Extending the Planning Poker Method to Estimate the Development Effort of Parallel Applications

Gabriella Andrade[1], Dalvan Griebler[2], Rodrigo Santos[3], Luiz Gustavo Fernandes[2]

[1]Federal Institute of Rio Grande do Sul (IFRS) – Campus Restinga
Porto Alegre – RS – Brazil

[2]School of Technology – Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

[3]Department of Applied Informatics, Federal University of the State of Rio de Janeiro (UNIRIO), Rio de Janeiro – RJ – Brazil

gabriella.andrade@restinga.ifrs.edu.br,

{dalvan.griebler,luiz.fernandes}@pucrs.br, rps@uniriotec.br

Abstract. *Since different Parallel Programming Interfaces (PPIs) are available to programmers, evaluating them to identify the most suitable PPI also became necessary. Recently, in addition to the performance of PPIs, developers' productivity has also been evaluated by researchers in parallel processing. Some researchers conduct empirical studies involving people for productivity evaluation, which is time-consuming. Aiming to propose a less costly method for evaluating the development effort of parallel applications, we proposed modifying the Planning Poker method in this paper. We consider a representative set of parallel stream processing applications to evaluate the proposed modification. Our results showed that the proposed method required less effort for practical use than the controlled experiments with students.*

1. Introduction

Multi-core is a popular and widely used parallel architecture for increasing the performance of sequential code by executing it in parallel on multiple cores [Kirk e Hwu 2016]. However, this task is challenging as the developer must deal with low-level architectural details and address parallelism-specific aspects, such as load balancing, thread management, and synchronization. Over the years, several Parallel Programming Interfaces (PPIs) have been created, providing abstractions to relieve programmers from dealing with lower-level implementations and architecture-specific optimizations. PPIs can be based on structured or non-structured approaches. Structured parallel programming is a higher-level approach where the concept leverages parallel patterns that can be a receipt/guide to writing efficient parallel software. Furthermore, it can be provided as ready-to-use templates that already implement lower-level parallelisms, such as thread communication and synchronization, regardless of the target architecture [McCool et al. 2012].

Over the years, parallel programming researchers began to be concerned with assessing the different PPIs available to determine the most suitable one. Most researchers in parallel programming focus on evaluating the execution time and performance of an application without considering the effort involved in the application development process, making it difficult to determine the productivity of PPIs precisely. However, productivity (or efficiency) is an important factor that, together with effectiveness and user satisfaction, are usability indicators [ISO 2018]. By considering productivity and usability, it is

possible to provide indicators for designing new PPIs and refining existing ones. This approach allows the creation of easy-to-use PPIs that continue to increase the abstraction of parallelism without compromising the application's performance.

In this context, productivity is commonly measured by human effort concerning the results achieved [ISO 2018], including the time to develop software. Experiments should be conducted in controlled environments to evaluate productivity, in which human subjects (usually students) resolve small programming tasks using different PPIs [Nanz et al. 2013, Miller e Arenaz 2019]. However, experimentation is time-consuming because it must be carefully planned and executed [Wohlin et al. 2012], and finding a representative sample of participants in the parallel programming domain is challenging. Instead, several researchers on parallel programming use established off-line code metrics to estimate the development time and facilitate productivity evaluation.

Our previous work [Andrade et al. 2022b] identified that Halstead and the Constructive Cost Model (COCOMO) are the most widely used metrics for estimating the time required to develop parallel applications. However, these metrics aim at evaluating general-purpose software without considering a specific domain. Therefore, in our previous work [Andrade et al. 2022b], we proposed an approach to evaluate parallel applications using Halstead (PHalstead) and a refined COCOMO II reuse model version. Aiming to evaluate our proposed approaches, we compared them with a series of classical prediction metrics when estimating the effort to develop parallel stream processing applications on multicore systems using FastFlow [Aldinucci et al. 2017], SPar [Griebler et al. 2017] and TBB [Voss et al. 2019]: COCOMO II variations (original, reuse and maintenance), Function Points [Wazlawick 2013], Planning Poker, Putnam [Putnam e Myers 1991], SEER for Software [Galorath e Evans 2006] and Use Case Points [Wazlawick 2013]. The results showed the effectiveness of the Planning Poker method in estimating the development effort of parallel stream processing applications [Andrade et al. 2022b]. Although this method requires input from expert developers, it requires less effort to be applied in practice than experiments that collect the actual development time. Therefore, in this study, we proposed a new methodology to estimate the effort required to develop parallel applications based on the Planning Poker estimation method.

