A Spiral Bilingual-Programming Approach for Teaching Concurrency and Parallelism

Dhananjai M. Rao
Computer Science and Software Engineerring (CSE) Department
Miami University
Oxford, OHIO 45056. USA
raodm@miamiOH.edu

***Abstract*—Innovative Practice Full Paper: Concepts of concurrency and parallelism are indispensable aspects of today's technology-centric human societies. However, application of these concepts are often taught using a single programming language whose semiotics, despite multimodal instruction, can obfuscate and hinder thorough understanding of the concepts. Consequently, a spiral bilingual-programming approach in which topics are reemphasized with increasing depth of coverage using two different programming languages is proposed. The proposed approach increases pedagogical modalities and learning opportunities for multifaceted comprehension to enable synthesis of various concepts. This article presents experiences of pursuing a spiral bilingual-programming (namely Java and C++) instruction in an undergraduate course for teaching concepts of concurrency and parallelism. Importantly, it contributes quantitative results (that is conspicuously missing in related literature) from various assessments conducted to evaluate the proposed method. The paper discusses statistics (N=85) collated from direct as well as indirect formative and summative assessments conducted during two offerings of the course to evaluate the effectiveness of the proposed approach. Analysis of the assessment data shows that a spiral, bilingual-programming approach is beneficial in increasing the cognition of concepts related to concurrency and parallelism thereby improving student learning and overall course outcomes.**

***Index Terms*—Spiral learning Bilingual-programming, Java, C++, Parallel and Distributed Computing (PDC), multithreading, multiprocessing**

I. Introduction & Motivation

The demand for increased computing capabilities in diverse form factors, ranging from high density servers to mobile handheld devices, is steadily growing. Coupled with a multitude of design, manufacturing, thermal, and power constraints the growing demand has led to widespread adoption of multicore, multiprocessor architectures. The rapid increase in sophistication and complexity of modern computing devices have necessitated corresponding advancement in the software systems that enable effective utilization of today's highly parallel devices and distributed systems.

A. Curricular significance of Parallel and Distributed Computing (PDC)

The parallel and distributed software systems and hardware platforms are designed, developed, and maintained using a broad range of concepts and skills that encompass the field of Parallel and Distributed Computing (PDC). PDC has rapidly expanded from parallel scientific computing and custom distributed systems to include modern cloud computing, "big data" analytics, and web-services integrated using Service Oriented Architectures (SOA). PDC is ubiquitous and is the turnkey solution for the current and foreseeable future [27]. Designing, developing, and maintaining software for these contemporary systems requires extensive use of PDC concepts and skills. Moreover, demands for PDC skills is rapidly growing not only in computer science but also in a broad range of non-computing disciplines including, medicine, business intelligence, data analytics & data science, and computational sciences involving physics, chemistry, and biology.

TABLE I
Comparison of changes in course hours recommended for several knowledge areas in Tier 1 (core coverage) and Tier 2 (additional coverage) in the CS2013 ACM/IEEE computer science curriculum [1].

Knowledge Area	**CS-2013**		**CS-**	**CS-**
	Tier 1	**Tier 2**	**2008**	**2001**
Algorithms & complexity	19	9	31	31
Parallel distributed computing	5	10	N/A	N/A
Networking & communication	3	7	15	15
Operating systems	4	11	18	18

The pervasive use of PDC has added a new dimension to the sequence of undergraduate courses that are offered in computer science programs in many institutions including the Computer Science and Software Engineering (CSE) department at Miami University [9] (mid-west United States), with an exceptionally strong commitment to teaching. The recent (2013) revision of ACM/IEEE computer science curriculum guidelines [1] explicitly calls for 15 hours of coverage of PDC concepts (5 hours at Tier 1 and 10 hours at Tier 2) as summarized in Table I. Some of the PDC concepts stressed in the 2013 guidelines were conspicuously absent in the previous revisions as indicated by `N/A` in Table I. The guidelines also discuss the growing need for PDC knowledge and the importance of teaching PDC concepts in undergraduate curricula despite the broad range of geographic and cultural contexts of various institutions [1].

B. *Curricular enhancements & ensuing pedagogical challenges*

Envisioning the importance of Parallel and Distributed Computing (PDC), the CSE-381: Operating Systems (OS) course was enhanced include contemporary PDC concepts. The OS course was a good candidate because coverage of core Parallel and Distributed Computing (PDC) concepts typically occurs in junior (3rd year) or senior (4th year) levels in many undergraduate curricula [20], including CSE-381. The 3rd or 4th year coverage stems from the fact that PDC builds upon foundations of computing, data structures, and computer architecture.

Enhancements to the course included the introduction of core topics proposed by the NSF/IEEE-TCPP curriculum initiative on PDC [20] at different levels of cognition as defined by Bloom's taxonomy. The revised curriculum also includes application and analysis of the PDC concepts through extensive programming centered laboratory exercises, homework assignments, and term projects to foster synthesis and evaluation of PDC topics.

However, increased coverage of PDC concepts proved to be a challenge for the student community as illustrated by the chart in Figure 1. The average GPA for the course noticeably degraded and was of concern to both the student body and the department.

