due to lecture exposure and not entirely due to the modules. Third, while the population of students in the cohort is diverse, the students may have completed computer science courses while enrolled in other programs. Therefore, the students may have a stronger background in computer science than that indicated by their degree program.

V. Conclusion

In this paper, we have presented pedagogic modules that teach core PDC concepts through data-intensive algorithms. The modules are designed to address students with diverse educational backgrounds and goals. We evaluated the modules using a population of graduate students in the first author's HPC course. We found that the hands-on approach employing examples from data-intensive application areas was successful at helping students learn key PDC concepts. We hope that the experience enables students to achieve research goals in their academic careers.

We encourage instructors to incorporate these modules into PDC courses. The first author of this paper would be delighted to assist any interested individuals.

Undergraduate Education: While the pedagogic modules were evaluated in a graduate level class, the materials may also be appropriate for teaching undergraduate computer science students. These students would need to learn about the cluster environment. Despite this potential drawback, one benefit of teaching undergraduate computer science students is that all students are expected to have mastered the extraneous concepts that graduate students found challenging (reported in Section IV-D), such as the command line environment, Linux, and programming. Students learn these concepts in the lower division and are expected to be proficient in these concepts by the time they start enrolling in upper division courses. Therefore, to some degree, there will be fewer stumbling blocks when teaching undergraduates than a diverse population of graduate students.

Future Work: Future work includes developing additional modules as follows: (i) Modules that capture excluded concepts, such as increasing focus on communication and latency hiding. (ii) Modules with other data-intensive algorithms so students have some choice in their assignments. (iii) Modules that target undergraduate students.

Additionally, we are working on incorporating feedback from our student surveys into the modules to improve their presentation and detail. Finally, we will perform evaluations of the modules using several cohorts of students in the course. This will allow us to draw stronger conclusions regarding module efficacy, and yield new insights into teaching PDC.

Acknowledgment

We thank our cluster administrator, Chris Coffey, and the HPC team at NAU for their support.

References

[1] J. Ortiz-Ubarri, R. Arce-Nazario, and E. Orozco, "Modules to teach parallel and distributed computing using MPI for Python and Disco," in *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2016, pp. 958–962.

[2] D. N. Parikh, J. Huang, M. E. Myers, and R. A. van de Geijn, "Learning from optimizing matrix-matrix multiplication," in *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2018, pp. 332–339.

[3] H. Casanova, A. Legrand, M. Quinson, and F. Suter, "SMPI Courseware: Teaching Distributed-Memory Computing with MPI in Simulation," in *2018 IEEE/ACM Workshop on Education for High-Performance Computing*, 2018, pp. 21–30.

[4] 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*, 2019, pp. 1–9.

[5] P. T. Bauman, V. Chandola, A. Patra, and M. Jones, "Development of a Computational and Data-Enabled Science and Engineering Ph.D. Program," in *2014 Workshop on Education for High Performance Computing*, 2014, pp. 21–26.

[6] P. M. Leidig and L. Cassel, "ACM Taskforce Efforts on Computing Competencies for Undergraduate Data Science Curricula," in *Proc. of the ACM Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 519–520.

[7] S. K. Prasad *et al.*, "NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates, Version II-beta," http://tcpp.cs.gsu.edu/curriculum/, 2020, accessed: January 3, 2020.

[8] G. Zarza, D. Lugones, D. Franco, and E. Luque, "An innovative teaching strategy to understand high-performance systems through performance evaluation," *Procedia Computer Science*, vol. 9, pp. 1733–1742, 2012.

[9] E. Luque, R. Suppi, and J. Sorribes, "A quantitative approach for teaching parallel computing," in *Proc. of the 23rd SIGCSE Technical Symposium on Computer Science Education*, 1992, pp. 286–298.

[10] A. N. Pears, "Using the DiST Simulator to teach parallel computing concepts," in *Proc. of the 1st International Forum on Parallel Computing Curricula*, 1995.

[11] J. Hartman and D. Sanders, "Teaching parallel processing using free resources," in *Technology-Based Re-Engineering Engineering Education Proc. of Frontiers in Education FIE'96 26th Annual Conference*, vol. 3, 1996, pp. 1483–1486.

[12] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson *et al.*, "XSEDE: accelerating scientific discovery," *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, 2014.

[13] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.

[14] O. Yasar and R. H. Landau, "Elements of computational science and engineering education," *SIAM review*, vol. 45, no. 4, pp. 787–805, 2003.