In order to evaluate the proposed method, we conducted a quasi-experiment with developers who are specializing in parallel stream processing applications. Stream processing is a prominent real-time paradigm for collecting, processing, and analyzing high-volume, heterogeneous, and continuous data streams. Therefore, parallelism exploitation in stream processing applications can be implemented mainly using the Pipeline and Farm patterns [Andrade et al. 2014]. Such patterns can be more easily implemented using PPIs based on structured parallel programming. In this study, we requested participants to use the Planning Poker modification to estimate the effort required to develop parallel stream applications using structured (FastFlow and TBB) and unstructured (OpenMP and Pthreads) PPIs, as well as a domain-specific language (DSL) for stream processing (SPar) at a high abstraction level. Therefore, comparing the productivity provided by PPIs with different abstraction levels was also possible.

The scientific contributions provided in this study are: (i) an analysis of developers' perceptions of the effort required to develop stream processing applications in multi-core environments; (ii) an analysis of PPIs with different programming models when expressing stream parallelism in multi-core systems; and (iii) a less costly method for estimating the development effort of parallel applications.

This paper is organized as follows. Section 2 presents the background; Section 3 introduces the proposed modification to the Planning Poker method; Section 4 provides the experimentation plan; Section 5 provides the results; Section 5.1 presents the threats to validity; finally, Section 6 concludes this study.

2. Background

2.1. Planning Poker method

Over the years, several quantitative metrics have been proposed to estimate software effort, quality, and reliability based on code size, parametric methods etc. Nowadays, Planning Poker [Cohn 2005] is a metric commonly used by software agile teams to estimate the user story, which defines features and requirements that provide information to the user or customer [Durán et al. 2019]. It relies on experts' opinions about the software to be developed to guess the development effort. Participants in the planning poker method include all persons in the developing team, such as the developers, testers, engineers, and others. In addition, there is a moderator to coordinate the execution of the method.

Before starting, the moderator should prepare a deck of cards with a valid sequence of numbers written on each card. Planning poker decks are usually based on Fibonacci sequence [Gandomani et al. 2019]: 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100. The original Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...) also can be used. However, modifying the Fibonacci sequence allows development team members to estimate projects with development effort close to 1/2. In addition, each number must have a meaning, such as story points, number of product backlogs, or development time.

After the decks have been distributed, the moderator or team leader should explain to the team members the application or project requirements to be developed. Then, each participant receives a deck of cards and selects a card representing their estimation opinion. All participants must discuss justifying their choice. If the experts disagree with the estimates, they can repeat the process until the results converge. On the other hand, averaging the estimates can be done to avoid too many rounds [Cohn 2005]. Figure 1 illustrates the steps required to use the Planning Poker method to estimate story points.