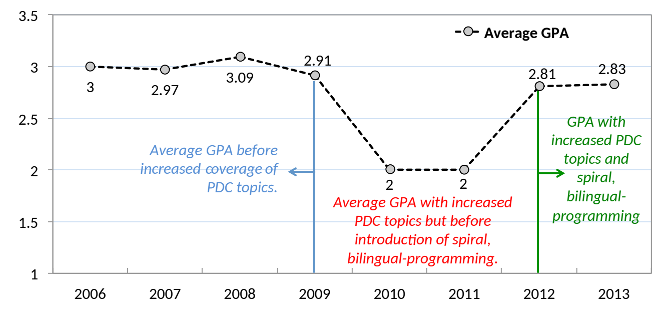


Fig. 1. Changes in average GPA in CSE-381 due to increased coverage of PDC concepts before and after the proposed spiral bilingual-programming approach

II. Pedagogical challenges of teaching PDC

The degradation in overall GPA (illustrated in Figure 1) spurred analysis of the challenges faced in in learning and teaching PDC concepts. Analysis of formative assessments, anonymous surveys, and informal feedback from students revealed the following three categories of issues:

1) **Conceptual challenges of PDC topics**: Most algorithmic and programming courses in modern computer science curricula focus on traditional sequential or serial approaches. Since notions of concurrency and parallelism are paradigmatically different from serial or sequential programming concepts, learning PDC concepts is a challenge [12]. Understanding issues in PDC requires the ability to conceptualize and build a mental model of multiple, concurrent threads of execution with interplay between them. Transitioning from serial thought process to a concurrent mental model is challenging [12].
2) **Challenges due to programming language transitions**: In most institutions, the sequence of prerequisite courses are often taught with a managed programming language such as Java/.Net or even scripting languages such as Python. In our CSE department, the prerequisite courses are predominantly taught in Java, with only one prerequisite data structures course (CSE 274) that utilizes C++. However, PDC is predominantly performed using systems programming language such as C or C++. The transition to a systems programming language is a significant challenge for many students [25], [16].
3) **Challenges due to novel computational platforms**: Modern supercomputers, distributed computing platforms, and computational clouds predominantly run using the GNU-Linux operating system. Adeptness with GNU-Linux is often a requirement in professional settings. Therefore GNU-Linux environment is a good choice for teaching PDC concepts. However, most prerequisite courses rely on desktop operating systems such as Microsoft® Windows™. Consequently, for most students PDC courses typically provides the first experience in using a GNU-Linux environment via a traditional Command Line Interface (CLI).

The aforementioned multifaceted challenges posed hurdles in effective teaching and learning of PDC concepts.

A. *Overview of proposed approach*

The preliminary assessments and identification of the pedagogical challenges in teaching and learning PDC concepts led us to hypothesize that – "A spiral bilingual-programming approach will enable effective teaching and learning of core Concurrency and Parallelism (C&P) concepts." Accordingly, a novel spiral bilingual-programming approach in which topics are remphasized with increasing depth of coverage using two different programming languages, namely Java and C++, was introduced and assessed. The motivation for a combined spiral and bilingual-programming is to address the pedagogical challenges discussed in Section II. The proposed approach increases pedagogical modalities and learning opportunities for multifaceted com- prehension to enable synthesis of various concepts as discussed in detail in Section III. Feedback and statistics collated from both formative direct assessments and summative indirect assessments conducted during the course to evaluate the effectiveness of the proposed approach are discussed in Section IV. Section V concludes the paper summarizing the key findings from this study.

B. *Related research: Practices in teaching computer programming*

The pedagogical challenges involved in teaching and learning programming continues to remain an active area of research as related technologies evolve and computing continues to permeate new disciplines. A common approach to programming, that applies to both novice learners as well as programmers introduced to new paradigms, is that of "bricolage" [25] – *i.e.,* students often develop programs through unstructured thought processes and by testing them with only given samples of inputs. Consequently, students are unable to effective advance from "knowledge" and "comprehension" to "application" or "analysis" cognitive levels as defined by Bloom's taxonomy [25]. Moreover, students face cognitive obstacles and misconceptions that make it different for them to explain program function or justify their algorithmic strategies.

The teaching and learning difficulties associated with computer programming have spurred a broad range of efforts involving active learning supported by constructivist learning theory [4]. Information technologies are often used to provide appropriate feedback on programming using online systems and forums [21], intelligent tutoring systems [16], [8], and various online systems [11]. Bi-modal approaches that involve a combination of text and graphics has been proposed to ease promoting computational thinking [14], [7]. Web-based interactive tools with animations has been proposed for teaching sequential programming in C++ and Java [6].

The aforementioned investigations focus on teaching conventional sequential or serial algorithms and programming concepts. Furthermore, most of the courses typcically use only a single programming language throughout the course to streamline various aspects including course contents, laboratory setup, training of teaching assistants, and student experience. Nevertheless, since many programming languages and technologies continue to predominantly utilize English, efforts on using bilingual approaches are growing in importance in computing disciplines [18], [23]. However, such bilingual or multilingual approaches focus primarily on spoken languages rather than programming languages.

A limited amount of work has been done on examining student understanding of parallelism [17]. The use of concept maps to explore understanding of parallelism has been proposed by Lammers *et al* [17]. However, Lammers *et al* do not draw any conclusions about the effectiveness of concept maps due to small sample size (16 participants) of their experiments. Recently Brown *et al* [5] discuss series of modifications to multiple courses to introduce PDC concepts, including changes to an elective course titled "Object-Oriented Programming Using C++ and Java". However, Brown *et al* do not use a spiral approach and explore concepts of parallelism using Java and C++. Freitas [10] use a series of projects based on many-core processors to motivate learning concepts of concurrency and parallelism. A similar approach has been utilized by Iparraguirre *et al* [15] to introduce parallel processing via an elective course and draw the following inferences from student feedback — although students perceive they understand PDC concepts, they have a hard time when applying them; and when working on projects, the students' time was dominated dealing with basic concepts and technology, rather than solving the issues. These observations are consistent with ours and corroborate the challenges discussed in Section II, but Iparraguirre *et al* do not address these issues in [15]. Extensive literature survey indicates that the proposed spiral bilingual-programming method is unique and a similar method has not been reported in the literature.

