Teaching Parallel Programming for Beginners in Computer Science

Davi Jose Conte
Institute of Mathematics and Computer Sciences
University of São Paulo
São Carlos, Brazil
daviconte@usp.br

Paulo Sergio Lopes de Souza
Institute of Mathematics and Computer Sciences
University of São Paulo
São Carlos, Brazil
pssouza@icmc.usp.br

Guilherme Martins
Institute of Mathematics and Computer Sciences
University of São Paulo
São Carlos, Brazil
guilhermemartins@usp.br

Sarita Mazzini Bruschi
Institute of Mathematics and Computer Sciences
University of São Paulo
São Carlos, Brazil
sarita@icmc.usp.br

***Abstract*—This research full paper describes our experience in teaching parallel programming for students without previous knowledge of basic concepts of computing, comparing their levels of learning. The use of parallel software grew considerably in recent years due to the increasing availability of multi and many-core devices. The evolution of hardware and software resources collaborated for a remarkable computational processing power offered by parallel programs. However, parallel programming is taught usually in more advanced years of the undergraduate computer courses, due to its supposed prerequisites as sequential programming, operating systems, computer architectures and others. Postponing parallel programming teaching hinder students to apply parallelism other subjects, reducing the probability of these future professionals think on parallel solutions naturally. We executed 05 experiments teaching parallel programming subjects for 252 students. We analyzed whether students without prerequisites could learn parallel programming in the same level verified with students with prior computing knowledge. We used three different teaching methodologies: traditional, Problem Based Learning (PBL), and Team-Based Learning (TBL). The teaching and learning evaluation took into account such metrics: parallelism thinking of students, use of programming-model, correct output of the program, source-code readability and satisfaction of the students. The paper shows that it is possible to teach parallel programming to students without previous knowledge of computing, obtaining high scores and interest in such learning. Our results contribute positively to disseminate parallel programming, which is vital to extract performance from nowadays computers.**

***Index Terms*—Education, Teaching, Learning, Parallel Programming, Computer Science Teaching**

I. Introduction

The need for software solutions that make proper use of parallel computing is growing in today's society. Multi/many-core hardware and computer clusters are already considered usual in academia as well as in commercial, financial, and industrial activities [1]. Today's society needs skilled computer professionals to exploit the benefits of parallel programming on these machines in the best possible way [2]. However, teaching parallel programming is not trivial [3]. Students need to change their programming strategies considering parallelism since it implies the division of a problem into independent tasks, distribution of these tasks to processing resources, and other aspects that do not exist in sequential programming [4].

Students at the beginning of computing graduation courses (first-year students) usually learn sequential programming. Nevertheless, such students do not have patterns of development yet, and if they are exposed to "parallel thinking", as soon as possible, they could incorporate the parallel paradigm naturally [1], [5], [6]. This expectation also applies to students of courses in areas other than computing, as these do not have advanced knowledge in computer programming.

Academics and market professionals believe that parallelism is one of the leading teaching contents of the computer course curriculum [7]–[9]. ACM and IEEE advocate bringing the parallel programming teaching forward in undergraduate courses, despite the dependence on other concepts [10], [11].

The available literature usually reports case studies on the use of methodologies or tools applied to parallel programming teaching. We did not find studies defining what topics could or should be appropriate for teaching parallel programming to students at the beginning of graduation, with a comprehensive and systematic assessment of students' learning [7].

We investigate in this paper the hypothesis that students without primary computer education can understand and apply concepts of parallelism, and can even develop parallel codes with quality in terms of correct practice of parallel programming fundamentals.

The objective of this paper is to evaluate the possibility of bringing the parallel programming teaching forward for students without previous computing background through class experiments. In this context, we aim to determine the depth and breadth of the content taught in classes, abstracting direct

Acknowledgement to CAPES and FAPESP (process number 2013/07375-0) for financial support.

dependencies with concepts from other not studied subjects and verify how receptive the students are to the early learning of parallel programming.

We organized topics of parallel computing in layers and identified a subset of them to teach for students without previous computing knowledge. In the sequence, we carried out experiments showing parallel computing for different classes of such students, using distinct methodologies [12]. Our main results show that the parallel programming teaching students without previous knowledge in computing can present excellent learning, analogous to the observed in mature students in computer science courses. Such results contribute positively to disseminate parallel programming, which is vital to extract performance from nowadays computers.

The remainder of this paper contains the organization below. Section II presents related work to show how our community approaches such a research topic. Section III describes our proposal to organize the Parallel Computing teaching topics in layers. The description of our experiments are in Section IV. Section V presents an analysis from our findings running the experiments. Section VI describes the threats to validity. Finally, Section VII concludes the paper.

II. Related work

This section presents other studies related to our research, grouping them into the categories described below.

Disciplines in the first year of graduation in computing: Some papers describing disciplines of introduction to parallel programming occurred in the second semester (first year) of computer courses. The proposals focus on parallel thinking, leaving the programming itself for more advanced periods. Some works use the Scratch block programming language that allows programming with visual representations. The results were positive in terms of learning and students' receptivity to the study of the topic [13]–[18].

Modifications in curricula to cover parallelism concepts: The publications in this category propose the integration of fundamental concepts of parallel and distributed programming from the first disciplines. The results show that contact at the early stages of graduation is a crucial factor for students to learn parallelism and that the idea is to develop programming practice as soon as possible. These papers show the importance of identifying specific content and developing lesson plans for the objectives of the courses [7], [19], [20].

