Learning Parallel Programming Through Programming Challenges

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

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

Davi Jose Conte
Institute of Mathematics and Computer Sciences
University of São Paulo
São Carlos, Brazil
daviconte@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 the use of challenges to teach parallel programming, regardless of teaching methodology (traditional, Problem-Based Learning and others) or programming-contest support systems. We verify how challenges contribute to the learning of parallel programming, considering technical and motivational aspects. Studies demonstrate the potential of using programming contests as educational resources at the end of courses, encouraging students to practice and learn more about parallel programming. However, there are no proposals exploring programming challenges as part of the learning process, since the beginning of the course. This paper presents four different experiments applying challenges to teach parallel programming for undergraduate and graduate students. We evaluated different teaching/learning methodologies, contexts, educational resources (including programming marathon systems), and groups of students. Considering all the experiments, almost 250 students participated and were evaluated. We developed and applied 58 challenges to analyze theoretical and practical knowledge about parallel programming. Different metrics were used to evaluate the students during the experiments: theoretical assessments, source-code quality, program output correctness, and student´s receptivity/motivation to learning from challenges. Results from experiments show scores of up to 96% of learning on technical aspects and up to 83% on student satisfaction. Challenges stimulated the development of high-quality parallel solutions and promoted a healthy environment among students, where the lightly-competitive context contributed to create a collaborative atmosphere. The use of programming marathon systems as an educational tool is not imperative. The use of challenges for the students' learning, not on their classification, showed that it is possible to teach parallel programming, maintaining students focused and motivated.**

***Index Terms*—Teaching, Learning, Parallel Programming, Challenges, Contests.**

I. Introduction

The importance of parallel software has increased considerably due to the popularization of architectures such as multi and many core [1].

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

However, the teaching of parallel programming has not kept pace with this evolution and still has unresolved problems such as the need for parallel hardware in practical classes, the difficulty students have in using tools that allow them to visualize the behavior of parallel programs, and the need for prerequisites of basic computing concepts for the development of parallel programs. Some authors even state that the previous sequential programming teaching can make more difficult further learning of parallel programming [2].

The main prerequisites for a more comprehensive teaching of parallel programming are programming and data structures, analysis and complexity of algorithms, performance evaluation, computer organization and architecture, operating systems and computer networks [3]. Unfortunately, apart from these problems, there are no well-established methodologies for teaching parallel programming that is effective from the learning point of view and, at the same time, awaken students' interest in the area [3], [4].

These difficulties and the growing demand for professionals with skills and competences in the development of parallel programs justify the need for new teaching methods and resources capable of facilitating learning, following the demand for training professionals in this area [3].

The Association for Computing Machinery - ACM [5] and the Institute of Electrical and Electronics Engineers - IEEE [6] act in this perspective, proposing curricular guidelines for teaching parallel programming. The ACM and IEEE curricular guidelines include concepts to be addressed and suggestions for teaching methods consisting mainly of hands-on activities that illustrate the application of theoretical content.

Traditional pedagogical approaches are commonly used in teaching parallel programming [7]. However, new proposals are being developed in recent years to foster student learning and motivation. These proposals include Problem-Based Learning - PBL ([8]–[16]); programming by standards ([17]–[24]); and project based learning ([25], [26]). These techniques are seen as promising for parallel programming learning because they consider the practical teaching (hands-on) of parallel programming activities, encouraging the development of higher quality solutions [27].

Other studies point to the use of programming challenges as educational resources that can encourage students to develop quality, high performance solutions [10], [28]. Such studies are very dependent on the use of marathon programming systems, which limits the applicability of the methodologies and requires additional effort from teachers to prepare courses and maintain the necessary computer infrastructure.

The hypothesis investigated in this research project is that if parallel programming can be learned using programming challenges. In this scenario, the resolution of the challenges includes the development of correct, appropriately designed parallel programs with the expected performance. Students are expected to be motivated to learn through an aggregated, results-focused and at the same time challenging approach.

The main objective of this work is to evaluate the use of programming challenges in the learning of parallel programming in an orthogonal way to other factors such as teaching methodologies, programming models, execution platform and number of students in classrooms.

To achieve the proposed objective we executed four experiments with different students of parallel programming subjects. Our main results from experiments show high scores of both the learning of technical aspects and the student satisfaction. Challenges stimulated the development of high-quality parallel solutions and promoted a healthy environment among students, where the lightly-competitive context contributed to create a collaborative atmosphere.

This paper has the following organization: Section II presents the related works, their main contributions, characteristics and limitations. Section III describes the scientific methodology used in this work, the materials and methods used to obtain the results and their subsequent analysis. Section IV details the four experiments carried out and presents their results. Section V presents an summary of the results obtained, analysis and comparison between the experiments. Section VI presents the final conclusions, contributions, limitations and future works regarding learning parallel programming trough programming challenges.

II. Related Work

This section presents papers available in the literature that are related to this paper.

The publication of [29] describes the use of a hybrid methodology of Competition Based Learning - CBL and Problem-Based Learning in a business capstone course. The paper shows that the combination of competition and collaboration in a dynamic environment fosters student motivation, quality of solutions and provides a context close to market reality. Although it is not a computer-related work, the methodology provides a guide for building pedagogical environments based on challenges and healthy competition.

The paper of [30] describes a framework for the implementation of Competition-Based Learning together with other pedagogical methodologies, based on game theory, in the context of teaching sequential programming. The author addresses different problems, application cases and makes a quantitative and qualitative analysis of students' performance over five years. This paper does not contemplate parallel programming disciplines, but mainly the teaching using game theory.