C. *Distinguishing aspects and contributions of proposed pedagogical method*

In contrast to the aforementioned broad body of research, this investigation distinguishes itself by: ❶ focusing on a bilingual-programming approach, in which two different computer programming languages, namely Java and C++; ❷ the investigation uses a spiral approach to address the multifaceted hurdles discussed in Section II; ❸ the focus of this article is on teaching and learning core concepts of concurrency and parallelism (C&P) in an undergraduate computer science curriculum; and ❹ Several educators have provided anecdotal mentions of using some aspects of the proposed methods (as part of personal discussions) but had not conducted any assessments or published their curricular enhancements. Consequently, another contribution of this paper includes quantitative results from various assessments.

III. Method: Spiral bilingual-programming

In response to the issues highlighted in the foregoing discussions, a spiral bilingual-programming approach for teaching PDC concepts was pursued. The spiral approach is a well established pedagogical method in which topics are reemphasized with increasing depth of coverage. In the spiral approach, basic concepts are initially learned without much emphasis on many of the underlying details and nuances. The initial learning typically occurs at the "knowledge" or "comprehension" level in Bloom's taxonomy [2]. However, the same concepts are reemphasized with increased level of details enabling advancement of concepts to higher order processes of "analysis" and "application" [2]. The spiral approach for learning and teaching was pioneered by Bruner [24] and has been widely used in many disciplines including science, and mathematics [19]. Studies have reported successful use of spiral approach for teaching programming [26], [22].

In this context, the primary objective of using the spiral approach is to provide a degree of "separation of concerns" and balance the three hurdles discussed in Section II. The initial iteration provides a gentle introduction to PDC concepts while focusing on gaining familiarity environment and programming language. Subsequent iterations emphasize PDC concepts and further expanding programming expertise.

In addition to the spiral approach, a bilingual-programming strategy is also proposed. The motivation for utilizing different programming languages is multifaceted. The primary objective is to facilitate distilling the core PDC concepts from the semiotics and semantics of programming languages. The concepts were initially introduced using Java, the programming language that students are most familiar with and then the concepts were spirally reemphasized in more depth in C++. The primary objective is to aid students to acquire a through understanding of the various PDC concepts while leveraging familiarity with a programming language to provide students with a "comfort zone" in an otherwise unfamiliar programming environment. Another motivation is also to ensure students were conversant with PDC concepts in at least two widely used programming languages so as to aid with securing jobs or pursuing graduate studies. Moreover, such an approach increases the number of diverse modalities and learning opportunities for multifaceted comprehension. Analogous, multilingual strategies that utilize different spoken languages have shown to be effective in facilitating teaching and learning of complex concepts in various disciplines such as: computing, mathematics, and sciences [3], [18], [23].

The proposed method and this paper focus on the collective effectiveness of spiral and bilingual-programming aspects and not on each aspect individually. The collective treatment is needed to address the pedagogical challenges discussed in Section II – *i.e.,* spiral learning with one programming language is not sufficient meet all the learning outcomes, while just bilingual-programming (without spiraling) would merely exacerbate the pedagogical challenges already faced by students.

A. Course enhancements for spiral bilingual-programming

A spiral bilingual-programming strategy introduced in Section III was piloted in two consecutive offering of an operating systems course. Figure 2 summarizes the concepts covered in the spiral bilingual-programming modality. Initially, basic PDC concepts of multithreading, multiprocessing, and synchronization, and multiprocessing are introduced to students using a programming language that is familiar to students (shown with L1 labels). Subsequently, as illustrated in Figure 2, the topics are revisited in greater detail and exercised using a different programming language (shown with L2 labels). The additional instruction was accommodated by converting several introductory and background topics into extended reading-centered, essay style homework. Students were required to read selected chapters from the textbook and then respond to numerous short and few essay style questions to demonstrate knowledge of these topics.

In this study the two popular programming languages used were Java and C++. In the first iteration C++ was initially used followed by Java while in the second iteration Java was used as the initial language followed by C++. Accordingly, selected laboratory exercises and homework projects were performed in one language and the concepts were reemphasized in greater detail using the other programming language as illustrated by Figure 2. The initial iterations explored concepts of concurrency and issues of race conditions using a single shared object with two threads using one programming language. The next iterations expanded on the concepts and discussed parallelism, programming patterns such as producer-consumer, and synchronization between multiple threads using the second programming language. However, several laboratory experiments and homework assignments required students to explore PDC concepts in both Java and C++. Selected programming projects required students to compare and contrast their solutions against solutions supplied in the alternative language.

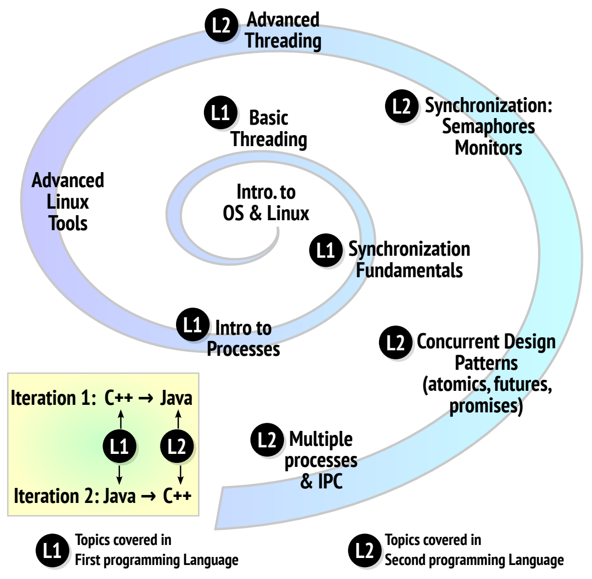

L2 Advanced Threading
L1 Basic Threading
L2 Synchronization: Semaphores Monitors
Advanced Linux Tools
Intro. to OS & Linux
L1 Synchronization Fundamentals
L1 Intro to Processes
L2 Concurrent Design Patterns (atomics, futures, promises)
Iteration 1: C++ → Java
L1
L2
Iteration 2: Java → C++
L2 Multiple processes & IPC
L1 Topics covered in First programming Language
L2 Topics covered in Second programming Language