Universities extension courses: In this category, there is a proposal for a practical course based on project development. The course contains five modules on parallel programming using GPUs and OpenCL. The course is not described with the necessary details to reproduce it and does not define metrics to evaluate the students' level of learning [9];

Educational resources: Imam and Sarkar (2014) present a framework for teaching thread mechanisms and synchronization to a second-year class of a computer science course. The framework allows for understanding parallelism concepts and developing applications for multicore architectures. Students must know Java [21]. The study of Libert & Vanhoof (2017) analyzes 27 software visualization systems to support the teaching of programming with message passing [22].

Parallelism for high school: Torbert et al. (2010) present an initiative to bring the parallel teaching forward for high school students, using Java and Python to teach message passing with MPI, Explicit Multi-Threading (XMT), PThreads, OpenMP and Cuda. The authors highlight the positive results of using XMT in the experiments [23].

Learning assessment tool: Rague (2011) presents the use of the POPS (Perceptions of Parallelism Survey) as an instrument to assess students' perception of parallelism and associated content. The results show a positive impact of practical activities on student learning [6].

The analysis of the related work in this section shows that it is still necessary to identify topics accurately for the introduction to parallel programming teaching when considering first-year students [7]. Many papers found in the literature report experiences of the use of teaching methodologies or tools applied to parallel programming teaching, but do not present a statistical evaluation of their results using quantitative and qualitative metrics. This scenario hinders a more consistent validation and analysis of the results available in the literature. Our research differs from others because it presents a ranking of parallel computing topics that can be brought forward to students. We show an evaluation of the possibility of bringing the parallel programming teaching forward in different contexts, changing factors such as methodologies, class sizes, and parallel programming models.

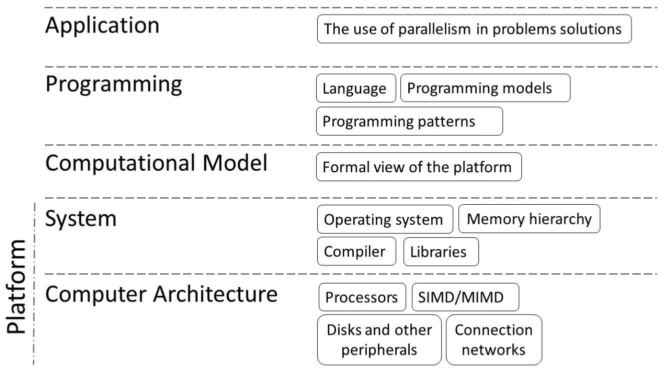
III. Organization of Layered Parallel Computing

Computing's teaching is conducted in layers and that's not new. Consider as an example the sequential programming teaching that normally occurs in the first year of computer courses. Students learn how to sequentially program computers using any language, without having prior knowledge of computer organization, operating systems or complexity of algorithms. These students abstract concepts that they have not studied yet and develop their programs even without knowing that they do not know. After the first year and already knowing how to program, students learn subjects that have been encapsulated in layers located at other levels (usually below) of a hierarchical system of layers. By learning the contents of other levels of abstraction, they will understand the motivation for concepts that will provide quality and performance to the programming they are learning in the first year of graduation. This same knowledge organization is practiced in the teaching of computer architecture, operating systems and computer networks [24]–[26].

Although this layered approach is already applied in many fields of computing, this is not the same for parallel computing teaching. By organizing the concepts of parallel computing in layers, we are able to identify which topics pertaining to parallel programming can be taught to learners who have no other prior knowledge. Hierarchical layers allow to separate contents and abstract more complex concepts in a first moment, thus bringing the parallel programming teaching forward at the same time as teaching sequential programming. This possibility of bringing the teaching forward would allow students to already learn how to program computers in parallel, creating in the future professionals a strong "parallel thinking" in their computational solutions.

We consider the IEEE Curriculum Initiative to determine the parallel computing content that should be taught to students [11]. This content has been grouped into five levels or hierarchical layers, as shown in Figure 1. The lowest layer comprises the level of **Computer Architecture** where there are various aspects of hardware as processing resources, memory, networks and other I/O devices. At this level we have taxonomies of parallel architectures like Flynn's Taxonomy. The **System** layer comprises the basic software such as operating systems, compilers and libraries that make it possible to run the parallel application over the architecture. At the next level, there is the **Computational Model** layer that abstracts the platform (architecture and system software) and focuses on aspects such as performance and security. At this level there is a formal representation of a parallel solution so that one can estimate its performance with different workloads and number of hardware resources. The **Programming** layer refers to the programming models used, including software resources used in development and languages/libraries that allow the coding of parallel algorithms. Our proposal is to bring the parallel computing teaching forward for students without previous knowledge in computing, starting with the layer of programming, just as it is done in sequential programming. Finally we have the **Application** layer, where we abstract aspects including the programming language. At this level, we consider mainly end users and data and business managers, concerned with the use of the parallel application.

Fig. 1. Organization of parallel computing contents in hierarchical layers.



IV. Experiments

We executed five experiments to analyze whether students without prior computing background can learn parallel programming effectively. The content of the experiments focused on the programming layer (defined in Section III), aiming to approach the parallelism in an algorithmic and practical view. The research questions (RQ) associated with the experiments and the hypothesis under investigation in this research are:

RQ1: Can students with no previous computing knowledge learn the concepts of parallel programming as much as students with previous knowledge?

RQ2: Can students without previous computing knowledge develop parallel code with quality in terms of correct parallelism usage, proper use of programming models, correct output, and source code readability?

RQ3: How receptive and motivated were students to learning parallel programming without the conceptual basis usually given previously?