The work of [28] details the use of the programming challenges of the Spanish programming contest as an educational resource used to support the teaching of the parallel programming subject. The paper focuses on the process of adapting the tool *Mooshak* for the use as an educational resource [28]. This paper is the first report available in the literature to address the use of programming challenges for teaching parallel programming. The partial results of [28] are essential information for the development of our work. However, it is a short paper describing a work in progress, therefore it is difficult to reproduce its results.

The paper from [25] presents a methodology for teaching parallel programming based on solving specific computer graphics problems. The approach is a project-oriented model and encourages students to design high-performance graphics systems. The union of the concepts of these two areas fostered practical learning, motivating students during the course. Although describing a promising teaching methodology, this work deals with the application of parallel programming in the context of computer graphics. The use of Project Based Learning requires longer and more complex projects when compared to parallel programming challenges applied in a class.

The work of [31] proposes the gamification of the parallel programming on heterogeneous multiprocessors teaching. The paper proposes an evaluation in game challenges, in a similar way to electronic game competitions. There is a high level of student satisfaction with the proposed methodology. Different from our work, this paper addresses a whole methodology for learning parallel programming with a strong dependence on gamification.

The paper presented in [10] details a practical teaching method for parallel programming based on problems of the Spanish marathon parallel programming contest (Spanish Parallel Programming Contest). The paper presents lesson plans and associated teaching resources. The results show the increase in student satisfaction and approval in the subject. This work has a distinct proposal when compared to our paper, since it focuses on the Mooshak system and parallel programming marathons as an evaluation tool.

In a general way, these works differ from ours as they present specific case studies for a proposed methodology or tool. The research described in this paper does not propose teaching methodologies or tools. We analyze the use of challenges in teaching parallel programming, regardless of the tools to support the programming marathons, teaching methodologies, number of students, platforms or other factors.

III. Methodology

We carried out four different experiments, with parallel programming content, target audience, infrastructure and number of students as factors. The experiments followed the methodology proposed by [32]. The contents taught to students were planned based on the curricular structure proposed by ACM and IEEE [3], [5]. All four experiments took place at the University of São Paulo in São Carlos, Brazil.

We have designed 58 parallel programming challenges for the experiments, based on previous resources developed by the authors themselves in previous editions of the parallel programming subject. The challenges contain descriptions of the problem, source code and test cases, and are available, in portuguese, at http://tiny.cc/prog_challenges.

The challenges used in experiments II and IV have theoretical content associated with the problem, being such challenges used in classes with the Problem-Based Learning methodology [33]. This material is available, in portuguese, at http://tiny.cc/cadernos_de_desafios.

The performance of the participants of these experiments took into account **quantitative evaluation of theoretical knowledge, analysis of the correctness of the algorithms and quality of the source codes developed, in addition to a qualitative assessment of student satisfaction with the pedagogical method applied, and content taught**. We defined a weighted and standardized evaluation for the students who participated in the experiments. The standardization allowed us to compare the results of the experiments. The metrics used in the evaluations and their weights are:

- **Correct parallelism**: analyzes the design of the parallel algorithm, i.e., the identification of parallel aspects of the problem/application and adequacy to the parallel architecture. It is equivalent to 40% of the final weighted average grade.
- **The correct use of the programming model**: verifies the appropriate use of the programming tools in the development of the parallel algorithm, such as languages, libraries, primitives, directives, arguments and others. It is equivalent to 30% of the final weighted average grade.
- **Correct/expected output**: analyzes whether the parallel program output is correct and also the performance of the solution. Equals 20% of the final weighted average grade.
- **Legibility and good programming practices**: checks how readable the parallel code is, including practices like indentation, use of comments and others. It represents 10 percent of the final weighted average grade.

The analysis of the results used Shapiro-Wilk test to confirm the normality of the data, with a significance level of 95 %; the Wilcoxon test for paired samples, also with 95 % significance, in order to evaluate the equivalence (*h0*) or not (*h1*) of the means obtained in each test and the Wilcoxon test, for samples of varied and unpaired size (Mann-Whitney's U test) [34]. The qualitative evaluations, with the exception of experiment I, were based on the CIS instrument (Course Interest Survey), which has 34 questions distributed in four categories: Attention (A), Relevance (R), Confidence (C) and Satisfaction (S) [35].

IV. Experiments

The four experiments involved 235 students in classes of different sizes: from 8 to 151 students. The objective of these configurations is to analyze the behavior of teaching oriented to parallel programming challenges in both large and small classes.

The four experiments performed are described below:

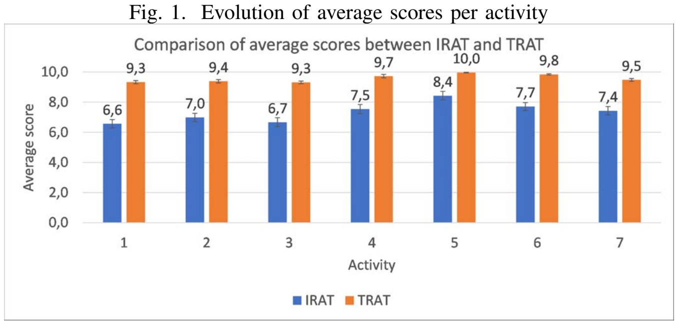
- Experiment I: Aimed at verifying knowledge retention, code quality and student satisfaction for the OpenMP, MPI and CUDA programming models, using the Team Based Learning (TBL) [36] methodology in the Concurrent Programming subject for Computer Engineering and Bachelor of Computer Science courses in 2018. The experiment had the participation of 151 students, divided in 3 classes. It was 45 hours course, divided into 15 weekly classes. We used the PC^2 [37] programming marathon system.
- Experiment II: Aimed at verifying knowledge retention, code quality and student satisfaction for the content of the OpenMP programming model, using the traditional teaching methodology in a 2019 course open to the community. The focus of this course was to teach the *OpenMP* programming model for freshman students of any computing course. It was a 18 hours course, divided into 6 classes and had the participation of 29 students. The challenges applied to the students were presented as a programming marathon held in the last class, with the support of the Mooshak [38] programming marathon system;
- Experiment III: Aimed at verifying the retention of knowledge, code quality and student satisfaction for the content of the OpenMP programming model, using the Problem-Based Learning methodology in a course open to the community in 2019. It was a 12 hours OpenMP course divided into 04 classes and had the participation of 47 students, from 15 different undergraduate or graduate courses. The programming challenges were supported by the Mooshak system, in all classes. The last class was a parallel programming marathon including awards for the three best ranked teams;
- Experiment IV: Aimed at verifying knowledge retention, code quality and student satisfaction for the OpenMP, MPI and CUDA programming models, using Problem-Based Learning in a High Performance Computing subject of Computer Engineering course, in 2019. This experiment had the participation of 8 students. The course had 60 hours divided into 30 weekly classes. The parallel programming challenges were supported in all classes by the Mooshak system.