Fig. 2. Salient PDC topics covered in spiral bilingual-programming approach

One comprehensive two phase programming project required students to develop solutions in both programming languages. The project involved using a given (specified as command-line argument) number of threads to process text files whose path was stored in a shared input queue. The text file processing required the program to display number of lines, words, and valid English words in the file. The results from the threads had to be coordinated to ensure the order of outputs matched the order in which files were present in the shared input queue. In addition to developing the programs, the second phase of the project also required the students to develop a detailed report contrasting various aspects of the implementations in the two programming languages, such as: design, lines of code, development time, run-time performance, and peak memory usage (please see `Homework 10 Report Document`).

IV. Results & Discussions: Analysis of direct and indirect assessments

Assessment of the spiral, bilingual-programming approach was conduced using several direct and indirect methods. Several direct formative assessments were conducted as an integral part of the course in the form of laboratory exercises and homework assignments. Indirect assessments were conduced as surveys from students. The surveys were optional as per Institutional Review Board (IRB) requirements that were stipulated by the IRB while granting approvals for the various surveys and assessments published in this article. Furthermore, additional indirect assessment was collected in conjunction with the department's ABET accreditation process. This section presents the data collated from the formative and summative assessments conducted from two consecutive course offerings, first iteration with C++ followed by Java and second iteration with Java followed by C++.

A. *Course setting and student demographics*

The spiral bilingual-programming approach was used to teach selected parallel and distributed computing (PDC) concepts in an undergraduate Operating Systems (OS) course. Due to prerequisite courses, the students have at least a junior standing when they enroll for the OS course. The first course offering consisted of 39 students with an almost even split between juniors and seniors, with seniors slightly outnumbering the juniors. The second course offering consisted of 46 seniors and just 10 junior students. Furthermore, the classes included students (shown in format $\langle iteration1, iteration2 \rangle$, from the following four majors from two different departments: $\langle 23, 43 \rangle$ computer science and $\langle 4, 1 \rangle$ software engineering students from the CSE department; $\langle 9, 11 \rangle$ computer engineering and $\langle 3, 1 \rangle$ electrical engineering students from the Electrical and Computer Engineering (ECE) department.

B. *Results from formative, direct assessments*

The formative, direct assessment of student learning was conducted in both supervised laboratory settings and via homework assignments. The laboratory exercises were designed to guide the students in developing short programs on specific topics, validating their programs, conducting experiments to collect performance data, tabulating the results, and recording their conclusions based on inferences drawn from their observations. Several laboratory exercises consisted of converting C++ program to corresponding Java versions and vice versa. The laboratory experiments also compared and contrasted performance of multithreaded versions using a varying number of threads to single threaded versions.

The inferences and conclusions submitted by the students in the laboratory notes indicated that the students' understanding of the concepts increased with the use of multiple programming languages. For example, students were expected to use runtime information of a multithreaded program that they developed to answer questions on theoretical versus observed runtime and performance characteristics of the programs. Initially, when only one language was used, the inferences drawn by the students were rather terse and to the point. Furthermore, the students descriptions favored the use of language-specific keywords and API methods names (such as: `synchronized`, `notifyAll`, etc.) rather than equivalent PDC concepts (such as: critical section, mutex, etc.). In contrast, similar exercises performed using a second languages elicited more deliberations and promoted the use of appropriate terminology rather than API method names. This observation strongly suggests that practice in two different programming languages facilitates students to distill concepts from their implementations. Moreover, students engagement in the form of spontaneous discussions contrasting the programs in the two languages notably increased, particularly on the topic of synchronization and runtime performance.

Laboratory experiments involving performance improvements due to multithreading elicited verbose commentary from the students as they explored various configurations and varying number of threads. Similarly, experiments exploring

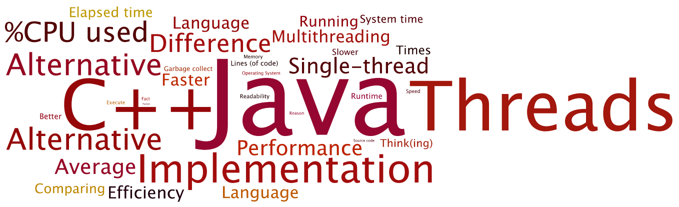


Fig. 3. High frequency terms to highlight common themes in student reports.

the effect of compiler optimizations (using compiler flags `-O2` to optimize and `-O3` to optimize aggressively) also excited the students and piqued their interest. These performance optimizations, particularly the ability to multithread programs, seemed to swing the student's interest to focus on PDC concepts and increased questions about internals of how the libraries accomplished the task. Furthermore, the compiler optimizations triggered discussions on instruction set architectures, pipelining, Streaming SIMD Extensions (SSE), and superscalar operations on modern CPUs. The topics also spurred discussions contrasting the Java Virtual Machine (JVM) against native compilers in the context of Parallel and Distributed Computing (PDC).