We split the participants of the experiments in two groups, taking into account their academic background: with and without previous knowledge in computing. We also use the following metrics when evaluating parallel programs from students:

- **Correct use of parallelism**: correctly explore parallel aspects of the problem/application. Equivalent to 40% of the final weighted average value.
- **Correct use of programming model**: Proper use of available programming model. Equivalent to 30% of the final weighted average value.
- **Correct/expected output**: correct result and solution performance. Equivalent to 20 percent of the final weighted average value.
- **Legibility and good programming practices**. Equivalent to 10% of the final weighted average value.

We have chosen these metrics to standardize the evaluation criteria of the algorithms developed by the students. These metrics are comprehensive and include aspects of the design of the parallel algorithms (i.e., students' thinking for the parallel solution), how correct the algorithm's output is concerning their requirements, and the quality of the parallel code developed, taking into account the parallel programming model.

A. Experiment I: Teaching OpenMP using TBL approach

We distributed the 151 students participating in this experiment in three distinct classes of Computer Science Bachelor's, Computer Engineering and Computer Science graduate courses. All students had previous knowledge of computing and such disciplines had 45 hours long each one.

The objective of this experiment was to evaluate the learning of these students with previous knowledge, when submitted to the collaborative Team-Based Learning (TBL) methodology [27] to learn programming OpenMP. We compared the scores of this experiment with the scores of experiments whose students had no previous knowledge.

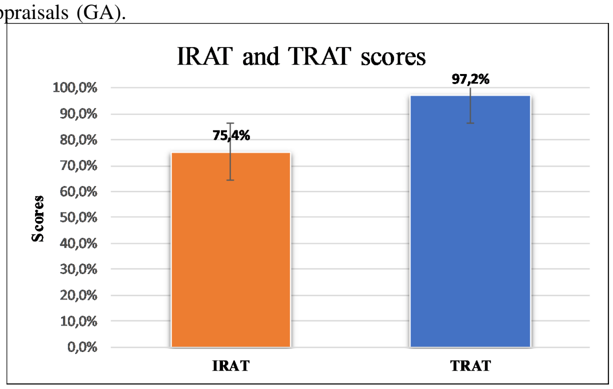
Following the stages of TBL, we divided each class into:

1) Individual preparation guarantee (15 - 20 minutes);
2) Team preparation guarantee (30 minutes);
3) Discussion about the activity (15 - 20 minutes);
4) Practical activity (45 - 60 minutes).

Figure 2 illustrates the evolution of the average score of students in the first two stages of classes (theoretical aspects only).

The values presented in the graph show that there was a significant evolution in the theoretical performance of the

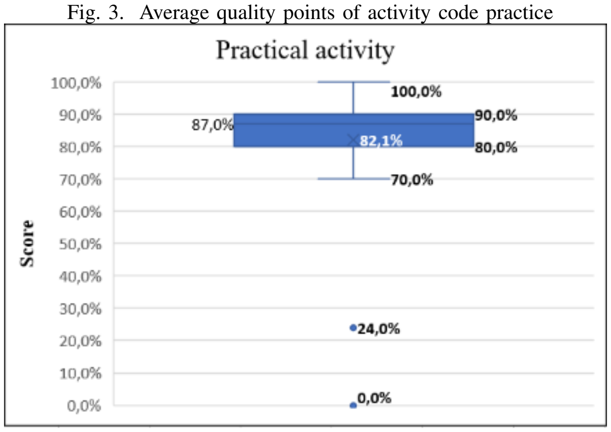
Fig. 2. Evolution of Average Scores in Individual Appraisals (IA) and Team Appraisals (GA).



students after the discussion of the questions with other members of the group, characterizing an advance from 75.4% to 97.2% in conceptual knowledge. These results show that the students of this experiment performed well in both stages and were able to learn the theoretical content satisfactorily.

Figure 3 represents the distribution of scores in the final practical activity by the total sample of students.

Fig. 3. Average quality points of activity code practice



The median shows that more than half of the notes were over 87.0%, indicating that the students of this experiment were able to develop parallel applications with quality and use the resources of the parallelism model appropriately.

Upon completion of all planned classes, an anonymous survey was sent out for students to respond to their satisfaction with the teaching method, content, and didactic. We obtained 100 responses (66% of participants), and the results show that there was the acceptance of TBL, indicating student satisfaction with the classes.

B. Experiment II: Introduction to OpenMP

This experiment verified whether students without prior computing knowledge could understand and apply the concepts of parallelism with OpenMP as much as students with prior computing knowledge.

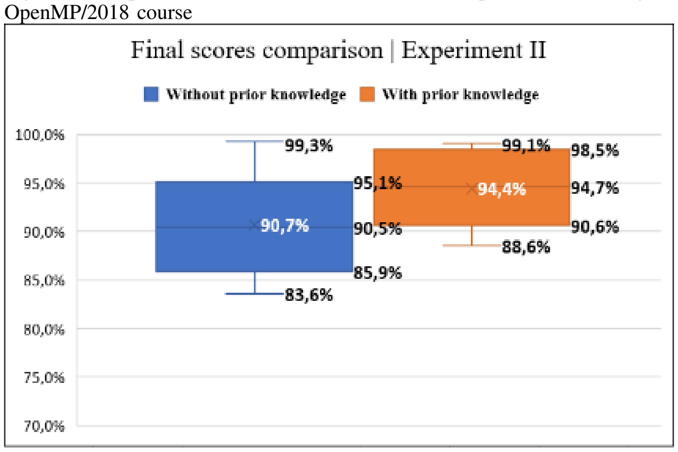
The class was composed of 15 students with no previous knowledge in computing (65.2% of the sample), and 08 students with previous knowledge (34.8%). We highlight the highest percentage in the group without previous knowledge, indicating that the subject is attractive to students in this context, whether in the first year of graduation or other areas beyond computing.