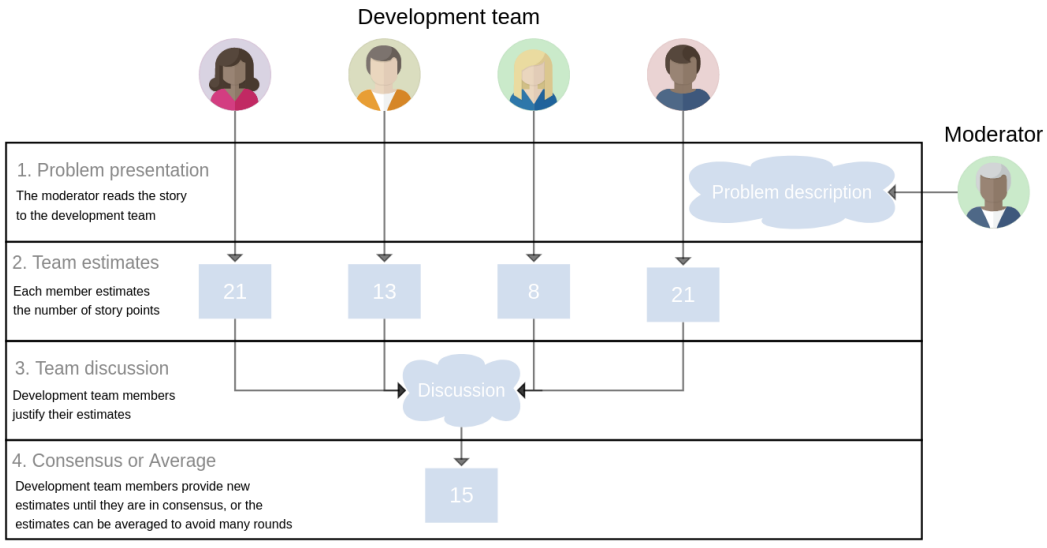


Figure 1. Planning Poker steps for estimating the number of story points.

According to [Cohn 2005], Planning Poker is a metric that works by several factors. First, it gathers several opinions from experts in estimation, who form a cross-functional team from all stages of a software project. Secondly, estimators are called upon to justify their estimates, a factor that improves the estimation's accuracy. Finally, studies have shown that averaging the individual participants' estimates lead to better results and holding group discussions.

2.2. Related Work

Based on the following literature reviews [Fernández-Diego et al. 2020, Durán et al. 2019], the Planning Poker is the most widely used metric for estimating software projects' complexity and development effort based on the Scrum methodology. Regarding development effort, [Moløkken-Østvold et al. 2008] conducted empirical studies to compare the Planning Poker estimation with the estimation performed by individual experts, which showed similar accuracy. [Finco 2021] proposed the combination of Planning Poker with machine learning to assess the development effort required for software development teams. [Haugen 2006] investigated whether using the Planning Poker estimation process could improve the ability of XP teams to estimate story points. [Tamrakar e Jørgensen 2012] conducted an empirical study to assess software development effort using Planning Poker with a linear scale instead of the usual Fibonacci sequence. [Gandomani et al. 2019] evaluate the use of average or consensus opinion in Planning Poker to estimate user stories.

To produce more accurate estimates using Planning Poker, [Zahraoui e Idrissi 2015] proposed an adjustment to the story point calculations using priority, size, and complexity factors. [Sudarmaningtyas e Mohamed 2020] proposed a new model to improve the actual performance of planning poker by modifying the estimation process and the consensus process of this method. [Mahnič e Hovelja 2012] ran experiments with students and experts to estimate the number of user stories through the Planning Poker method, whose results showed that the experienced participants' estimates were more accurate than the students' estimates. [Raith et al. 2013] assessed the accuracy of the effort estimates using the Planning Poker, which led them to develop a prototype to apply this method to a student project.

Unlike previous works, our study aims to apply a modification of the Planning Poker method to the parallel programming domain. In this study, we proposed a modification of Planning Poker since our previous study [Andrade et al. 2022b] identified it as a good metric for evaluating parallel applications.

3. Extending the Planning Poker Metrics for Parallel Programming

In the original Planning Poker, the estimation is performed for all the people who compose the development team, such as the developers, testers, engineers, analysts, and others. In parallel programming, mainly in academic environments, the application will be developed by only one developer. The researchers usually develop their applications without being in a development team, although there are research groups. In other words, the same person will be responsible for coding, debugging, testing, and evaluating an application. Therefore, the Planning Poker adaptation estimation will be performed only by a developer instead of a development team, as seen in Figure 2. Furthermore, the research results presented in [Andrade et al. 2022a] highlight developer experience as an essential factor impacting the parallel application development effort. These results led us to consider only the opinion of experienced developers for the Planning Poker method.

Unlike the original Planning Poker method, which aims to estimate the number of story points or Product Backlog, our goal is to estimate the number of hours required for parallel application development. Furthermore, we will not perform our estimations based on a non-linear sequence, such as the Fibonacci sequence. [Tamrakar e Jørgensen 2012] showed a decrease in estimated development effort by up to 60% when using a Fibonacci scale instead of the traditional linear scale. Using a Fibonacci scale and other non-linear scales probably affect development effort estimates because they induce people to make biased estimates, especially when the uncertainty is substantial. When considering the modified Fibonacci sequence (0, 1, 1/2, 2, 3, 5, 8, 13, 20, 40, 100), a developer may be in doubt when estimating the number of development hours for a particular application. For example, in the developer's opinion, a specific application requires about 60 hours to develop. Therefore, they might choose the value 100 from the Fibonacci sequence, resulting in an inaccurate estimate. On the other hand, a developer could choose the specific value of the evaluation (60 hours in this case), if a linear scale was used.