Similarly, projects that required students to contrast solutions from first iteration against those developed in the second iteration elicited a more vibrant discussion from the students. Specifically, in the second iteration, solutions utilizing modern concurrent design patterns [27] such as: atomics, promises, and futures (see Figure 2) excited students about these topics. The students focused not only on performance comparisons but also on contrasting various aspects of the two programming languages and libraries as well. The reports submitted by students for bilingual projects were fairly lengthy with an average of 317 ($\pm$163) words. The key topics involved in the student discussions are summarized by the word cloud in Figure 3. As illustrated by Figure 3, the bilingual-programming approach elicited attention on much broader range of topics, including: alternative approaches, comparative analysis, system calls, garbage collection, readability, and efficiency.

The projects also encouraged friendly competition between two camps of students, namely those who seemed to prefer Java versus those who seemed to favor C++. The friendly competition had a positive effect on lecture sessions motivating further introspection and discussions on various aspects of PDC. These observation were consistent in both versions when C++ was taught first followed by Java and vice versa. This observation indicates that the order of languages may not be as significant as contrasting solutions in two languages.

C. *Results from direct survey assessments*

A set of 18 PDC questions (please see supplementary material for the complete set of questions) were developed to systematically assess progress of students when the spiral bilingual programming approach was employed. The predominant subset of questions focused on core PDC concepts and were programming language agnostic. The survey consisting of 18 questions were administered to the students three times

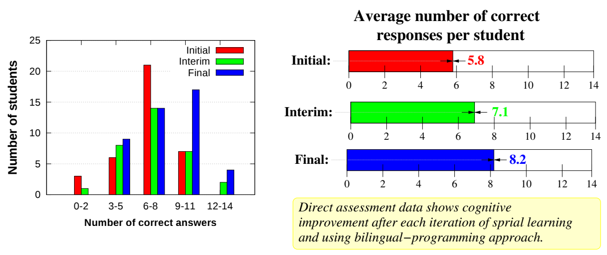


Fig. 4. Histogram of number of correct answers to survey assessment before, after initial coverage, and second coverage of topics along with average number of correct answers per student.

and the results from them are summarized in Figure 4. The initial round of surveys (shown in red in Figure 4) were collected just after introductory materials on multithreading and multiprocessing were covered in the course. The interim data was gathered after the topics were covered in additional depth in Java. The final surveys were collected after spiraling over the topics in greater depth in C++. Since the assessments were optional, the number of students who responded to the survey questions varied.

The mean (μ), median ($\tilde{x}$), and mode ($\hat{x}$) for the scores from the initial, interim, and final surveys shown in Figure 4 were: μ: 6.55 (σ: 1.97), $\tilde{x}$: 6, $\hat{x}$: 6; μ: 7.12 (σ: 2.56), $\tilde{x}$: 8, $\hat{x}$: 6; and μ: 8.1 (σ: 2.47), $\tilde{x}$: 8, $\hat{x}$: 9 respectively. The data indicates that the students comprehension of concepts consistently improved as expected in the span of 2.5 to 3.5 weeks between successive assessments.

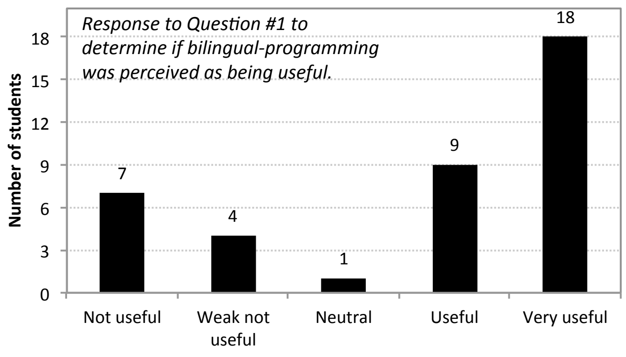
The survey questions that showed strong growth between the three surveys are tabulated in Table II. As summarized in the table, although the questions are conceptual and language agnostic, the proposed spiral bilingual-programming approach improved understanding of core PDC concepts. Classroom experience suggests that delving into the details underlying Java's `java.lang.Process` and `java.lang.Thread` APIs via the use of `fork` and `clone` Linux system calls helped students obtain a better understanding of the operations.

D. SGID: Indirect Assessment

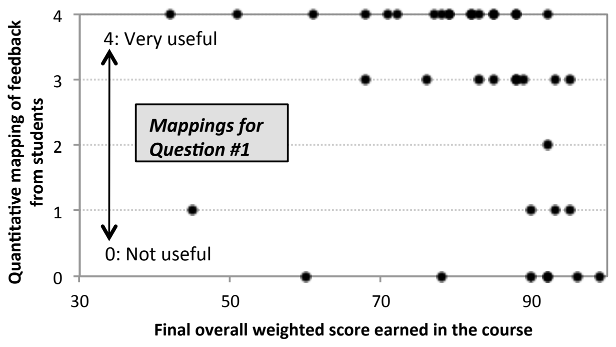
The University promotes the use of Small Group Instructional Diagnosis (SGID) that uses small-group discussion among students, lead by the University's Advanced Learning Technologies (ALT) staff (*i.e.,* not by any faculty), to provide informal feedback to an instructor during a course. In the SGID, 98% of the student voiced their opinion that teaching concepts first in Java and then delving into details with C++ as one of the significant strengths of the course. Furthermore, 100% of the students agreed that the class gradually proceeded faster with increased coverage, reflecting the increased level of detail and pace of the spiral teaching approach.

