Enrich a data structures course with parallelism

Qiong Cheng

University of North Carolina at Charlotte

qcheng1@uncc.edu

Abstract— **This Research to Practice Work in Progress paper develops an adaptive learning module aimed at enriching data structures and algorithms courses (CS2/DS) with introductory parallel computing. We emphasize exploitable shared-memory parallelism with the intention of teaching students to decompose a problem or its underlying big data structure into parts that can execute effectively in many- or multi-core processes. The module can set up conditional mastery paths and differentiate assignments for individual students automatically. The adaptive learning reflects the student-directed learning by doing and so fits naturally into the conventional CS2/DS courses. We conducted a 50-minute learning session in an adaptive learning CS2/DS class for a group of 56 students in the Fall 2019 semester. The experimental results, from two surveys executed before and after the session, was successful in engaging students, instigating students' interest in parallel computing, and improving the students' awareness and appreciation. The module can be extended to support other parallel computing concepts, such as critical section, race condition, and multi-thread synchronization and cooperation techniques. As a side benefit, we would like to show that adaptive learning prepares students, gets students to engage, and enhances their performance.**

Keywords— adaptive learning, personalized learning, parallel computing, parallelism, data/task decomposition, data structures and algorithms

I. INTRODUCTION

More and more computing systems are evolving in parallel. Simultaneously the proliferation of big data across most sectors necessitates the training of students to efficiently use modern parallel computing systems in attaining higher performance [1]. These systems possess homogenous or heterogeneous many- or multi-core processes or are interconnected within a network, forcing fundamental changes in the design of software and in turn challenging the speed of curriculum evolution [25-27].

In the 2013 ACM/IEEE-CS curriculum guidelines for undergraduate CS degrees, parallel computing topics have been explicitly included as crucial [29]. CDER center for parallel computing education proposed a detailed curriculum to integrate parallel computing in the early stages of undergraduate curriculum [1].

Currently, while many institutions provide a senior-level undergraduate or graduate parallel computing course, these are located at the late stages of the curriculum [2]. As we and others have observed, the student's transition from sequential to parallel thinking in a later stage appears to be difficult [22]. In response, our institution has offered an upper-level parallel computing course. Its emphasis is on parallel algorithmic design and implementation through different parallel programming models and patterns in C, such as OpenMP, PThreads, MPI, and CUDA. Students who enter the upper-level parallel programming course are not used to the critical thinking of parallel computing. It is challenging for students to learn the abstract concepts of parallel computing and at the same time to learn different programming models.

Yousun Ko etc. designed an introductory one-semester parallel programming course for 1st and 2nd-year students who had experience with C programming but no prior knowledge of computer architectures, operating systems or algorithms [22]. The innovation added an entire required course early on in the curriculum, where students faced the same challenge and struggled even more with the sequential and parallel algorithm design and problem solving [24].

It is suggested that students would be better served to be exposed into introductory parallelism before an advanced course in parallel computing [1, 24-25]. A group of computer science educators have recommended the "sprinkling" of fundamental parallel computing throughout courses as a complementary medium in the curriculum, without significantly reducing traditional content [14, 17-22]. Parallel computing is explicitly introduced to Operating Systems and Networking [13] and Computer Architectures [22]. Steven Bogaerts etc. created materials for CS1 in Python [15-16], which cover a high-level overview of parallel computing concepts and technologies and introduce basic use of the Python multiprocessing package. Kim B. Bruce etc. introduced concurrency to CS1 in the context of event-driven programming [23]. Dan Grossman and Ruth E. Anderson claimed that CS2/DS (Data Structures) was a natural place to introduce parallelism and integrated a three-week introduction of multithreading with data structures [24]. Daniel J. Ernst and Daniel E. Stevenson designed a parallel program design process, introducing concurrency to CS1, reinforcing concurrency in CS2, and delivering concurrency limitations in the algorithms course [28]. The CS in Parallel (CSinParallel) provide a platform to collect and share parallel computing modular teaching materials [3].