The estimators will be also free to choose their estimates and can compare the estimated values between the evaluated interfaces considering their possible difficulties. There are more straightforward applications where parallelization can be done within one to two hours of development time, such as the RGB channel extraction application presented in [Andrade et al. 2023]. Therefore, a linear scale can accurately estimate development time for more straightforward applications, as in the case.

In the proposed Planning Poker adaptation, there is still a moderator to coordinate the execution of the method. Figure 2 presents the step-by-step to be followed through the proposed methodology. Initially, the moderator should show the code to be parallelized with a given PPI. Next, a developer should analyze the code and give an estimate. If there are any inconsistencies with the developer's estimate, a discussion should occur between the developer and the moderator. For example, the development time estimated by the developer was only two hours to parallelize a stream processing application with OpenMP. During the discussion, the moderator should give the developer reasons to disagree with the estimate. For example, the moderator should explain that the OpenMP programming

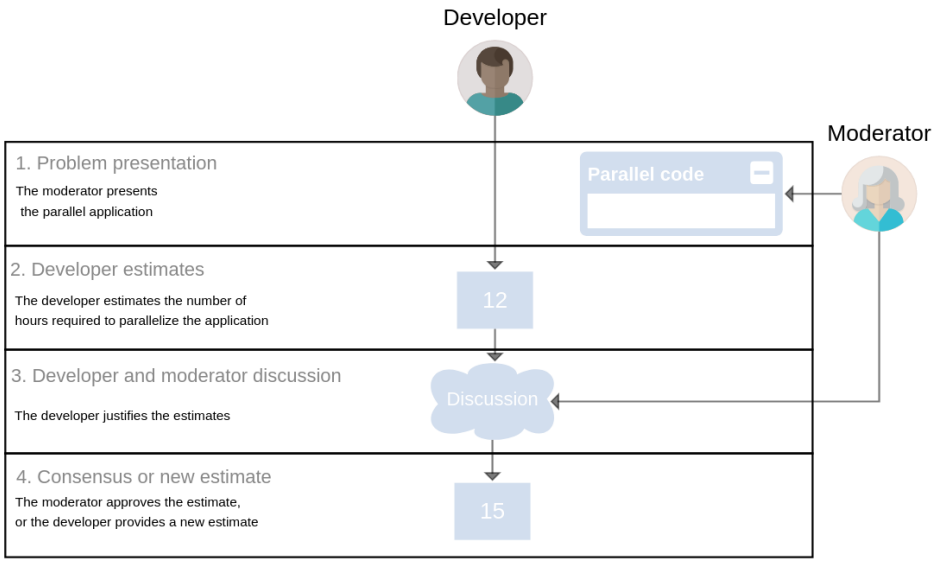


Figure 2. Planning Poker method for estimating parallel development time.

model is not based on structured programming (such as TBB). Therefore, the developer should consider the time needed to implement extra synchronization mechanisms when using OpenMP to exploit stream parallelism. From the moderator's exposition, the developer should make a new estimate. Moreover, the estimation will be based only on the developer's opinion, and the moderator should only explain the characteristics of the application to be parallelized and the PPI to be considered in parallelization.

4. Method

To evaluate the Planning Poker adaptation applied to parallel programming, we performed an experiment to verify the productivity of FastFlow, OpenMP, Pthreads, SPar, and TBB PPIs for multi-core systems in the development of the following C++ stream processing applications. This section presents the experimentation plan, in which are included the variables, the goals, the hypotheses, the context, the activity given to the participants, the procedure followed, and the instruments used in this study [Wohlin et al. 2012].

4.1. Independent and dependent variables

This section presents the variables of this study. The development time estimated using the Planning Poker method is a dependent variable. The independent variables include the PPIs evaluated, the parallel applications evaluated, the participant's experience, and the study environment because they impact the dependent variable.

4.2. Goals

The main goal of this study is to use the Planning Poker method to estimate the development effort required for implementing parallelism in C++ stream processing applications for multi-core environments using FastFlow, SPar, TBB, OpenMP, and Pthreads. To do so, we measured the effort required to exploit parallelism using the Planning Poker for Bzip2 compress and decompress, Person Recognition, Lane Detection, and Ferret applications.