E. Results from summative indirect assessments

The summative, indirect assessments were collated from student feedback using both an identified and an anonymous survey. The identified feedback was obtained only in the first iteration (IRB suggested to avoid identifiable assessments



(a) Histogram of quantitative mapping of qualitative feedback obtained from students for the free form question in Section IV-E.



(b) Scatter plot of quantitative mapping of student feedback for Question #1 (see Section IV-E) versus the students' final weight score.

Fig. 5. Summary of statistics collated from summative indirect assessments administered to students

for second iteration) as responses to the following question presented to the student at the end of their final exam:

Question: Unlike previous semesters, this semester covered threading and synchronization in two different languages, namely C++ and Java. Briefly comment on whether you think this approach of covering same concepts from two different languages was helpful to you to understand the concepts better. Note: This question is soliciting your honest opinion and therefore there is no correct or incorrect answer for this question.

The students' free form responses were then mapped to a Likert scale in the range 0 to 4 (inclusive), where 0 corresponds to "not useful" while 4 indicates "very useful". The histogram shown in Figure 5 shows the distribution of the feedback to the above two questions. Mapping of qualitative feedback from the students to the Likert scale was performed using selected keywords that were consistently used by the students. For instance, if the student used phrases such as: "very useful" or "very helpful" etc. those feedback was mapped to 4. On the other hand, phrases such as "not useful" or "unnecessary" were mapped to 0. Other feedback that expressed mixed opinions were suitably mapped to suitable numeric scores depending on the degree to which the opinions were expressed.

TABLE II

MULTIPLE CHOICE QUESTIONS THAT SHOWED STRONG IMPROVEMENT BETWEEN INITIAL, INTERIM, AND FINAL SURVEYS (FULL SET OF QUESTIONS ARE IN SUPPLEMENTARY MATERIALS)

Survey question (multiple choices not shown for brevity)		# Correct answers		
		Initial	Interim	Final
Q#2	The simplest concurrent program must have	44.7%	58%	63.63%
Q#3	The simplest parallel program must have	28.94%	58.06%	72.72%
Q#4	The minimum number of CPUs and cores that are absolutely necessary to utilize concurrency inherent in a multi-threaded program (with 8 threads) is	31.57%	29.03%	65.9%
Q#8	In the Linux operating system, creating (or forking) a new process	15.78%	22.58%	34.09%
Q#10	A race condition can occur	55.2%	67.7%	70.45%
Q#15	The types of resources that can be shared between two threads are	52.6%	54.8%	75%
Q#18	When a parent process creates two child processes, then the easiest way for a parent to tie/connect the the two child processes is by	26.3%	41.9%	43.18%

The data suggests that many students perceived that bilingual-programming was indeed beneficial, consistent with SGID data presented in Section IV-D. However, as indicated by Figure 5(a) about 28% of the students did not express the same opinion. In order to further explore this finding, the identifiable feedback from exams was correlated with the student's overall weighted score in the course. The scatter plot as shown in Figure 5 attempts to illustrate the relationship between final scores and student opinion regarding effectiveness of bilingual-programming approach pursued in the course for teaching PDC concepts.

The plot in Figure 5(b) indicates that there was a weak negative correlation between final scores and student opinions. The Pearson correlation coefficient for the data was -0.276 indicating that some of the high scoring students did not consider the bilingual-approach to have had a significant impact on their learning. The observations were confirmed through another round of review of the student feedback. Most of the stronger, high scoring students (namely students that earned a “B+” grade or higher) indicated preference to have covered more advanced topics in the time that was invested in the first round of the spiral to introduce PDC concepts. These students seem to be able to grasp many of the core PDC concepts and are able to effectively apply it in other languages as well. Nevertheless, almost all of these high achieving students provided positive feedback about the knowledge and experience gained from comparative analysis of Java and C++. They indicated that bilingual-programming provided them with a better understanding and appreciation of design alternatives and PDC concepts.

F. Data from Course Outcomes Survey

The computer science program offered by the CSE department is ABET accredited and uses a system of outcomes-based assessment for the program and for each course in the curriculum. Continuous collation of necessary data is facilitated via a custom Learning Management System (LMS), called Cascade [13], that has been developed by the Department. Cascade-LMS was customized to collect pre- and post-course survey question to indirectly assess learning outcomes for the changes discussed in Section III-A. The questions were designed to assess appropriateness of topic, level of coverage, learning outcomes, and impact on program outcomes. However, it must be noted that the outcomes survey is optional and students are not obligated to complete the survey.

Table III lists the pre- and post-course survey responses. Pre-course responses were obtained from 41 students while post-course surveys were completed by just 26 students. The numerical values in the range 1 through 5 correspond to the following set of choices — *1*: strongly disagree, *2*: disagree, *3*: indifferent, *4*: Agree, and *5*: strongly agree. Lower average values for pre-course outcomes survey indicates that students are not very confident about their understanding of the topic. Conversely, higher numbers recorded for post-course outcomes survey indicate that most students felt that they had gained a better understanding of the PDC concepts after completing the course.

G. Feedback from Course Evaluations

In addition to direct and indirect assessments, routine *anonymous* course evaluations were also collected from students electronically. The course evaluations are voluntary; 90% and 76% of students completed the evaluations in the two iterations of the course. The course evaluations are obtained in a Likert scale of 0 through 4. Overall the evaluations for the two course offerings were very positive with an overall student rating of 3.38 (SD: 0.5) and 3.61 (SD: 0.54) which were above the department averages of 2.8 and 3.2 in the two years respectively. The students also highly rated the contribution of this course to their education as reflected by responses to the following questions — Analyze problems and issues: 3.6 (SD: 0.68) and 3.9 (SD: 0.3); Appreciation for topic increased: 3.49 (SD: 0.87) and 3.78 (0.41); and Increased understanding of material: 3.51 (SD: 0.69) and 3.61 (SD: 0.54).