A. *Experiment I: Parallel Programming Course using TBL*

As part of the TBL methodology, all classes of this experiment were composed of prior study, individual evaluation, team evaluation and practical application, the latter being a parallel programming challenge with the support of the PC^2 tool. This system was used as an automatic judge, helping in the correction of the submitted exercises and providing instant feedback on the correctness of the codes submitted.

Figure 1 shows the average student score for each activity.

Fig. 1. Evolution of average scores per activity



It is possible to observe an average class score of 7.3 in the Individual Readiness Assurance Test (IRAT) and 9.6 in the Team Readiness Assurance Test (TRAT), demonstrating a significant evolution in the students' theoretical knowledge absorption in the TBL cycles. The error bars on the graph represent the normal 95 percent confidence interval, which are replicated in the other bar graphs.

The challenge proposed in the final stage of each TBL class was a hands-on parallel programming activity. These activities had a 10-point score each, assigned according to the criteria established in Section III. Figure 2 shows the averages of code quality scores by activity.

Fig. 2. Average code quality scores by practical activity

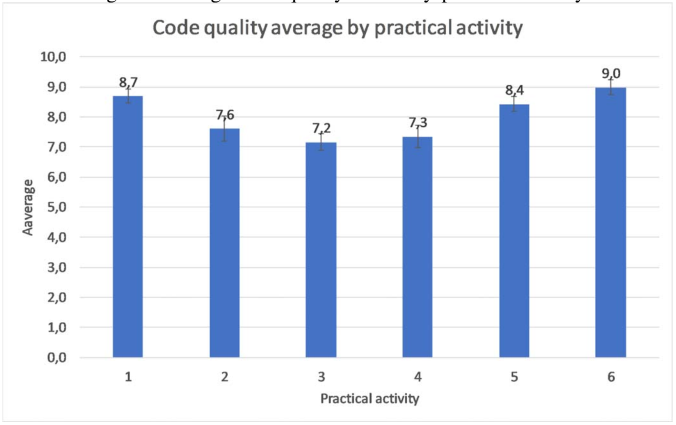


Figure 3 shows the average scores in code quality per programming model.

At the end of the course, students answered an anonymous, optional online questionnaire. There were 100 responses, corresponding to 66% of the participants. The questions and answers, written in portuguese, can be found at http://tiny.cc/exp1_qualitativo.

Fig. 3. Code quality averages per programming model

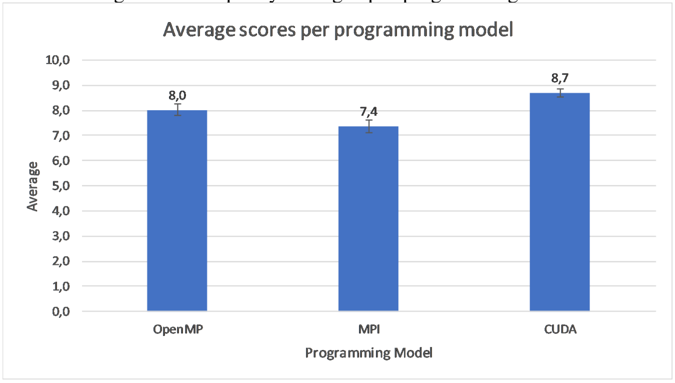
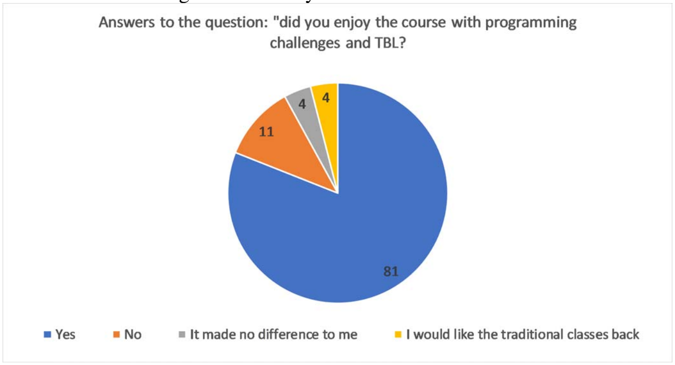


Figure 4 summarizes student satisfaction with the course, methodology and challenges used. It is possible to see that 81% of the students respond positively about the new methodology.

Fig. 4. Summary of student satisfaction



B. *Experiment II: Traditional Teaching for an Introduction to OpenMP*

This course was open to the community and was supported by challenges that were implemented in sequential and parallel versions. The *Mooshak* system used in this course was installed in a virtual machine of *Google Cloud* and the students could access it through a web interface.

On the last day of the course 3-member teams were set up, and they received the challenge descriptions. The professor and two master students were in the laboratory to answer students´ questions and monitor the system. During most of the class, the marathon scoreboard was displayed, and the teams could consult their scores, which is also available through the Mooshak system itself.