We executed this experiment as an extension course of eight hours long, with four modules of two hours long each.

The evaluation of technical knowledge occurred gradually throughout the course through 6 exercises developed individually by the students.

We asked students to develop parallel solutions using the techniques taught in the respective module. This approach made each evaluation to correspond to the application of different OpenMP resources. The graph in Figure 4 shows the dispersion of the final averages of students at the end of the course. The final average considers the evaluation of all exercises performed in the course.

Fig. 4. Comparison of students with and without previous knowledge - OpenMP/2018 course

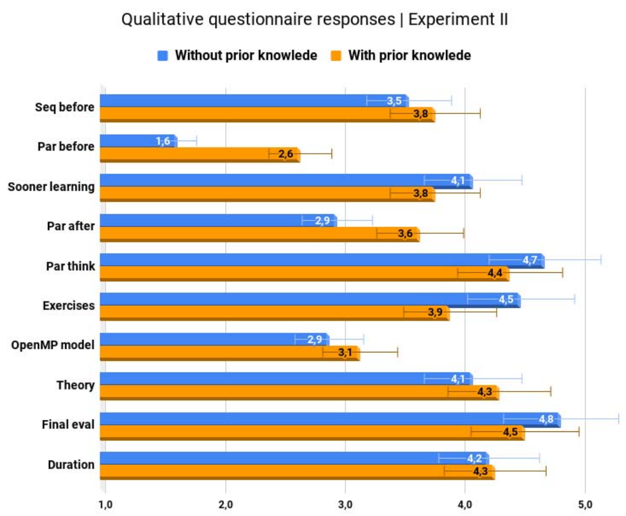


Analyzing the dispersion of the final averages, we can notice that the groups have statistically equal scores, although the averages of the students without previous training are slightly below the averages of the other group. The averages of both groups are very close to and above 90%.

We also apply a questionnaire about the students' opinions about the course and learning. The results show that the participants felt stimulated to think in parallel solutions by constructing the proposed algorithms during the course and were able to understand the principles of parallel execution. They show that the students had a positive feeling of the content taught in the course, indicating that they were able to understand the content taught, even without previous knowledge. The questionnaire is available at: http://tiny.cc/openmp_2018_qualitative, and the answers obtained are displayed in Figure 5.

Figure 5 shows in the answers to question 1 that students had prior knowledge of sequential programming, but had limited prior knowledge of parallel programming in both groups (question 2). The answers to question 4 show that students with and without prior knowledge increased their knowledge of parallel programming after the course. The answers to

Fig. 5. Results of the qualitative evaluation for OpenMP/2018 course.



question 7 demonstrate students' fears about the mastery of the OpenMP programming model. The students' justification for this fear is due to the short time of the course, which did not allow them to exercise the content taught anymore.

C. *Experiment III: Introduction to Pthreads*

The third experiment aimed to vary the programming model taught from OpenMP to PThreads and also the students who participated in classes. This way, we checked if the students in that class in experiment II or the OpenMP programming model influenced the results. As in the second experiment, the class had students with and without previous computing knowledge (3 and 14, respectively, representing 17.6% with knowledge and 82.4% without). This third experiment was an extension course of 08 hours long split into four modules with 2 hours long each.

We proposed five practical programming exercises, solved individually by the students. Each exercise requested the application of the content taught in the respective module, allowing the evaluation of different concepts.

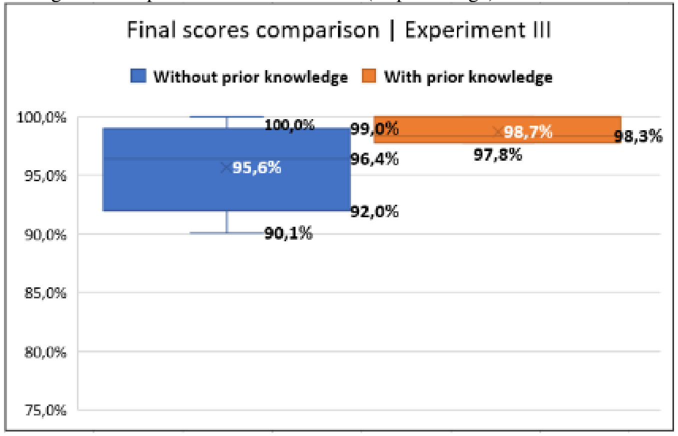
Figure 6 shows the dispersion of the averages obtained by students at the end of the course.

The dispersion in the group without prior knowledge is greater than the group with knowledge, which was expected due to the larger number of students without prior knowledge. Despite the higher dispersion, we observed that the achievement scores were above 90% for students without previous knowledge.

The hypothesis test of Wilcoxon for independent samples with non-normal distribution performed in these samples showed that, with 95% of significance, there is no statistically significant difference in the groups.