4.3. Hypotheses

Based on the goals, we consider the following hypothesis in our experiment: *(i) Null Hypothesis (H_0):* The effort required to implement parallelism is the same for FastFlow, SPar, TBB, OpenMP, and Pthreads; *(ii) Alternative Hypothesis (H_1):* The effort required to implement parallelism is not the same for FastFlow, SPar, TBB, OpenMP, and Pthreads.

4.4. Context of the study

This study is offline because it was conducted in an academic environment under controlled conditions. The participants were five graduate students from the Graduate Program in Computer Science at the PUCRS in Porto Alegre city, South of Brazil. They have more than two years of experience developing parallel stream processing applications. Finally, this is a specific study because it focuses on evaluating the productivity of PPIs for parallel programming of stream processing applications in an academic environment.

4.5. Activity of study

The activity given to the participants was to estimate the time required to develop a series of stream processing applications developed using C++ and parallelized using FastFlow, OpenMP, Pthreads, SPar, and TBB: *(i) Bzip2:* An application designed for data compression and decompression in Bzip2 format ; *(ii) Person Recognition:* A video application

designed to recognize people; *(iii) Lane Detection:* A video application designed for detecting road lanes on video feeds; *(iv) Ferret:* A PARSEC application whose goal is to content similarity search in data such as video, audio, and images.

Stream processing is a paradigm for collecting, processing, and analyzing high-volume, heterogeneous, and continuous data streams in real time. Therefore, we chose the abovementioned applications because they are standard in real-world stream processing, including video and audio processing and data compression applications [Andrade et al. 2014].

4.6. Procedure and execution

Participants should estimate the time required to develop the parallel applications, considering an existing sequential application. Participants should also consider that it is their first time parallelizing the application, regardless of the PPI used. They also should not consider reusing code from another parallel version to avoid contaminating the experiment with the learning effect. After analyzing the target application, the developers reported their estimates in hours required to develop the applications.