All these above-mentioned state of art work inspired our study. We agree to train students earlier with parallel thinking and programming. However, learning modules resulted from these curriculum innovation are still in a "one size fits all" model. This unfortunately intensify the explosion of learner diversity. The need for pedagogical innovation is all the more urgent.

With the aim of engaging students to actively participate in the learning process, the CS2/DS has deployed student-focused techniques such as algorithmic visualization, simulation or animation, group study, problem-based learning, and project-based learning. Recently, "flipped classroom" or blended learning pedagogy has become popular, which usually includes the "one size fits all" preparation before class and peer instructions in class. To effectively flip a classroom, requires both students' self-discipline and self-paced learning before classes [12]. However contemporary students have been socialized via a smart phone. They are prone to distraction and many lack purposeful motivation. The result is a serious challenge to the regular "one size fits all" model in "flipped classroom" preparation.

Our considered response to these above-mentioned emerging requirements and challenges is to investigate the parallel computing curriculum innovation in an adaptive learning environment, where student's learning path can be customized automatically. Since performance tuning in modern computing devices relies not only on program paralleling but also on data structure parallelization, in turn, the corresponding course curriculum in CS2/DS (Data Structures) needs to reflect this.

In the paper, we propose an innovative adaptive learning parallelism module. Our modular materials and mini-projects are designed to expose our CS2/DS students to parallelism. We implemented the design in a 50-minute class session of 56 students. Our research hypotheses in the paper are as follows:

Hypothesis 1: Introducing parallel computing to Data Structures course increases students' awareness and appreciation of parallelism in modern devices.

Hypothesis 2: Adaptive learning has the potential to assist in the preparation, engagement and enhanced performance of students.

The rest of the paper describes our methodology and introductory parallel computing module (section 3) after a primer on our academic context (Section 2). We then evaluate our exploration in section 4. Section 5 discusses related comments and issues that require further investigation.

II. ACADEMIC CONTEXT

A typical CS2/DS (Data Structures) is characterized in a single-thread context. Our data structures and algorithms course falls into this category and with JAVA as the programming language. It covers ten modules, such as generics and ADTs, algorithmic analysis, queues, stacks, lists, search and sorting, recursions, trees, hash tables/maps, and graphs. It plays a critical role in teaching students the design and programming of sequential data structures, which is difficult because of the unpredictability when being deployed in concurrent environments with multiple processors. Moreover it is impossible to fully leverage modern computing devices by using sequential data structures. Similarly converting or adapting sequential data structures to concurrent data structures becomes arduous. Students entering the upper-level parallel computing course without the knowledge of concurrent data structures invariably struggle [2].

Additionally, there is a wide variance in the incoming students' prior knowledge, capabilities to learn programming, and undertaken critical thinking and problem-solving. More than half of class body (58.2%) were transfer students; about half of students were working students. Around 83.3% of the students were male. Most were exposed to different programming languages at different levels. Some instructors in CS 1 introduced threads to students but most did not, which brought forth a growing disparity on programming experience among students entering our data structures and algorithms courses. The increasing disparities was likely the reason for the observable tendency of the less experienced to be overwhelmed and the experienced to be bored. Another difficulty lies in finding time within already packed lectures to cover additional materials.

The Data Structures and Algorithm course is a gateway core course taken by all CS majors and minors as well as a prerequisite to most of our senior-level classes. Accordingly we set up our central pedagogic theme in it as a combination of diverse constructivist teaching techniques and have used adaptive learning for two semesters. The impact of adaptive learning from the first semester can be found in [9].