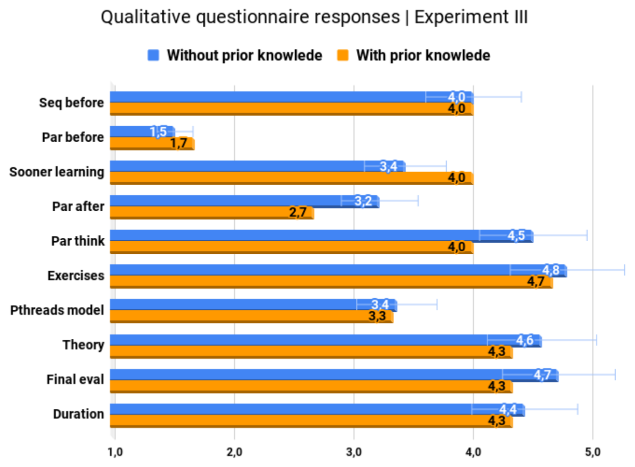
A qualitative questionnaire evaluated the participants' opinions about the course. The responses showed that participants felt stimulated to think in parallel solutions through

Fig. 6. Comparison of course scores (in percentage) - Pthreads/2018.



the resources offered by the course. Besides, they were able to understand the programming model with Pthreads (Figura 7). The answers to the questionnaire are available at http://tiny.cc/pthreads_2018_qualitative.

Fig. 7. Results of the qualitative evaluation for Pthreads/2018 course.



D. *Experiment IV: Introduction to OpenMP only for beginners*

This experiment aimed to verify whether students in the first year of undergraduate computing courses can understand parallelism and develop correct parallel applications using OpenMP. The population of this experiment was 100% formed by computing first-year students.

This experiment had a more extensive time load (18h), divided into 06 modules with 3h each. The objective here was to give more time to the students to solve exercises and fix the knowledge.

We performed pre and post-tests to compare the evolution of students' knowledge in parallel programming with OpenMP (see Figure 8). The post-test shows the growth in knowledge of parallel programming for all students. Some students with

0.0% of success on the pre-test were able to get 100.0% on the post-test, confirming that it was possible to introduce parallel thinking to students without prior knowledge in computing.

Fig. 8. Pre and post testing results for OpenMP/2019 course, taking into account questions about paralelism theory.

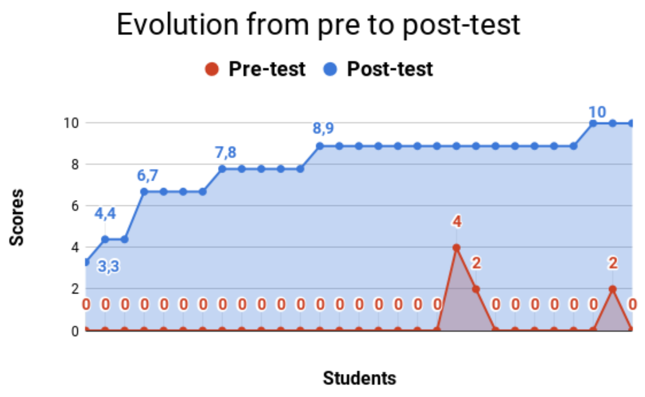
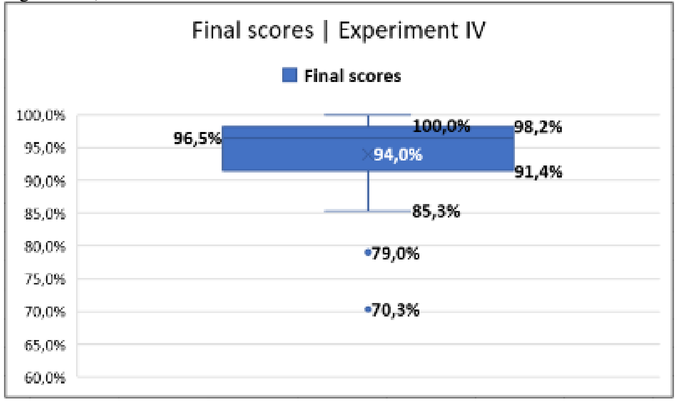


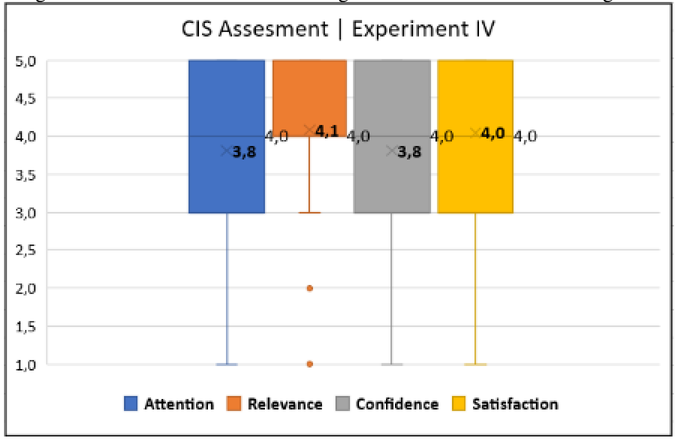
Figure 9 shows the distribution of final averages considering all exercises implemented by students. We observed that the final average was 9.4 (on a scale from 0 to 10). The experiment presented final averages between 8.5 and 10.0 if we disregard the two outliers lower than 8.5. Given the frequency of grades close to the median of 9.4 we found that the students in this experiment understood and used correctly the topics of parallelism taught.

Fig. 9. Final average scores from students of OpemMP/2019 course (parallel algorithms).