[15] A. Fitz Gibbon, D. A. Joiner, H. Neeman, C. Peck, and S. Thompson, "Teaching high performance computing to undergraduate faculty and undergraduate students," in *Proc. of the 2010 TeraGrid Conference*, 2010, pp. 1–7.

[16] V. Holmes and I. Kureshi, "Developing high performance computing resources for teaching cluster and grid computing courses," *Procedia Computer Science*, vol. 51, pp. 1714–1723, 2015.

[17] D. Toth, "A portable cluster for each student," in *IEEE International Parallel & Distributed Processing Symposium Workshops*, 2014, pp. 1130–1134.

[18] L. Alvarez, E. Ayguade, and F. Mantovani, "Teaching HPC systems and parallel programming with small-scale clusters," in *IEEE/ACM Workshop on Education for High-Performance Computing*, 2018, pp. 1–10.

[19] R. Brown, J. Adams, S. Matthews, and E. Shoop, "Teaching Parallel and Distributed Computing with MPI on Raspberry Pi Clusters," in *Proc. of the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 1054–1054.

[20] J. Eckroth, "Teaching future big data analysts: Curriculum and experience report," in *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2017, pp. 346–351.

[21] D. Deb, S. Cousins, and M. Fuad, "Teaching Big Data and Cloud Computing: A Modular Approach," in *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2018, pp. 377–383.

[22] L. A. Wilson and S. C. Dey, "Computational Science Education Focused on Future Domain Scientists," in *2016 Workshop on Education for High-Performance Computing*, 2016, pp. 19–24.

[23] "Data-Intensive High Performance Computing Pedagogic Modules," https://jan.ucc.nau.edu/mg2745/pedagogic_modules/courses/hpcdataintensive/, accessed: December 31, 2020.

[24] B. S. Bloom *et al.*, "Taxonomy of educational objectives. Vol. 1: Cognitive domain," *New York: McKay*, vol. 20, p. 24, 1956.

[25] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, no. 34, 1996, pp. 226–231.

[26] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proc. of the ACM SIGMOD International Conference on Management of Data*, 1995, pp. 71–79.

[27] M. Gowanlock and B. Karsin, "Accelerating the similarity self-join using the GPU," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 107 – 123, 2019.

[28] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[29] R. Sedgewick and K. Wayne, *Algorithms*. Addison-Wesley Professional, 2011.

[30] G. Graefe, "Implementing sorting in database systems," *ACM Computing Surveys (CSUR)*, vol. 38, no. 3, 2006.

[31] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys (CSUR)*, vol. 40, no. 4, pp. 1–58, 2008.

[32] B. Sesar, J. S. Stuart, Ž. Ivezić, D. P. Morgan, A. C. Becker, and P. Woźniak, "Exploring the Variable Sky with LINEAR. I. Photometric Recalibration with the Sloan Digital Sky Survey," *The Astronomical Journal*, vol. 142, no. 6, p. 190, 2011.

[33] J. Zhou, X. Hong, and P. Jin, "Information Fusion for Multi-Source Material Data: Progress and Challenges," *Applied Sciences*, vol. 9, no. 17, p. 3473, 2019.

[34] R. Baecker, "Sorting out sorting: A case study of software visualization for teaching computer science," *Software visualization: Programming as a multimedia experience*, vol. 1, pp. 369–381, 1998.

[35] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[36] J. Kim and B. Nam, "Co-processing heterogeneous parallel index for multi-dimensional datasets," *Journal of Parallel and Distributed Computing*, vol. 113, pp. 195–203, 2018.

[37] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[38] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," in *Proc. of the ACM SIGMOD International Conference on Management of Data*, 1984, pp. 47–57.

[39] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[40] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[41] A. d. Blanche and T. Lundqvist, "Terrible Twins: A Simple Scheme to Avoid Bad Co-schedules," in *Proc. of the 1st COSH Workshop on Co-Scheduling of HPC Applications*, 2016, p. 25.

[42] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," *ACM Sigplan Notices*, vol. 45, no. 3, pp. 129–142, 2010.

[43] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *38th Annual International Symposium on Computer Architecture*, 2011, pp. 283–294.

[44] D. Xu, C. Wu, and P.-C. Yew, "On mitigating memory bandwidth contention through bandwidth-aware scheduling," in *Proc. of the 19th International Conference on Parallel Architectures and Compilation Techniques*, 2010, pp. 237–248.