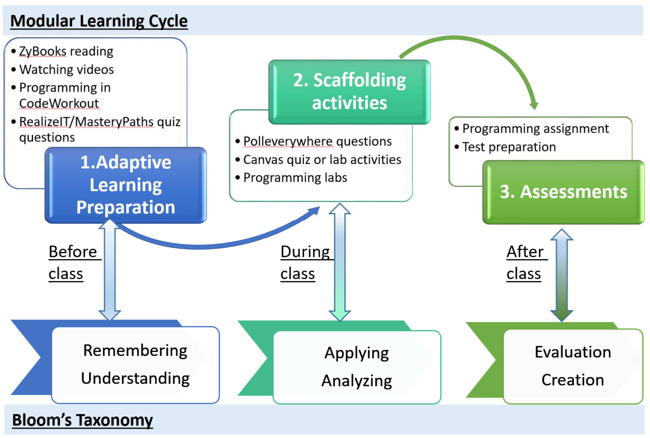

Fig. 1 Modular learning cycle fits with the revised Bloom's Taxonomy. Before classes, students finish the lower level of cognitive work by reading book chapter(s), lecture notes, watching videos, and working on quiz question(s) or short programming questions. During class, students engage in higher cognitive levels of learning with peers and instructors through scaffolding activities, filling knowledge gaps with the guidance of instructor(s) and carraying out programming practices. Thirdly, students work on programming assignments or prepare for tests.

III. METHODOLOGY

A. *Diverse Constructivist Teaching in a Data Structures and Algorithms Course*

The core notion of our teaching pedagogy is that students learn to construct and build up their knowledge map, with new knowledge built upon the foundation of previous learning, in which instructors function as a helper or facilitator of learning by deploying diverse technology or teaching techniques. Such a pedagogy is grounded in three educational, psychological theories, such as instructivism, constructivism, and connectivism. Our ultimate goal is to work towards personalized student-centered learning.

We employed the diverse constructivist teaching strategy in our Data Structures and Algorithms course for two regular semesters. Each had three 50-minute lectures per week. It includes students-focused active learning techniques, such as algorithmic visualization, simulation or animation, collaborative study, problem-based learning, mini-project-based learning, and "flipped classroom" [5-8]. To take account of students' diversity in prior-knowledge and programming experience, we used an adaptive learning preparation platform, named Realizeit [4] for each regular module so that students could prepare and explain what they knew and did not know before the class. Additionally, we adopted modern technologies such as 1) Remind App for daily reminders and 2) Piazza for a class community-based free discussion.

All modules follow a certain learning cycle, which fits with the revised Bloom's taxonomy (see Fig. 1):

1. Instructors motivate students by showing the relevance and importance of preparation via an adaptive learning platform

(Realizeit or MasteryPaths in canvas). The platform judges student's knowledge levels, automatically customizes learning path, and differentiates preparation assignments for students. Thus, the workload in preparation may vary from a student to another. Note that this is different from the traditional "one-size-fits-all" preparation design.

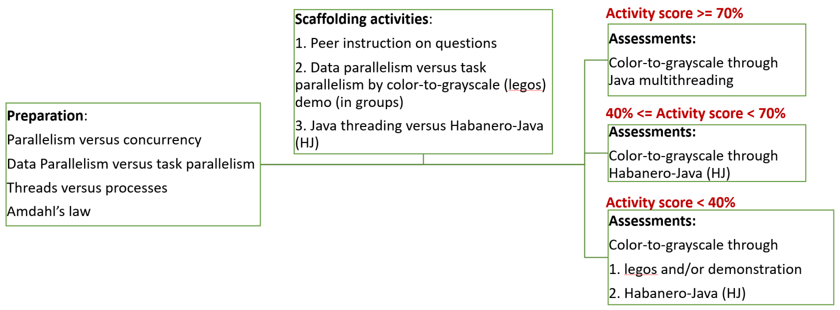
Preparation:
Parallelism versus concurrency
Data Parallelism versus task parallelism
Threads versus processes
Amdahl's law
Scaffolding activities:
1. Peer instruction on questions
2. Data parallelism versus task parallelism by color-to-grayscale (legos) demo (in groups)
3. Java threading versus Habanero-Java (HJ)
Activity score >= 70%
Assessments:
Color-to-grayscale through Java multithreading
40% <= Activity score < 70%
Assessments:
Color-to-grayscale through Habanero-Java (HJ)
Activity score < 40%
Assessments:
Color-to-grayscale through
1. legos and/or demonstration
2. Habanero-Java (HJ)

Fig. 2 Parallelism learning module where an assessment assignment is automatically customized according to student scores.