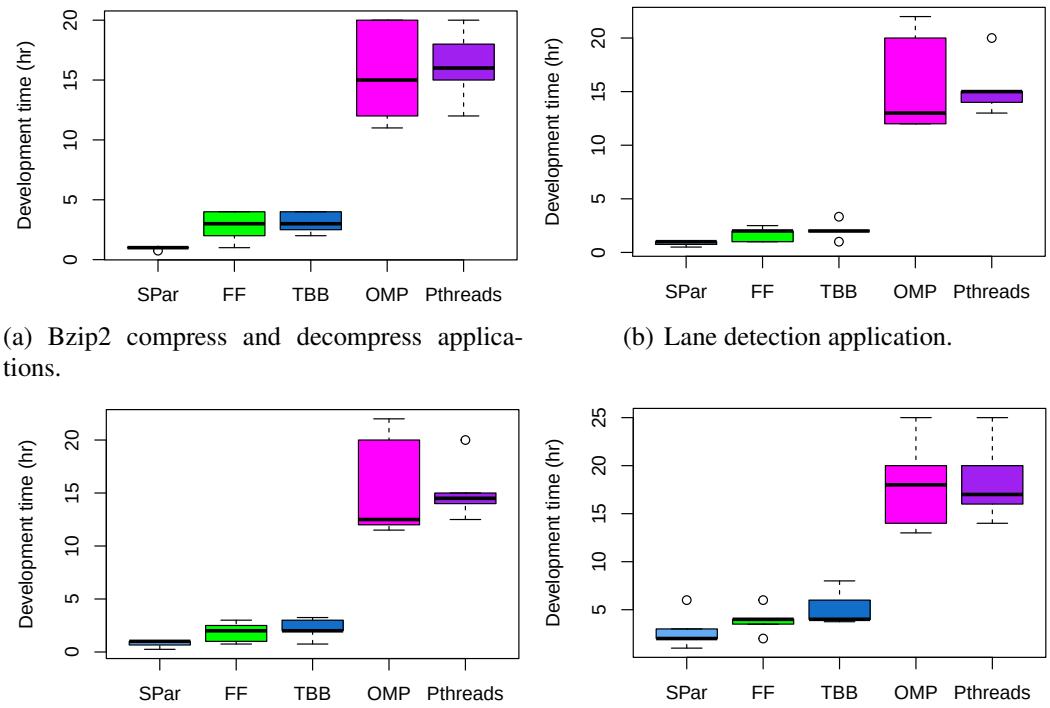
5. Results

Figure 3 shows the box plots of the estimated development time by each of the five developers using the Planning Poker modification, who are experts in developing parallel stream processing applications. Figure 3 shows that all developers in this study agreed that SPar is the interface that requires the least effort to develop parallel stream processing applications. According to the developers, FastFlow and TBB require a development effort close to that of SPar. In addition, OpenMP and Pthreads also showed close results, which are the PPIs that require more effort to develop the evaluated applications according to the participants' opinions. Therefore, it was necessary a hypothesis test to see if there was a significant difference between the results obtained.

Initially, we performed a normality test to verify that the data collected had a normal distribution. A parametric test should be performed if the samples have a normal distribution (p-value ≥ 0.05). Otherwise, a non-parametric test should be performed (p-value < 0.05). Therefore, we used the Shapiro-Wilk test at the conventional significance level ($\alpha = 0.05$) to verify whether the collected data had a normal distribution. Moreover, we chose the Shapiro-Wilk test because it is an efficient test for all distribution types and can be used regardless of sample size [Razali et al. 2011].

Table 1 shows the results achieved for the Shapiro-Wilk tests. Only the SPar results for the Bzip2 compress and decompress applications do not show a normal distribution since, for these cases, the p-values are close to zero (in bold). Therefore, we used the non-parametric Wilcoxon test to compare the results of the SPar with the results of the other interfaces for the Bzip2 compress and decompress applications. We ran a Student's t-test for two samples in all other cases. In addition, we performed paired tests as the samples were not independent since we collected the data from the same participants.

Table 2 shows the results for the Student's t and Wilcoxon tests. The average time to develop the applications with SPar is shorter due to the annotation-based programming model of this DSL. However, through hypothesis testing, it was possible to observe that there is no significant difference between SPar and FastFlow development times for the applications Bzip2 Compress, Bzip2 Decompress, Person Recognition, and Ferret. This result occurred because the FastFlow development model is based on the use of templates



(a) Bzip2 compress and decompress applications.

(b) Lane detection application.

(c) Person recognition application.

(d) Ferret application.

Figure 3. Estimated development time with modified Planning Poker.

Table 1. P-value of the Shapiro-Wilk test for Planning Poker evaluation.

	Bzip2 comp.	Bzip2 decomp.	Lane det.	Person recog.	Ferret
	p-value	p-value	p-value	p-value	p-value
SPar	**0.0001**	**0.0001**	0.05	0.05	0.22
FastFlow	0.42	0.42	0.20	0.61	0.68
TBB	0.38	0.38	0.28	0.55	0.10
OpenMP	0.21	0.21	0.07	0.08	0.70
Pthreads	0.99	0.99	0.12	0.18	0.64

for designing parallel patterns. The results of our previous study [Andrade et al. 2023] showed that FastFlow provides similar productivity for developers as SPar.

Regarding Bzip2 compress and decompress applications, when we analyzed only the participants' average, SPar required less development effort. However, the hypothesis test showed no significant difference between the average development times for SPar and the others PPIs evaluated. This may have happened because the SPar samples for the Bzip compress and decompress applications do not have a normal distribution. In these cases, it was necessary to use the Wilcoxon test to compare them with the results of the other interfaces. We normalized these data using the Min-Max normalization [Larose e Larose 2014] to overcome this limitation. With the normalized samples, we used the Student's t-test to compare them. Table 2 shows that when comparing the normalized samples, the SPar development time differs statistically from FastFlow TBB, OpenMP, and Pthreads (p-value ≤ 0.0217).

FastFlow and TBB showed close results, as shown in Figure 3. For all the applications evaluated, the hypothesis test showed that there is no significant difference between the development time estimated for FastFlow and TBB, as shown in Table 2 (p-value $>$ 0.05). These results occurred due to the similarities in FastFlow and TBB programming

Table 2. P-value of the Student's t-test and Wilcoxon test for Planning Poker evaluation.