At the end of the class, the quality of the codes were evaluated manually. The three teams with the best scores received prizes for their achievement.

A pre-test was applied on the first class and a post-test on the last class in order to evaluate the absorption of theoretical knowledge by the participants of this experiment. A total of 29 students took both tests. Figure 5 shows an improvement from 44% in the pre-test correct answers to 82% in the post-test.

Fig. 5. Comparison between pre-test and post-test scores

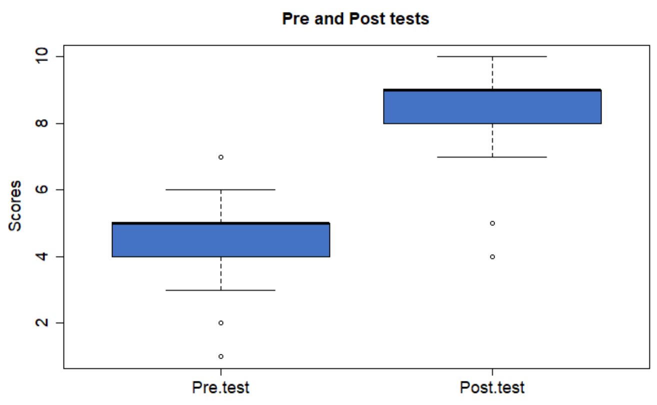
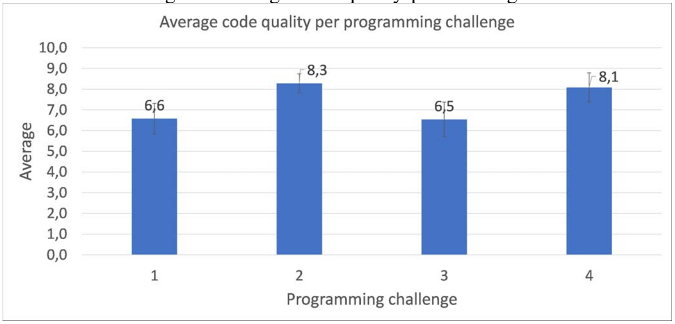


Figure 6 shows the code quality average scores per challenge. In all challenges the average quality score obtained was higher than 65% of the valid score for this criterion.

Fig. 6. Average code quality per challenge



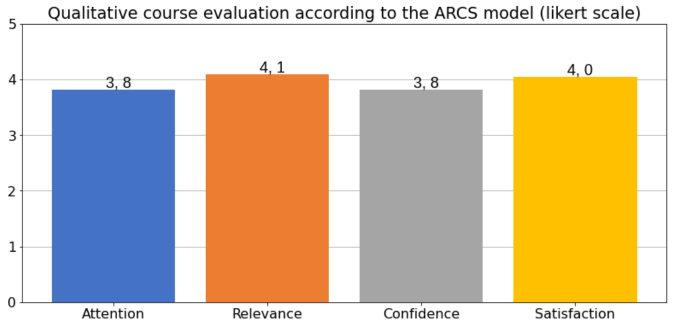
A form evaluated the satisfaction of students in this experiment. Figure 7 shows the average scores, for each of the ARCS model categories [35]. It is possible to notice that the average degree of motivation for this experiment was 72.5%. The graphs illustrating the answers to the 34 questions in the qualitative assessment can be accessed http://tiny.cc/omp1_qq.

C. Experiment III: Introduction to OpenMP using PBL and Parallel Programming Contest

This course was also open to the community and was attended by students from different areas, all interested in parallel programming. As it was not offered only for freshman students, in this course we had a large number of students compared with the course offered for Experiment II.

As in Experiment II, the Mooshak 2 (installed on a virtual machine on *Google cloud*) was used to run the challenges in all classes and the parallel programming marathon in the last class. The students were grouped together in the first class and kept the same until the last class.

Fig. 7. Evaluation of the course in the likert scale, by category of the ARCS model

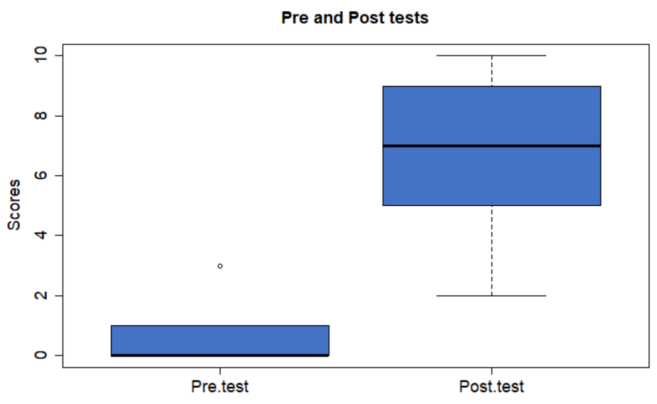


The students of this course were given 16 challenges arranged in notebooks, handed out in print and available online at the beginning of each class. These notebooks contained the description of the parallel programming challenges to implement, basic theoretical material for the solution of the challenge and a sequential implementation.

In every class, the teachers conducted a quick presentation of the content presented in the challenge notebook, and the teams started the solution of the challenges. A datashow was used to display the teams' scores, while the teachers answered students' questions and checked the *Mooshak* system.

Figure 8 illustrates the difference between the average scores obtained by the students in the pre-test and the post-test. It is possible to observe that the evolution of students' performance in the post-test was significant compared to the scores obtained in the pre-test.