2. In classes, we encourage teamwork. Instructors scaffold instructions, design lab activities, review main concepts, resolve confusion, and organize hands-on programming activities. Ideally in solving a problem students should collaboratively apply learning concepts, compare and analyze related concepts, and construct necessary data structures.

3. After classes, students are required to independently work on programming assignment or prepare/take a test to assess their understanding from low-order to high-order, according to Bloom's taxonomy.

B. Data and Loop-based Parallelism Module

The essence to parallel computing in any parallel programming environment is exploitable parallelism and concurrency. These exist in a computational problem which can be decomposed into non-overlapping subproblems or in which underlying big data structure can be decomposed into independent parts. As described in [24], parallelism focuses on using many- or multi-core processors to solve a problem faster; concurrency focuses on correctly and efficiently managing access to shared resources through synchronization or cooperation techniques, such as mutexes, semaphores, read-write locks, condition variables, and interleaving.

In the paper, we emphasize introducing exploitable shared-memory parallelism to the CS2/DS course with the goal of enabling students to apply loop-based parallelism or data-based parallelism to solve a problem. Concurrency topics such as race conditions and deadlock are not covered in the current adaptive learning module but can be expanded to the learning module.

In shared-memory programming environments, a data-based parallelism implies a loop-based parallelism. The first step of parallelism is decomposition. We used image color to grayscale conversion as an application example and designed mini projects, since an image consists of a matrix of pixels which can be decomposed in a variety of ways (rows, columns, or blocks of varying shapes). The details of the mini-projects are in next subsection.

Our goal of the module is as follows, given a problem in which central data structure is non-recursive, students enable to identify the task/data parallelism in the problem or underlying data structure, design the algorithm so that the parallelism can be exploited, and then implement the solution using a suitable programming environment. We pose three questions: 1) whether and how the central data structures defining the problem can be broken down into chunks that can be operated on concurrently, 2) how to implement it, and 3) how to evaluate the speedup when using multiple processes.

Considering our students' variance in prior-knowledge, we propose an adaptive learning module and designed its first version (see Fig. 2.), which follows the modular learning cycle specified in Fig. 1. Since the Realizeit platform was shared with other instructors, we originated the exploratory adaptive learning module in canvas through MasteryPaths. We enabled MasteryPaths to automatically assign assessment assignments based on student scores.

In the proposed learning module, we provide different parallel programming environments to be customized for students. Java multithreading is a relatively low-level environment, requiring that the programmer explicitly specify the behavior of each thread. Habanero-Java (HJ), on the other hand, like OpenMP, supports loop-based parallelism and allows programmers to develop at a higher level. Through HJ, one could just specify a block of code that should be executed in parallel, and the compiler and the run-time Java virtual machine will determine how to parallelize the tasks. Since Habanero-Java (HJ) allows programmers to incrementally parallelize existing serial programs, the transition from serial to parallel becomes easier, for those students who had no prior experience in threading. Such students could ignore the syntax of Java threading and mainly focus on the fundamental parallelism concepts, and capture, apply, and evaluate them in a reasonably acceptable way. Students are assigned four basic documents to read before class. During class activities, students get a short quiz test. For those students who have score higher than 70%, the Java multithreading programming environment is designated. But for others, the Habanero-Java (HJ) programming environment [10] is designated. The threshold is arbitrarily determined by the ratio of fundamental parallel programming questions to Java threading questions in the parallelism module.