Wilcoxon	Bzip2 compress	Bzip2 decompress	Lane detection	Person recog.	Ferret
	p-value	p-value	p-value	p-value	p-value
SPar x FastFlow	**0.0975**	**0.0975**	-	-	-
SPar x TBB	**0.0579**	**0.0579**	-	-	-
SPar x OpenMP	**0.0579**	**0.0579**	-	-	-
SPar x Pthreads	**0.0625**	**0.0625**	-	-	-
Student's t-test	Bzip2 compress	Bzip2 decompress	Lane detection	Person recog.	Ferret
	p-value	p-value	p-value	p-value	p-value
SPar x FastFlow	0.0217	0.0217	0.0434	**0.05272**	**0.3131**
SPar x TBB	0.0036	0.0036	0.0265	0.0224	0.0009
SPar x OpenMP	0.0012	0.0012	0.0021	0.0024	0.0008
SPar x Pthreads	0.0002	0.0002	0.0002	0.0003	0.0002
FastFlow x TBB	**0.2080**	**0.2080**	**0.1803**	**0.1836**	**0.1855**
FastFlow x OpenMP	0.0012	0.0012	0.0024	0.0023	0.0022
FastFlow x Pthreads	0.0003	0.0003	0.0004	0.0007	0.0013
TBB x OpenMP	0.0018	0.0018	0.0036	0.0034	0.0012
TBB x Pthreads	0.0004	0.0004	0.0006	0.0009	0.0003
OpenMP x Pthreads	**0.6657**	**0.6657**	**0.8486**	**0.8486**	**0.7572**

models. TBB [Voss et al. 2019] is an open-source and general-purpose C++ template-based PPI from the industry, which provides a Pipeline pattern constructor that can also perform as the Farm pattern, while FastFlow [Aldinucci et al. 2017] is a representative interface from the scientific community with a model-based C++ interface similar to the TBB. However, each model has particularities, although parallelism is implemented similarly. Nevertheless, the results showed that they require the same programming effort. The same occurred in the experiment presented in a previous work [Andrade et al. 2023], where the hypothesis test also showed no significant difference between the time needed to parallelize the RGB channel extraction application by beginners developers.

OpenMP and Pthreads also presented similar results, as seen in Table 2. The hypothesis test also showed no significant difference between the estimated development times for OpenMP and Pthreads (p-value > 0.1018), which require at least 85% more effort to parallelize the applications evaluated from the opinion of the participants of this study. OpenMP and Pthreads do not have similar programming models as FastFlow and TBB. Parallelization in OpenMP is exploited through compilation directives or pragmas defined in the C and C++ standards. The Pthreads library is based on the POSIX specification that defines a set of types, functions, and macros for creating and controlling multiple threads. Despite the differences between OpenMP and Pthreads, both have similar programming efforts to parallelize stream applications by not providing a structured programming model. To parallelize stream applications such as the ones evaluated in this study, it is necessary to implement parallelism patterns as Pipeline and Farm, which are composed of different processing stages. Therefore, to implement stream parallelism with Pthreads and OpenMP, it is necessary to develop mechanisms such as ordered insertion into chained queues manually, unlike the FastFlow, TBB, and SPar PPIs. Only experienced developers would realize such features. Therefore, considering the opinion of beginner developers in the Planning Poker method could result in incorrect estimates.

5.1. Threats to validity

This study showed promising results. However, some threats to validity remain. One threat to internal validity identified was the possibility of information sharing among the participants of this study. To address this threat, we contacted the participants individually. As a threat to external validity, it is not possible to generalize the results because we only evaluated stream processing in multi-core environments. In addition, the experiment

was conducted in a classroom rather than an industrial context, making it difficult to generalize the results. As such, results serve as indications, and the experimental plan can be replicated to generate new results to be analyzed and provide evidence.

We identified a threat to construct validity related to the possibility of participants giving incorrect answers because they thought they were being evaluated. To minimize this threat, we informed the participants that the experiment did not represent any personal evaluation and that the data collected would be anonymized. Finally, a threat to conclusion validity is the sample size, which may need to be more representative. Therefore, an experiment with a larger sample of participants may be necessary to confirm the results.