We assessed students' satisfaction and motivation regarding the course through the Course Interest Survey (CIS). The CIS contains 34 questions based on the ARCS model, which determines the grouping of these questions into the following categories: attention, relevance, confidence, and satisfaction [28]. Figure 10 shows the distribution of grades given by students in these categories. Grades in all categories show a median close to 4.0, on a scale up to 5.0. This result shows that at least 50% of the grades given were between 4.0 and 5.0. We observed that at least 75% of the scores were above 3.0. These results show that the students have interest, believe in the relevance of the content, feel confident, and have had great satisfaction in learning the parallel programming.

Fig. 10. Distribution of scores taking into account motivation categories.



E. Experiment V: Introduction to OpenMP using PBL approach

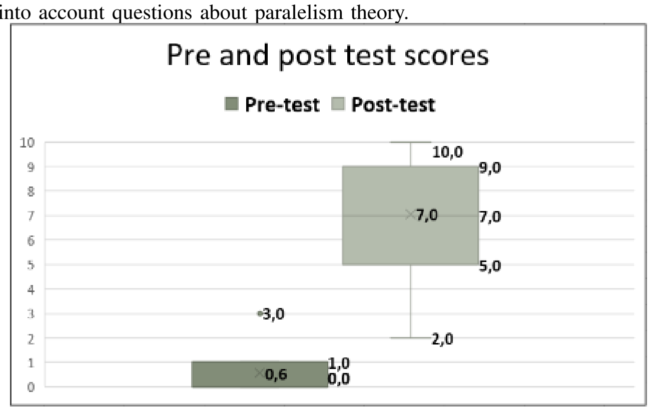
Experiment V considered teaching OpenMP again, but using now an active teaching methodology: *Problem Based Learning* (PBL). Another researcher from our University planned and conducted this course. The objective of this exchange was to verify if the presence of the same researcher was influencing the results. In this experiment, there was the participation of 23 students without and ten students with previous knowledge in computing.

We aimed to investigate how to bring the parallel programming teaching forward when using PBL methodology with parallel programming challenges. This evaluation allowed us to show that it is possible to bring the teaching forward independently of the teaching methodology employed. We structured the course at 12 hours long, divided into four modules of three hours long each one.

In this experiment, we also applied pre and post-tests containing questions about parallelism and OpenMP. Figure 11 shows the results of the students' responses to these two assessments. Significantly higher post-test averages are perceived when compared to the pre-test, showing that there was learning about the theoretical content taught.

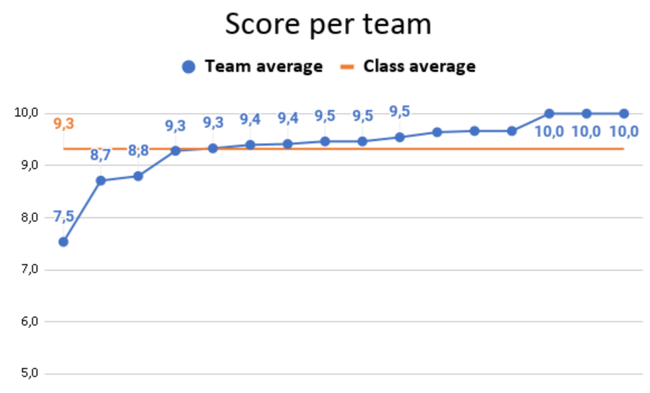
We applied nine challenges to students during the course. The challenges solved by students in each module could apply the concepts and resources taught in that respective teaching module. This approach allowed us to increase the scope of the assessments. In addition to these challenges, we applied a parallel programming marathon in the last module of the course with a further 07 challenges, totaling 16 challenges in all. The students were grouped in development teams for the solution of the challenges, allowing us to insert a collaborative view of this PBL-based approach. Figure 12 shows the average of the evaluations made on the 16 challenges proposed per

Fig. 11. Pre and post testing results for OpenMP-PBL/2019 course, taking into account questions about paralelism theory.



team, considering scores on a scale between o and 10. The results show that 81.2% of the teams scored above the class average of 9.3. Only 03 teams (18.7%) were below 9.0.

Fig. 12. Final average scores from students of OpemMP/2019 course, taking into account parallel challenges.



We evaluated the qualitative aspects of the course again through the 34 questions proposed by Course Interest Survey (CIS), these based on the ARCS model, which determines the grouping of these questions into the following categories: attention, relevance, confidence, and satisfaction [28]. Figure 13 shows the distribution of grades assigned to students by category. Grades in all categories show a median close to 4.0 (0.0 to 5.0). At least 50% of the grades given are between 4.0 and 5.0, and there is at least 75% of grades above 3.0. The results show that we had interested students, who believe in the relevance of the content, feel confident, and had great satisfaction in learning parallel programming.

V. Analysis of Results

We evaluated 252 students in the five experiments conducted in this research, of which 164 already had previous knowledge in computing, and 88 students had no previous knowledge.

Table I presents the score percentages of the parallel algorithms developed by students in all experiments, separating students with and without previous knowledge.

Fig. 13. Qualitative evaluation in the Likert scale - OpenMP-PBL/2019 course.

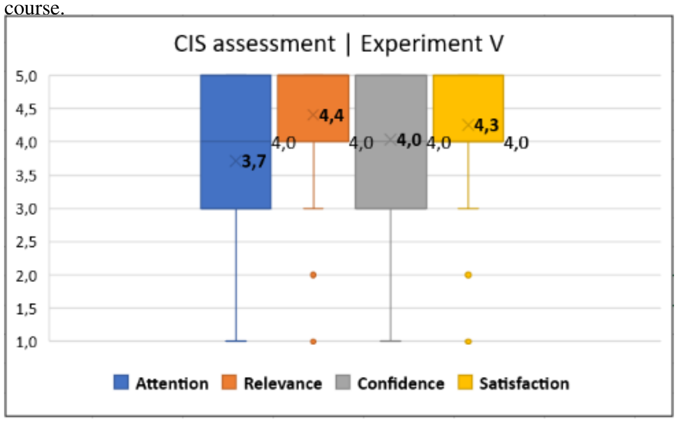


TABLE I
Final scores (in percentage) taking into account the parallel algorithms of all experiments.