C. Practice of Data and Loop-based Parallelism in Color to Grayscale Conversion through Habanero-Java (HJ)

In this subsection, we explain the real-life application that we used in our mini-projects, using Habanero-Java (HJ) library. The image color-to-grayscale conversion is a typical data parallelism problem on a two-dimension array. In a serial code, we first read the image into a two-dimension array, secondly convert the two-dimension-array image buffer from color scale to gray scale, and then write the buffer to another image file. The intensive loop-based computation lies in converting its RGB value of each pixel in the image to its grayscale value (See Fig. 3 and Table 1A).

By using Habanero-Java (HJ), we could transfer the serial color-to-grayscale loop in Table 1A to its parallel version in Table 1B. The HJ parallel for, using a lambda expression, is a self-contained parallel loop. Similar to OpenMP parallel for directive, it divides the iterations of the loop into equal parts, forks multiple threads to execute each part, and then joins up and exits when all threads finish.

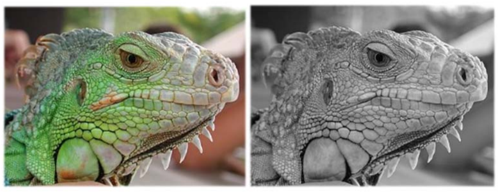

Fig. 3 An example of color to grayscale conversion. Left image is in color scale and right image in gray scale.

A) **Serial for**:

```
for (int y = 0; y < height; y++)
        for (int x = 0; x < width; x++)
                changeColor2gray(image, x, y);
```

B) **Parallel for**:

```
launchHabaneroApp(() -> {
            forall(0, 0, height-1, width-1, (y,x)->{
                    changeColor2gray(image, x, y);
             });
});
```

Table 1. Transfer a for loop from serial version to parallel version using Habanero-Java (HJ). In the table, we assume that all variables, image, image height, and width are already declared and initialized. The function changeColor2gray() is oriented to a single pixel, aiming at getting the RGB value of the pixel, computing the average of RGB i.e., avg = (R+G+B)/3, and replacing the R, G and B value of the pixel with the calculated average.

In the hands-on mini-project, we instructed students to 1) set up environment, 2) explore different configurations, 3) execute and evaluate the codes, 4) repeat experiments for ten times, 5) calculate the average execution time and speedup by Amdahl's Law, and 6) draw a plot to demonstrate the speedup over varying number of threads.

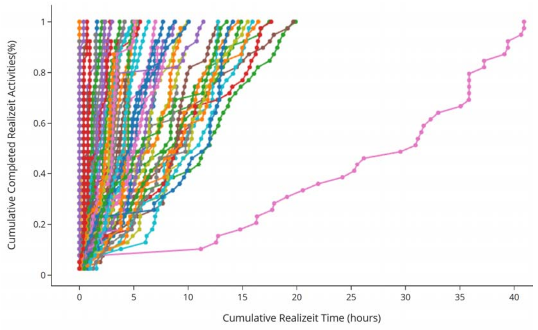

Fig. 4 Cumulative Realizeit time and covered percentage activities over ten modules. The marks in the same line correspond to cumulative time and completed percentage over Realizeit activities for a student.

Final exam	Average score of group A	Average score of group B	P-value of t test between scores of group A and B
Non-programming questions	41.6	46.3	0.012
Programming questions	34.0	42.8	0.019
Total score	76.6	90.1	0.034

Table 2. Statistically significant difference of final exam performance between two groups of students. Group A corresponds to unmotivated students and B motivated students.

IV. Evaluation

The first version of our implementation aimed at improving students' awareness and appreciation of parallel computing as well as exposing students to parallelism-based programming environment. We evaluated our exploration in a 50-minute adaptive-learning CS2/DS class of 56 students in the Fall 2019 semester. Here we reported our evaluation from two perspecitves which tested our hypotheses pleasantly.

A. Increasing students' awareness and appreciation of parallelism

We designed and conducted two types of surveys in measuring students' self-satisfaction, awareness, and appreciation of parallelism on 5-point Likert scales. About 25% more students in post-survey than in pre-survey agreed that 1) learning parallelism-based computing was very helpful for making full use of computing and storage devices and 2) parallelism-based computing would benefit them in career. Our post-survey feedback also indicated that 1) 87% students thought that learning parallelism-based computing were interesting, 2) 75% students thought that their career would involve parallelism, and 3) 61% students would like to recommend parallelism-based computing to friends.