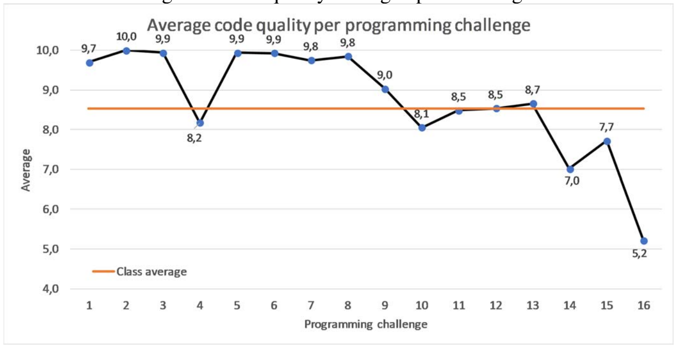
Fig. 8. Comparison between pre-test and post-test scores



A total of 214 submissions were expected, calculated from the product among the number of teams and the challenges given each class. 158 (74%) submissions were submitted, demonstrating that most students in a non-compulsory course committed to solving the proposed challenges.

In addition, the evaluation of the submissions according to their correctness and quality was carried out. The average score considering the quality criteria of code submitted was 85%. Figure 9 illustrates this averages per challenge.

Fig. 9. Code quality averages per challenge

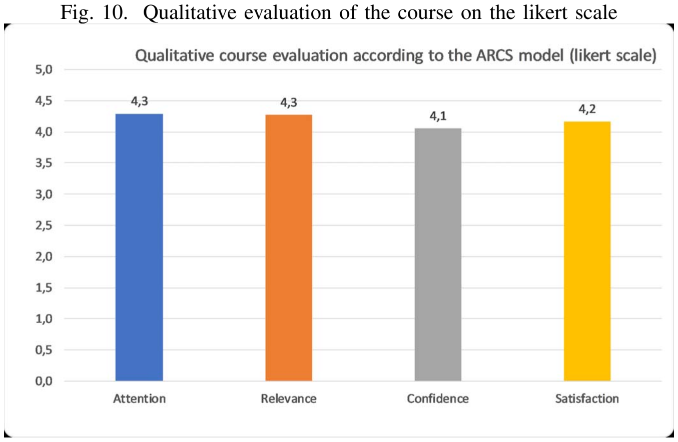


It is possible to observe in Figure 9, that, on average, students performed very well in almost all of the proposed challenges, where in only 5 challenges the average was less than 85 percent. As expected, the scores obtained were proportional to the level of complexity of the proposed challenges. The quality scores of the code in the last challenges were lower mainly because of two factors: these challenges were more complex and in this class there were more challenges, reducing the available time to develop each code.

In order to check the students´ receptivity in relation to the content, methodology and materials used, a questionnaire was used, adapted from the ARCS model and the proposal of Course Interest Survey [35]. A total of 19 students answered the questionnaire. The graphs illustrating the answers to the 34 questions in the qualitative assessment can be accessed at http://tiny.cc/graficos_qualitativo_open.

Figure 10 summarizes the information obtained from the assessments on the *likert* scale. It can be seen in the graph that the course evaluation was quite positive, with an average satisfaction of 83%.

Fig. 10. Qualitative evaluation of the course on the likert scale



D. Experiment IV: Parallel Programming Course using PBL

This experiment was carried out with students of a regular subject at a Computer Engineering course in 2019. At the beginning of the school semester, the planning of the course was done, according to the content to be taught, categorizing it by programming model: *OpenMP, MPI* and *CUDA*. For each model, a pre-test and post-test were prepared and the programming challenges to be applied were selected.

Depending on these defined programming challenges, a challenge notebook was prepared describing the exercises, their sequential implementation, as well as a synthesis of the theory and pertinent bibliographic references. Before each class, such material was released, through the *Moodle* system, in digital version.

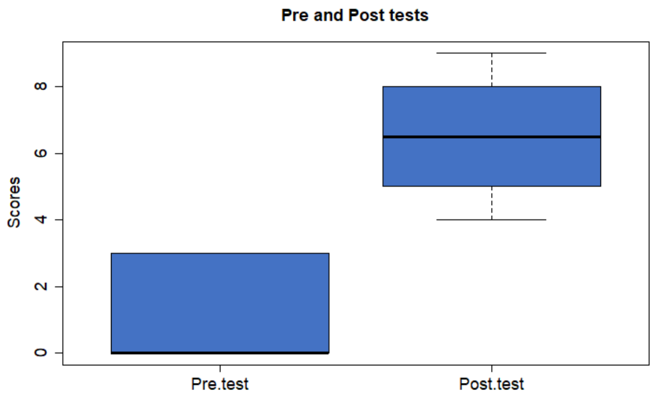
The students were organized in pairs on the first class and their groups were kept the same throughout the school semester. As in Experiments II e III, the *Mooshak* system was installed, configured and deployed in the infrastructure of the *Google cloud*. At each class the description of the challenges and corresponding test cases were inserted in the *Mooshak*.

During the class, the content was covered in the sequence provided in the challenge notebook, although in a more in-depth manner. The last 60 minutes (out of a total of 150 minutes) were allocated for implementing the challenges, and the teacher and student assistant were available to answer students' questions.

The challenge scores of each team in every class was regularly updated, after the manual evaluation of the codes submitted and the scores and derived classification made available through the *Moodle*.

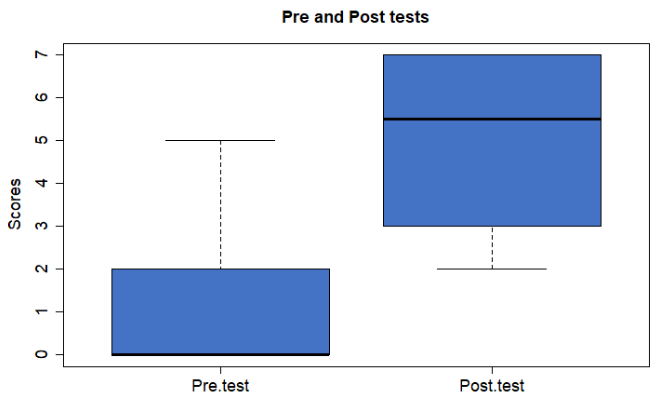
A pre-test was initially applied in the context of the first class concerning the *OpenMP* module, and a post-test in the last class of this module. A total of 6 students take both tests. Figure 11 represents the comparison between the average scores obtained by students on the pre-test and the post-test.