		Scores (%)	
Formats	**Experiments**	**With knowledge**	**Without knowledge**
Disciplines	(Exp I) Parallel Programming Computer Science Class (1)	83.0%	-
	(Exp I) Parallel Programming Computer Science Class (2)	79.8%	-
	(Exp I) Parallel Programming Computer Engineering Class	84.5%	-
	(Exp I) Parallel Programming Graduate Class	80.0%	-
Extension Course	(Exp II) Extension Course of OpenMP/2018	93.8%	88.9%
	(Exp III) Extension Course of Pthreads/2018	98.8%	94.6%
	(Exp IV) Extension Course of OpenMP/2019 (only comp first-year students)	-	94.1%
	(Exp V) Extension Course of OpenMP-PBL/2019	95.4%	95.8%
	Average of all experiments:	**87.9%**	**93.3%**
	Average of experiments II, III e V:	**96.0%**	**93.1%**

We apply the Wilcoxon test for independent samples with non-normal distribution to check whether the samples are statistically equivalent or have significant differences. The result of the statistical test revealed a P=0 and, at a significance of 95%, we can state that these results are statistically different, emphasizing that, considering all experiments, beginner students obtained a higher average score.

The grades of experiments II, III, and V, where both groups of students (with and without previous knowledge) participated in the same classes, show that students with previous knowledge obtained an average of 96.0%, while students without previous knowledge obtained an average of 93.1%. Wilcoxon's test for these samples shows P = 0.383. Therefore, at a significance of 95 percent, we can state that these results are statistically equivalent.

Experiment IV included only students from the first semester of computing courses. In this case, the students were able to learn parallel programming while understanding the basic concepts of sequential programming. The students developed codes that make use of threads executed in parallel and measured the execution performance obtained in their programs, although they did not know the operational and technical details of hardware, memory, performance evaluation, or operating systems. Students who have participated in this experiment will be able, from the beginning of their course, to think of parallel solutions and explore in practice the performance obtained by their application, unlike regular students who will probably obtain the first practical contact with parallelism only in other disciplines arranged in more advanced years of graduation. The average scores percentage in experiment IV is higher compared to the averages of the four regular classes (Exp I) that occur in the third year of graduation. This result, together with the comparisons made in the other experiments, shows that students without previous knowledge in computing can achieve scores in the same proportion or even better than students with previous knowledge.

Experiment V was planned and executed by another member of the research laboratory, independently, even adopting a completely different teaching methodology, such as problem-based learning. Even with another researcher planning and teaching the classes, the results of the students without previous knowledge were considered excellent (above 90%).

Analyzing the qualitative evaluation carried out in the courses, we found that the participants did not comment about problems when bringing the parallel programming teaching forward and, in general, were receptive to learning. This result is essential, especially considering students without previous computer education, showing that parallel programming teaching can occur regardless of the obstacles imposed by the limited computing experience. The analysis of motivation according to the ARCS model only with first-year students in Experiment IV intensifies the conclusion that parallel programming content can be brought forward for students in this context and provide excellent learning outcomes without restrictions from students.

VI. Threats

Different evaluators rated students' parallel algorithms and this could lead to subjectivity due to different judgments of each evaluator. In order to reduce the impact of this threat on assessments, we standardized the assessment criteria that allowed for a more uniform comparison.

Moreover, the samples, although composed of a considerable number of students, may not be representative of the population of all computer science beginners. Therefore, it would be desirable to replicate our experiments in other institutions to ensure a higher representation of the sample.

Finally, our experiments were successful in demonstrating that students without prior computing knowledge can learn parallel programming. However, we did not analyze, in a more profound way, the impact of different methodologies and educational resources for this learning. It is crucial to investigate such possibilities.

VII. Concluding Remarks

We evaluated the parallel programming teaching by conducting 05 experiments in 08 different classes, with different teaching methodologies, and for different topics. We taught 252 students in total, 164 of them with previous knowledge in computing, and 88 without previous knowledge. We proposed teaching the content of parallel programming in layers of concepts, allowing individual and objective study at different levels. We identified which subjects at these levels should be taught to students first.

Our results show that it is possible to teach parallel programming from the first year of undergraduate computing courses, preceding the teaching of formal concepts of parallelism. Such an approach stimulates the development of better performing computing solutions and, at the same time, show that algorithms can present better performance depending on how they are coded. According to our results, students without previous knowledge in computing concepts can learn parallelism concepts and develop programs that make use of the performance of parallel execution, even if they are from different areas of study. The results show that parallel programming teaching to first-year students in computing or other courses can have an excellent performance, analogous to that observed by more mature students in computer science courses.

We also present the use of Team-Based Learning (TBL) for parallel programming teaching. We did not find other papers describing the use of TBL for parallel programming teaching in the available literature.

Our results show that students do not require prior computer skills to learn basic parallel programming concepts. They also show that early teaching of parallel programming is a challenge that can be used as a motivating aspect for students in the first year of their computing degree.