B. Effectiveness of Adaptive Learning

We calculated cumulative preparation time and completed percentage over Realizeit activities of ten modules (see Fig. 4). In the experiment, we focused on two groups of students classified by the preparation work in Realizeit: A) unmotivated students who spent below-average time and scored below average, and B) motivated and diligent students who spent above-average time and scored above average. We extracted their final exam scores respectively on non-programming questions, programming questions, and the total. Student's t test on these scores of group A and B functions as a beacon to highlight the statistically significant difference of score means between two groups. Viewing from final exam performance of these two groups (see Table 2), we found out that group B performed significantly better than group A in final exam, which indicates that adaptive learning could help get students prepared, engaged, and perform well.

V. DISCUSSION

We proposed an exploratory adaptive learning parallelism module and designed materials and mini-projects to expose our CS2/DS students to parallelism and implemented it in a learning cycle of a 50-minute session of 56 students. We conducted two surveys before and after the session, which results turned out to be a success in the sense of increasing students' interest in parallel computing. Also adaptive learning is a means of getting students prepared, engaged, and performing well.

Our take-out is that adaptive learning environment enables the seamless integration of fundamental parallel computing with the gateway core course without sacrificing traditional content. In our future work, we would like to extend the adaptive learning module with concurrency and distributed computing. The study set up a stage for future extension in order to spread throughout introductory parallel and distributed computing early in the CS curriculum as discussed in [1].

ACKNOWLEDGMENT

The manuscript is partially based upon Faculty Teaching Development work supported by the National Science Foundation under Grant No. NSF-1730527 and the National Science FoundationAward 1519160. Thank Dr. Eric Saule and Dr. Yonghong Yan for sharing their expertise in parallel computing. Thank Dr. Celine Latulipe and Manuel Perez Quinones for sharing best practice in CS2/DS course design.

REFERENCES

[1] NSF/IEEE-TCPP Curriculum Working Group. NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing core topics for undergraduates. Technical report, IEEE-TCPP, 2012. http://tcpp.cs.gsu.edu/curriculum/?q=home

[2] Saule, E. (2018, May). Experiences on teaching parallel and distributed computing for undergraduates. In 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 361-368). IEEE.

[3] CSinParallel (Parallel computing in computer science curriculum). https://csinparallel.org

[4] Realizeit. http://Realizeitlearning.com/wp-content/uploads/2018/04/System-Overview-Feb-2018.pdf

http://Realizeitlearning.com/community/

[5] Edgcomb, A., & Vahid, F. (2015, October). How many points should be awarded for interactive textbook reading assignments?. In 2015 IEEE Frontiers in Education Conference (FIE) (pp. 1-4). IEEE.

[6] Edgcomb, A., Vahid, F., & Lysecky, R. (2015, October). Students learn more with less text that covers the same core topics. In 2015 IEEE Frontiers in Education Conference (FIE) (pp. 1-5). IEEE.

[7] Edgcomb, A., Vahid, F., Lysecky, R., Knoesen, A., Amirtharajah, R., & Dorf, M. L. (2015, June). Student performance improvement using interactive textbooks: A three-university cross-semester analysis. In 2015 ASEE Annual Conference and Exposition.

[8] A. Edgcomb and F. Vahid. (2016) Simplifying a Course to Reduce Student Stress so Students Can Focus Again on Learning, Proc. of ASEE Annual Conference, New Orleans, June 2016. https://www.zybooks.com/research

[9] An Award-Winning Course Amps Up Active Learning for Student Success:http://realizeitlearning.com/community/active-learning-computer-science-unc-charlotte

[10] Imam, S., & Sarkar, V. (2014, September). Habanero-Java library: a Java 8 framework for multicore programming. In Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools (pp. 75-86). ACM.