Fig. 11. Comparison between pre-test and post-test scores - OpenMP



Subsequently, a pre-test and a post-test were conducted for the MPI content. Figure 12 shows the comparison between the averages obtained by the class for the content of *MPI*.

Fig. 12. Comparison between pre-test and post-test scores - MPI



For the content of CUDA, a pre-test and post-test was also applied, similar to the other programming models. In this case, Figure 13 contains the comparison between the averages obtained in both tests.

Fig. 13. Comparison between pre-test and post-test scores - CUDA

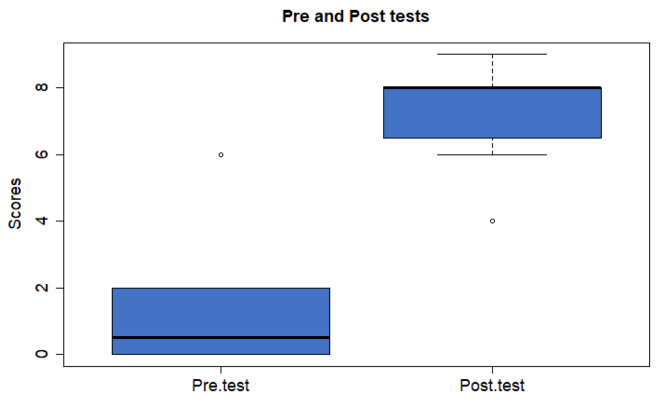


Figure 14 presents an overview of the average quality of code per programming challenge. It can be seen that there was high variability in the scores obtained, although in most challenges (58%) the score was above the average for the class.

At the end of the course, we applied an adapted form from the Course Interest Survey [35]. The questions and answers in this survey are available at http://tiny.cc/teste_qualitativo_cad. All students in the class answered the questionnaire.

Figure 15 summarizes the scores obtained for each of the categories in the *ARCS* model. It is possible to observe that, in general, the students who participated in the experiment were satisfied with it, illustrated by an average pass rate of 74%.

V. Analysis of experiments

In all experiments, the learning of the students in theoretical content has been improved. Thus, it is possible to state that the students were able to learn and absorb the theoretical content referring to the programming models used and other content addressed and, consequently, that the use of programming challenges is an effective instrument as an educational resource allied to the teaching-learning process of parallel programming.

Another result is the quality of the source code referring to the programming challenges. Figure 16 illustrates the average code quality obtained for each experiment. The overall average was 76% of the valid score for code quality.

Fig. 14. Average code quality per programming challenge

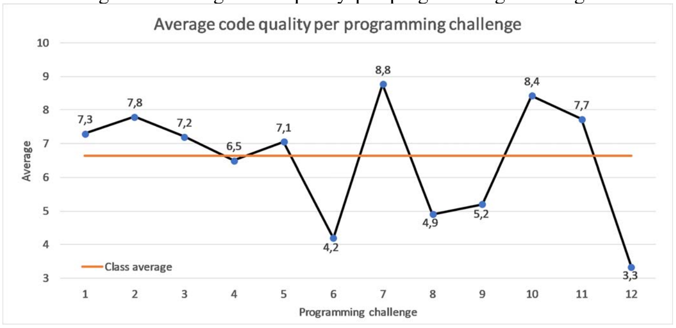


Fig. 15. Assessment of the discipline on the likert scale

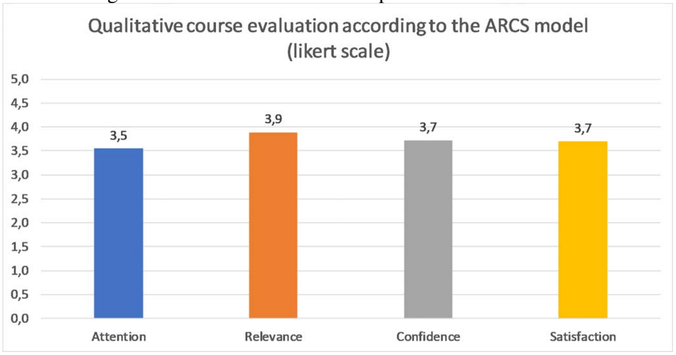
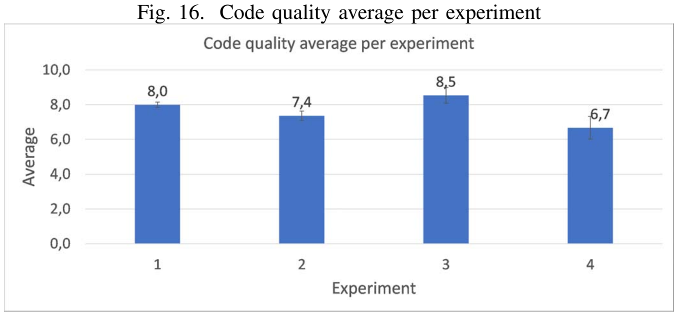


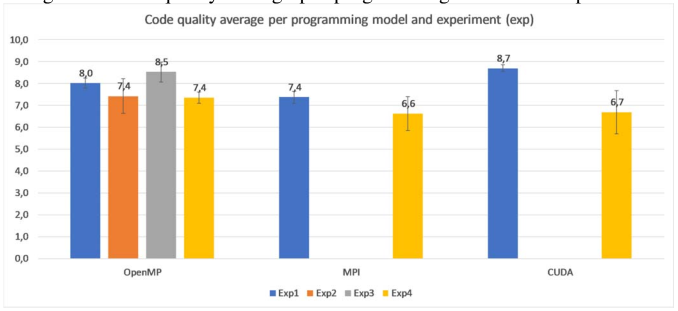
Fig. 16. Code quality average per experiment



The programming challenges solved, on average, were correct and the source codes had quality, considering good use of the structures, good practices and programming models, besides the performance of the solutions developed.