H. Overall Average GPA Trends

The overall GPA trends observed over several years is shown in Figure 1. As illustrated by the chart, increased coverage of contemporary PDC concepts clearly proved to be a challenge for the student body. However, the proposed spiral, bilingual-programming method had a positive effect on student learning and overall course outcomes despite increased coverage of contemporary PDC concepts.

TABLE III
STUDENT FEEDBACK TO SELECTED SUBSET OF OUTCOMES SURVEY QUESTIONS PERTAINING TO PDC CONCEPTS. SET OF VALUES IN { } ARE NUMBER OF STUDENTS SELECTING THE FOLLOWING CHOICES (IN THE SAME ORDER): *5: strongly agree*, *4: agree*, *3: indifferent*, *2: disagree*, AND *1: strongly disagree*. NUMERICAL SCORE WAS COMPUTED AS WEIGHTED AVERAGE OF OF MAPPING THE CHOICE TO SCORES IN THE RANGE 5 THROUGH 1.

Survey question	Pre-course Data	Post-course Data
I am able to understand fundamental issues in concurrency	2.15 {0, 1, 2, 15, 8}	4.15 {11, 18, 8, 3, 1}
I am able to demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks	2.05 {0, 2, 9, 18, 11}	4.19 {9, 15, 1, 0, 1}
I am able to apply Concurrency and Parallelism in Practice	2.02 {0, 3, 9, 13, 14}	4.32 {9, 15, 1, 0, 0}
I am able to summarize the range of mechanisms that can be used to realize concurrent systems & describe their benefits	1.9 {0, 0, 8, 20, 12 }	4.27 {8, 17, 1, 0, 0}
I am able to Implement a deadlock-free multi-threaded program using suitable concurrent constructs in major languages	1.93 {1, 1, 8, 15, 17}	4.16 {8, 12, 4, 0, 1}
I am able to develop multi-threaded program using appropriate design patterns (such as: promise, future, etc.)	1.84 {0, 0, 7, 18, 13}	4.3 {11, 12, 3, 0, 0}
I am able to describe the use of High Level Language libraries for multi-threaded algorithms (such as: parallel sorting)	1.78 {0, 0, 8, 16, 17}	4.16 {7, 14, 3, 0, 0}

V. CONCLUSION

The advancement of multi-core and multiprocessor computing platforms require effective use of Parallel and Distributed Computing (PDC). PDC concepts is a mainstream software development skill that is required in industry, military, and academia. Consequently, many academic institutions are continuing streamline and expand coverage of PDC concepts in their their curricula and adapting to the pedagogical challenges that arise in the process.

This article discussed one such effort, that explored the effectiveness of using a spiral bilingual-programming approach for teaching core PDC concepts in an undergraduate Operating Systems (OS) course. Specifically, application of several core parallelism and concurrency topics were practiced in both Java and C++. The primary objective for pursuing a spiral bilingual-programming approach was to address the steep learning curve and challenges experienced by the students due to introduction of new programming language, laboratory environments, and complex PDC concepts. The two programming languages, namely Java and C++, were employed to balance various factors such as: leveraging student background from prerequisite courses, coverage of contemporary skills, and adherence to program outcomes for continued accreditation. This paper presented the changes introduced to the course along with analysis of direct and indirect assessments conducted as part of the course.

This study focused on the collective effectiveness of spiral and bilingual-programming aspects and not on each aspect individually. The collective treatment is needed to meet the effective learning and outcomes – i.e., spiral learning with one programming language is not sufficient meet all the learning outcomes, while just bilingual-programming (without spiraling) would merely exacerbate the pedagogical challenges already faced by students.

The formative, direct assessments conducted during the course indicated that the spiral bilingual-approach had a positive impact on student participation and learning. The approach spurred lively interaction from the students in the form of questions and discussions comparing and contrasting programming languages. Topics related to multithreading, performance, and performance comparisons seem to hold a special appeal to the student community as ascertained from direct assessments of student homework assignments.

Quantitative mapping and analysis of the student feedback regarding the spiral bilingual-programming approach showed a weak negative correlation between student opinions and overall final scores earned in the course. The negative correlation and feedback indicated that the small subset of high achieving students are confident that they can grasp concepts taught in one programming language and apply them in the context of another programming language. On the other hand, majority of the students who earned a "B" or lower indicated a strong preference for spiral bilingual-programming. In feedback forms, these students often emphatically asserted that the spiral bilingual-programming approach significantly bolstered their understanding of PDC concepts. The students also commented positively on experience gained using two popular programming languages and their contributions toward their continued education and future employment. The proposed approach also had a marked positive impact in overall average GPAs from the course despite introduction of advanced PDC concepts.

The pedagogical experiences and assessments conducted as part of this research support the use of spiral bilingual-programming approach in teaching core PDC concepts. However, the bilingual-programming approach does requires substitution of selected topics to accommodate the necessary instruction. Experience suggest that converting suitable topics into inverted classes and reading-centered homework assignments are effective strategies to accommodate the proposed approach into existing curricula. The assessments collated from two batches of students indicates that spiral bilingual-programming is an effective pedagogical approach for emphasizing selected concepts and deepen student knowledge.

References

[1] ACM/IEEE, *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, Association of Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE), Dec 2013, http://www.acm.org/education/CS2013-final-report.pdf.

[2] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, 2nd ed. Pearson, Dec. 2000.