[11] Neeman, H., Lee, L., Mullen, J., & Newman, G. (2006). Analogies for teaching parallel computing to inexperienced programmers. ACM SIGCSE Bulletin, 38(4), 64-67.

[12] Warter-Perez, N., & Dong, J. (2012, April). Flipping the classroom: How to embed inquiry and design projects into a digital engineering lecture. In Proceedings of the 2012 ASEE PSW Section Conference (Vol. 39)

[13] Porter, L., Bailey Lee, C., Simon, B., & Zingaro, D. (2011, August). Peer instruction: do students really learn from peer discussion in computing?. In Proceedings of the seventh international workshop on Computing education research (pp. 45-52).

[14] Porter, L., Bailey Lee, C., & Simon, B. (2013, March). Halving fail rates using peer instruction: a study of four computer science courses. In Proceeding of the 44th ACM technical symposium on Computer science education (pp. 177-182).

[15] Bogaerts, S., & Stough, J. (2015). Parallelism in Python for novices. In Topics in Parallel and Distributed Computing (pp. 25-58). Morgan Kaufmann.

[16] Bogaerts, S., Burke, K., Shelburne, B., & Stahlberg, E. (2010, October). Concurrency and parallelism as a medium for computer science concepts. In Curricula for Concurrency and Parallelism workshop at Systems, Programming, Languages, and Applications: Software for Humanity.

[17] Nevison, C. H. (1995). Parallel computing in the undergraduate curriculum. Computer, 28(12), 51-56.

[18] Meredith, M. J. (1992, March). Introducing parallel computing into the undergraduate computer science curriculum: a progress report. In Proceedings of the twenty-third SIGCSE technical symposium on Computer science education (pp. 187-191).

[19] Maxim, B. R., Bachelis, G., James, D., & Stout, Q. (1990, February). Introducing parallel algorithms in undergraduate computer science courses (tutorial session). In Proceedings of the twenty-first SIGCSE technical symposium on Computer science education (p. 255).

[20] de Freitas, H. C. (2012, October). Introducing parallel programming to traditional undergraduate courses. In 2012 Frontiers in Education Conference Proceedings (pp. 1-6). IEEE.

[21] Ko, Y., Burgstaller, B., & Scholz, B. (2013, March). Parallel from the beginning: The case for multicore programming in thecomputer science undergraduate curriculum. In Proceeding of the 44th ACM technical symposium on Computer science education (pp. 415-420).

[22] Bruce, K. B., Danyluk, A., & Murtagh, T. (2010, March). Introducing concurrency in CS 1. In Proceedings of the 41st ACM technical symposium on Computer science education (pp. 224-228).

[23] Grossman, D., & Anderson, R. E. (2012, February). Introducing parallelism and concurrency in the data structures course. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (pp. 505-510).

[24] Shavit, N. (2011). Data structures in the multicore age. Communications of the ACM, 54(3), 76-84.

[25] Pingali, K., Nguyen, D., Kulkarni, M., Burtscher, M., Hassaan, M. A., Kaleem, R., ... & Prountzos, D. (2011, June). The tao of parallelism in algorithms. In Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation (pp. 12-25).

[26] Rague, B. W. (2012, February). Exploring concurrency using the parallel analysis tool. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (pp. 511-516).

[27] Ernst, D. J., & Stevenson, D. E. (2008, June). Concurrent CS: preparing students for a multicore world. In Proceedings of the 13th annual conference on Innovation and technology in computer science education (pp. 230-234).

[28] Sahami, M., Danyluk, A., Fincher, S., Fisher, K., Grossman, D., Hawthorne, E., ... & Cuadros-Vargas, E. (2013). Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. Association for Computing Machinery (ACM)-IEEE Computer Society.

[29] Demetry, C. (2010, October). Work in progress—An innovation merging "classroom flip" and team-based learning. In 2010 IEEE Frontiers in Education Conference (FIE) (pp. T1E-1). IEEE.