Figure 17 presents the averages obtained by programming models and experiments. The students developed quality codes for all programming models, with emphasis on *OpenMP*, with a percentage of 78% in code quality, followed by *CUDA*, with 77% and, finally, *MPI*, with 70%. The students presented more difficulty in the *MPI* programming model, which was also reflected in the score obtained in quantitative evaluations.

Fig. 17. Code quality average per programming model and experiment



Additionally, it was possible to notice that in all the experiments carried out students felt satisfied and motivated with the methodology, content, materials, educational and didactic resources, since for all the experiments, the average approval of students on a *likert* scale (regardless of the use of the Course Interest Survey) was over 72%.

We have also compiled the final scores obtained from parallel programming disciplines held between 2004 and 2017 in the University of São Paulo. The Figure 18 shows the dispersion of both the scores from previous classes and our experiments. The boxplot shows that the teaching using challenges had 75% of the best scores above 7.4, while the previous classes had 75% of the best scores above 6.4, only. Also, the lowest score above the outliers for the experiments was 5.6, while for the previous disciplines it was 3.3.

We also perform Mann-Whitney's U test comparing these samples. Although the difference between their means was narrow (7.1 and 7.7), the statistical analysis showed that the scores obtained in our experiments were higher.

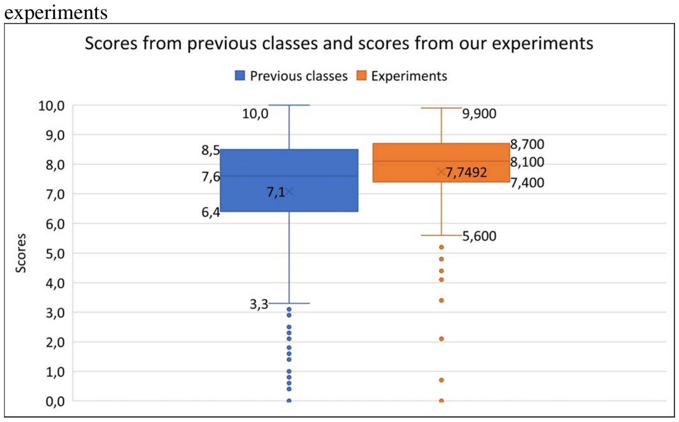
VI. Conclusion

It was possible to notice that the PC^2 system, even though it has more complete documentation than the *Mooshak* system, has lower usability and its configuration is considerably more complex than the latter. The PC^2 customization features are more limited than those offered by *Mooshak*.

The use of both systems was positive, facilitating the evaluation of the source codes developed by the students and allowing the immediate *feedback* on the correctness of the implementations developed. However, the systems present limitations derived from its scarce documentation and its original purpose, since both tools were not designed with the purpose of educational resources.

The systems can be used for teaching, although this requires considerable effort from the teacher, since they need to configure, deploy and monitor the tools. Moreover, this type of *software* is not mandatory for the application of programming challenges, although it offers advantages for this activity.

Fig. 18. Comparison of scores from previous classes and scores from our experiments



In all experiments performed, the performance of participating students was very positive, in all metrics considered, illustrated by average score values in theoretical evaluations of 72.6% and maximum of 96%; average code quality of 76% and maximum of 85%; and average motivation percentage of 76.5%, with a maximum of 83%.

Our results showed that it is possible to learn parallel programming using programming challenges, regardless of the pedagogical method or educational resource used.

Programming challenges constitute an important, effective and flexible educational resource that can be applied in different contexts, both in courses open to the community and in regular undergraduate courses, with students from different periods or courses, as long as they are adapted to the target audience and there is adequate course preparation. Challenges also contribute to the motivation of students, since a healthy competitive environment combined with awards, even if symbolic, stimulate the participation of students and the development of creative solutions.

Finally, the greatest contribution of this work goes to the student, by stimulating a healthy competition environment that allows the student to produce better solutions, even they have already submitted a correct solution. The student understands its evolution, which allows the constant improvement of the solutions developed, and it feels motivated to continue, and to overcome new learning challenges.

Based on our results, we suggest these future studies:

- Analyze different teaching methodologies in addition to those considered in this work;
- Study the impact, usability and functionality of different marathon programming systems in other contexts;
- Implement or adapt a marathon programming system for teaching purposes;
- Study the long-term impact of using challenges in parallel programming disciplines and also in other disciplines.

References

[1] S. Fiore, M. Bakhouya, and W. W. Smari, "On the road to exascale : Advances in High Performance Computing and Simulations — An overview and editorial," *FGCS*, vol. 82, pp. 450–458, 2018.

[2] F. F. de Vega, "To be, or not to be: That is the recursive question," in *2019 IEEE EDUCON*, 2019, pp. 1294–1299.

[3] 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. ACM, 2014.

[4] 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 & Education*, vol. 128, pp. 353 – 364, 2019.

[5] ACM/IEEE-CS Joint Task Force on Computing Curricula, "Computer science curricula 2013," New York, NY, USA, December 2013.

[6] S. K. Prasad, A. Y. Chtchelkanova, S. K. Das, F. Dehne, M. G. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc *et al.*, "Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing: core topics for undergraduates," in *Procedings of SIGCSE*, vol. 11, 2011, pp. 617–618.