[3] A. Arruarte, J. A. Elorriaga, I. naki Calvo, M. L. naga, and U. Rueda, "Computer-based concept maps for enabling multilingual education in computer science: A basque, english and spanish languages case," *Australasian Journal of Educational Technology*, vol. 28, no. 5, pp. 793–808, Jul. 2012. [Online]. Available: http://www.ascilite.org.au/ajet/ajet28/arruarte.html

[4] A. Berglund, M. Daniels, and A. Pears, "Qualitative research projects in computing education research: An overview," in *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ser. ACE '06, vol. 52. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 25–33. [Online]. Available: http://dl.acm.org/citation.cfm?id=1151869.1151875

[5] C. Brown, Y.-H. Lu, and S. Midkiff, "Introducing parallel programming in undergraduate curriculum," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1269–1274.

[6] M. Bruce-Lockhart and T. Norvell, "Developing mental models of computer programming interactively via the web," in *Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07. 37th Annual*, Oct 2007, pp. S3H–3–S3H–8.

[7] J. E. Burge, G. C. Gannod, M. Doyle, and K. C. Davis, "Girls on the go: A cs summer camp to attract and inspire female high school students," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13. New York, NY, USA: ACM, 2013, pp. 615–620.

[8] K. Chrysafiadi and M. Virvou, "Dynamically personalized e-training in computer programming and the language c," *Education, IEEE Transactions on*, vol. 56, no. 4, pp. 385–392, Nov 2013.

[9] CSE Department, "Computer science and software engineering," 2014. [Online]. Available: http://www.cec.miamioh.edu/departments/cse

[10] H. C. de Freitas, "Method for teaching parallelism on heterogeneous many-core processors using research projects," in *Frontiers in Education Conference, 2013 IEEE*, Oct 2013, pp. 108–113.

[11] K. C. Ekwunife-Orakwue and T.-L. Teng, "The impact of transactional distance dialogic interactions on student learning outcomes in online and blended environments," *Computers & Education*, vol. 78, no. 1, pp. 414 – 427, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131514001444

[12] N. Giacaman, "Teaching by example: using analogies and live coding demonstrations to teach parallel computing concepts to undergraduate students," in *Proceedings of the 2012 NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-12)*, Shanghai, China, May 2012. [Online]. Available: http://www.cs.gsu.edu/~tcpp/curriculum/?q=advanced-technical-program

[13] M. T. Helmick and G. C. Gannod, "Streamlining and integration of miami three-tier outcomes assessment for sustainability," in *The*

[16] M. Kordaki, "A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation," *Computers & Education*, vol. 54, no. 1, pp. 69–87, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131509001845

Proceedings of the 39th ASEE/IEEE Frontiers in Education Conference (FIE 2009), San Antonio, TX, Oct. 2009. [Online]. Available: http://www.cascadelms.org/

[14] K. Howland and J. Good, "Learning to communicate computationally with flip: A bi-modal programming language for game creation," *Computers & Education*, vol. 80, no. 0, pp. 224 – 240, 2015.

[15] J. Iparraguirre, G. R. Friedrich, and R. J. Coppo, "Lessons learned after the introduction of parallel and distributed computing concepts into ece undergraduate curricula at utn-bahia blanca argentina," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012, pp. 1317–1320.

[17] G. Lammers and C. Brown, "Exploring student understanding of parallelism using concept maps," in *Frontiers in Education Conference (FIE), 2012*, Oct 2012, pp. 1–5.

[18] C. Li-qing and Y. Yun-yang, "Practice and research on bilingual teaching in the course operating systems," in *Computer Science and Education (ICCSE), 2010 5th International Conference on*, Aug 2010, pp. 784–787.

[19] I. Milne and G. Rowe, "Difficulties in learning and teaching programming–views of students and tutors," *Education and Information Technologies*, vol. 7, no. 1, pp. 55–66, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1015362608943

[20] S. K. Prasad, A. Chtchelkanova, S. Das, F. Dehne, M. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc, M. Lumsdaine, D. Padua, M. Parashar, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu, "Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing: core topics for undergraduates," in *Proceedings of the 42nd ACM technical symposium on Computer science education*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, pp. 617–618. [Online]. Available: http://doi.acm.org/10.1145/1953163.1953336

[21] R.-S. Shaw, "A study of the relationships among learning styles, participation types, and performance in programming language learning supported by online forums," *Computers & Education*, vol. 58, no. 1, pp. 111–120, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131511001916

[22] B. Shneiderman, "Teaching programming: A spiral approach to syntax and semantics," *Computers & Education*, vol. 1, no. 4, pp. 193–197, 1977. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0360131577900082

[23] L. Shukun and Y. Xiaohua, "The application of bilingual teaching of operating system in private college," in *Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on*, June 2010, pp. 82–85.

[24] S. Smidt, *Introducing Bruner: A Guide for Practitioners and Students in Early Years Education*. Routledge, 2011.

[25] J. Tan, X. Guo, W. Zheng, and M. zhong, "Case-based teaching using the laboratory animal system for learning c/c++ programming," *Computers & Education*, vol. 77, no. 0, pp. 39–49, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131514000888

[26] F. Veladat and F. Mohammadi, "Spiral learning teaching method: Stair stepped to promote learning," *Procedia - Social and Behavioral Sciences*, vol. 29, no. 0, pp. 1115–1122, 2011, the 2nd International Conference on Education and Educational Psychology 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877042811028060

[27] A. Williams, *C++ Concurrency in Action: Practical Multithreading*. 20 Baldwin Road, Shelter Island, NY 1196332, USA: Manning Publications, Feb. 2012. [Online]. Available: http://my.safaribooksonline.com/9781933988-771