Based on the contributions of this work, we suggest some future studies:

- To investigate the differences in parallel programming teaching using distinct teaching methodologies, mainly active teaching methodologies.
- To investigate real industry problems and transcribe them into practical activities to be used as exercises in teaching parallel programming, aiming to stimulate the use of parallelism in problems obtained outside the academy.
- Study what is the impact throughout the undergraduate course of learning parallel programming from the beginning, and what will be the effect of this learning on other later subjects.

References

[1] Álvaro Fernández, C. Fernández, J. Ángel Miguel-Dávila, M. Ángel Conde, and V. Matellán, "Supercomputers to improve the performance in higher education: A review of the literature," *Computers and Education*, vol. 128, pp. 353 – 364, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131518302756

[2] D. Ernst, B. Wittman, B. Harvey, T. Murphy, and M. Wrinn, "Preparing students for ubiquitous parallelism," *ACM SIGCSE Bulletin*, vol. 41, p. 136, 03 2009.

[3] Y. Bi and J. Beidler, "A visual tool for teaching multithreading in java," *Journal of Computing Sciences in Colleges*, vol. 22, pp. 156–163, 06 2007.

[4] R. Muresano, D. Rexachs, and E. Luque, "Learning parallel programming: a challenge for university students," *Procedia Computer Science*, vol. 1, no. 1, pp. 875 – 883, 2010, iCCS 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050910000979

[5] K. McMaster, B. Rague, and N. Anderson, "Integrating mathematical thinking, abstract thinking, and computational thinking," in *2010 IEEE Frontiers in Education Conference (FIE)*, Oct 2010, pp. S3G–1–S3G–6.

[6] B. Rague, "Measuring cs1 perceptions of parallelism," in *2011 Frontiers in Education Conference (FIE)*, Oct 2011, pp. S3E–1–S3E–6.

[7] S. Ghafoor, D. Brown, and M. Rogers, "Integrating parallel computing in introductory programming classes: An experience and lessons learned," 08 2017.

[8] D. J. John, "Nsf supported projects: Parallel computation as an integrated component in the undergraduate curriculum in computer science," *SIGCSE Bull.*, vol. 26, no. 1, p. 357–361, Mar. 1994. [Online]. Available: https://doi.org/10.1145/191033.191167

[9] Y. Ko, B. Burgstaller, and B. Scholz, "Parallel from the beginning: The case for multicore programming in the computer science undergraduate curriculum," 03 2013, pp. 415–420.

[10] ACM/IEEE-CS Joint Task Force on Computing Curricula, "Computer science curricula 2013," New York, NY, USA, December 2013. [Online]. Available: http://dx.doi.org/10.1145/2534860

[11] S. K. Prasad, A. Chtchelkanova, A. Gupta, A. Rosenberg, and A. Sussman, "Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing: Core topics for undergraduates," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: ACM, 2012. [Online]. Available: https://grid.cs.gsu.edu/~tcpp/curriculum/

[12] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[13] T. R. Gross, "Breadth in depth: A 1st year introduction to parallel programming," in *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, pp. 435–440. [Online]. Available: http://doi.acm.org/10.1145/1953163.1953291

[14] M. E. Acacio, J. Cuenca, L. Fernández, R. Fernández-Pascual, J. Cervera, D. Giménez, M. C. Garrido, J. A. S. Laguna, J. Guillén, J. A. P. Benito, and M.-E. Requena, "An experience of early initiation to parallelism in the computing engineering degree at the University of Murcia, Spain," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012*, 2012, pp. 1289–1294.

[15] I. Skopin, "Early learning in parallel programming," pp. 219–229, 01 2014.

[16] R. Feldhausen, S. Bell, and D. Andresen, "Minimum time, maximum effect: Introducing parallel computing in cs0 and stem outreach activities using scratch," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014, p. 75.

[17] J. C. Adams, "Injecting parallel computing into cs2," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: ACM, 2014, pp. 277–282. [Online]. Available: http://doi.acm.org/10.1145/2538862.2538883

[18] R. Sakellariou, "Experiences with teaching a second year distributed computing course," 05 2016, pp. 28–37.

[19] D. Valentine, "HPC/PDC Immunization in the Introductory Computer Science Sequence," in *Proceedings of the Workshop on Education for High-Performance Computing*, ser. EduHPC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 9–14.

[20] T. Newhall, A. Danner, and K. Webb, "Pervasive parallel and distributed computing in a liberal arts college curriculum," *Journal of Parallel and Distributed Computing*, vol. 105, 01 2017.

[21] S. Imam and V. Sarkar, "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*, ser. PPPJ '14. New York, NY, USA: ACM, 2014, pp. 75–86.

[22] C. Libert and W. Vanhoof, "Survey of software visualization systems to teach message-passing concurrency in secondary school," 05 2017, pp. 386–397.

[23] S. Torbert, U. Vishkin, R. Tzur, and D. J. Ellison, "Is Teaching Parallel Algorithmic Thinking to High School Students Possible?: One Teacher's Experience," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: ACM, 2010, pp. 290–294.

[24] A. S. Tanenbaum and J. R. Goodman, *Structured Computer Organization*, 4th ed. USA: Prentice Hall PTR, 1998.

[25] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. USA: Prentice Hall Press, 2014.

[26] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. USA: Prentice Hall Press, 2010.

[27] D. Parmelee, L. K. Michaelsen, S. Cook, and P. D. Hudes, "Team-based learning: a practical guide: Amee guide no. 65," *Medical teacher*, vol. 34, no. 5, pp. e275–e287, 2012.

[28] J. Keller, *Motivational Design for Learning and Performance: The ARCS Model Approach*, 01 2010.