[7] J. Mellor-Crummey, W. Gropp, and M. Herlihy, "Teaching parallel programming: a roundtable discussion," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 17, no. 1, pp. 28–30, 2010.

[8] 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-bahía blanca argentina," in *Procedings of IPDPSW*, 2012, pp. 1317–1320.

[9] P. E. Strazdins, "Experiences in teaching a specialty multicore computing course," in *Procedings of IPDPSW*, 2012, pp. 1283–1288.

[10] D. Giménez, "A practical parallel programming course based on problems of the Spanish parallel programming contest," in *Procedia Computer Science*, vol. 80, 2016, pp. 1978–1988.

[11] J. Cuenca and D. Giménez, "A parallel programming course based on an execution time-energy consumption optimization problem," in *Procedings of IPDPSW*, May 2016, pp. 996–1003.

[12] B. White, M. Robinson, C. Mitchell, and J. Mache, "Using the n-body problem to engage undergraduates in parallel programming," in *Proceedings of FECS*. The Steering Committee of The World Congress in Computer Science, 2012, p. 1.

[13] G. Bernabé, J. Cuenca, L.-P. García, D. Giménez, and S. Rivas-Gomez, "A high performance computing course guided by the lu factorization," *Procedia Computer Science*, vol. 29, pp. 1446–1457, 2014.

[14] A. Yazici, A. Mishra, and Z. Karakaya, "Teaching parallel computing concepts using real-life applications," *International Journal of Engineering Education*, vol. 32, no. 2, pp. 772–781, 2016.

[15] I. Zakharova and A. Zakharov, "Key issues of low-level parallel programming in theindividual projects for graduate students," *The Int. J. of Eng. Education*, vol. 34, no. 4, pp. 1250–1260, 2018.

[16] J. Chen, L. Shen, J. Yin, and C. Zhang, "Parallel programming course development based on parallel computational thinking," in *Proceedings of ACM Turing Celebration Conference-China*, 2018, pp. 103–109.

[17] M. Arroyo, "Teaching parallel and distributed computing to undergraduate computer science students," in *Proceedings of IPDPSW*. IEEE, 2013, pp. 1297–1303.

[18] J. C. Adams, "Injecting parallel computing into cs2," in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 277–282.

[19] R. A. Brown, J. C. Adams, C. Ferner, E. Shoop, and A. B. Wilkinson, "Teaching parallel design patterns to undergraduates in computer science," in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 547–548.

[20] M. I. Capel, A. J. Tomeu, and A. G. Salguero, "Teaching concurrent and parallel programming by patterns: An interactive ict approach," *Journal of Parallel and Distributed Computing*, vol. 105, pp. 42–52, 2017.

[21] J. Adams, R. Brown, and E. Shoop, "Patterns and exemplars: Compelling strategies for teaching parallel and distributed computing to cs undergraduates," in *Procedings of IPDPSW*. IEEE, 2013, pp. 1244–1251.

[22] C. Ferner, B. Wilkinson, and B. Heath, "Toward using higher-level abstractions to teach parallel computing," in *Procedings of IPDPSW*. IEEE, 2013, pp. 1291–1296.

[23] ——, "Using patterns to teach parallel computing," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 2014, pp. 1106–1113.

[24] M. I. Capel, A. J. Tomeu, and A. G. Salguero, "A set of patterns for concurrent and parallel programming teaching," in *European Conference on Parallel Processing*. Springer, 2017, pp. 203–215.

[25] C. Lupo, Z. J. Wood, and C. Victorino, "Cross Teaching Parallelism and Ray Tracing: A Project-based Approach to Teaching Applied Parallel Computing," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, pp. 523–528.

[26] H. Burkhart, D. Guerrera, and A. Maffia, "Trusted high-performance computing in the classroom," in *Procedings of Workshop on Education for High-Performance Computing*. IEEE Press, 2014, pp. 27–33.

[27] M. Bourgoin, E. Chailloux, and J.-L. Lamotte, "Efficient abstractions for gpgpu programming," *International Journal of Parallel Programming*, vol. 42, no. 4, pp. 583–600, 2014.

[28] F. Almeida, J. Cuenca, R. Fern'ndez-Pascual, D. Giménez, and J. A. P. Benito, "The spanish parallel programming contests and its use as an educational resource," in *Proceedings of IPDPSW*. IEEE, 2012, pp. 1303–1306.

[29] A. Desai, M. Tippins, and J. B. Arbaugh, "Learning through collaboration and competition: Incorporating problem-based learning and competition-based learning in a capstone course," *Organization Management Journal*, vol. 11, no. 4, pp. 258–271, 2014.

[30] J. C. Burguillo, "Using game theory and competition-based learning to stimulate student motivation and performance," *Computers & Education*, vol. 55, no. 2, pp. 566 – 575, 2010.

[31] J. Fresno, A. Ortega-Arranz, H. Ortega-Arranz, A. Gonzalez-Escribano, and D. R. Llanos, *Applying gamification in a parallel programming course*, 2016.

[32] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Soft. Eng.*, 05 2012, pp. 123–151.

[33] B. J. Duch, S. E. Groh, and D. E. Allen, *The power of problem-based learning: a practical" how to" for teaching undergraduate courses in any discipline*. Stylus Publishing, LLC., 2001.

[34] P. Armitage, G. Berry, and J. N. S. Matthews, *Statistical methods in medical research*. John Wiley & Sons, 2008.

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

[36] 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.

[37] CSUS. Programming contest control system. Accessed: 2020-04-16. [Online]. Available: https://pc2.ecs.csus.edu/

[38] J. P. Leal and F. Silva, "Mooshak: A web-based multi-site programming contest system," *Software: Practice and Experience*, vol. 33, no. 6, pp. 567–581, 2003.