6. Conclusion and Future Work

This study presented a proposed extension of the Planning Poker method to evaluate the development of parallel applications. It is already a well-established method in agile development, widely used and spread among agile teams. The Planning Poker method has proven to be a promising and effective method for estimating the development time of parallel applications when considering the opinion of experienced parallel application developers in our previous study [Andrade et al. 2022b]. Additionally, the results of a survey we conducted previously [Andrade et al. 2022a] highlighted the developer's experience as one of the main factors affecting productivity. The potential of Planning Poker motivated us to create a methodology for its use in the parallel programming area.

The proposed methodology was evaluated through a quasi-experiment with five developers with experience in developing stream processing applications. The participants used the Planning Poker method to estimate the time required to develop five parallel stream processing applications using FastFlow, TBB, SPar, OpenMP, and Pthreads. Our results showed that the Planning Poker method has promising results comparable to the actual development time collected through controlled experiments as those presented in [Andrade et al. 2023]. Moreover, Planning Poker requires less effort to use in practice than such experiments. Therefore, we concluded that this method is an alternative for measuring development time.

The results showed that SPar is the PPI requiring the least programming effort, followed by FastFlow and TBB. However, for the applications Bzip2 compress, Bzip2 decompress, Person Recognition, and Ferret, the hypothesis test did not show a significant difference between FastFlow and SPar development times. The hypothesis test also indicated no significant difference between the estimated development times for FastFlow and TBB due to the similarities in their programming models. OpenMP and Pthreds required more development effort to parallelize the applications evaluated from the opinion of the participants, and Pthreads showed the worst results. Nevertheless, the same behavior also occurred for OpenMP and Pthreads. Despite the differences between OpenMP and Pthreads, both have similar programming efforts to parallelize stream applications by not providing a structured programming model.

Planning Poker has showed to be a promising method for estimating the time required to develop parallel applications. However, its use still has some limitations. By performing an experiment with people to get the real-time needed to develop a given application, it is possible to collect data about the difficulties and challenges faced by them during the execution of the task. This type of experiment allows us to collect the participants' opinions about their satisfaction with the interfaces. A qualitative analysis of the participants' answers can also provide valuable insights, such as those presented

in [Andrade et al. 2023, Andrade et al. 2022a]. To address this limitation, we propose the use of a questionnaire for reporting the reasons for the programming effort related to a given PPI, its limitations, and problems. Moreover, it is important that participants also report their satisfaction with the evaluated PPIs.

We evaluated several metrics for estimating the development effort of parallel applications, among which Planning Poker was the most prominent. Agile development teams widely use this method. Several other estimation models for agile projects are available, such as t-shirt sizing, dot voting, bucket system, large/uncertain/small, and affinity mapping [Mallidi e Sharma 2021]. Therefore, the evaluation of such models in the parallel programming domain still needs to be explored.

Acknowledgment

This research is partially funded by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

Aldinucci, M., Danelutto, M., Kilpatrick, P., e Torquati, M. (2017). Fastflow: High-level and efficient streaming on multi-core. In *Programming Multi-core and Many-core Computing Systems*, pages 261–280.

Andrade, G., Griebler, D., Santos, R., e Fernandes, L. G. (2022a). Opinião de brasileiros sobre a produtividade no desenvolvimento de aplicações paralelas. In *WSCAD 2022*, pages 276–287.

Andrade, G., Griebler, D., Santos, R., e Fernandes, L. G. (2023). A parallel programming assessment for stream processing applications on multi-core systems. *Computer Standards & Interfaces*, 84:1–25.

Andrade, G., Griebler, D., Santos, R., Kessler, C., Ernstsson, A., e Fernandes, L. G. (2022b). Analyzing programming effort model accuracy of high-level parallel programs for stream processing. In *SEAA 2022*, pages 229–232.

Andrade, H. C., Gedik, B., e Turaga, D. S. (2014). *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press.

Cohn, M. (2005). *Agile estimating and planning*. Pearson Education.

Durán, M., Juárez-Ramírez, R., Jiménez, S., e Tona, C. (2019). Taxonomy for complexity estimation in agile methodologies: A systematic literature review. In *CONISOFT 2019*, pages 87–96.

Fernández-Diego, M., Méndez, E. R., González-Ladrón-De-Guevara, F., Abrahão, S., e Insfran, E. (2020). An update on effort estimation in agile software development: A systematic literature review. *IEEE Access*, 8:166768–166800.

Finco, D. A. (2021). Combinando planning poker e aprendizado de máquina para estimar esforço de software. In *ERES 2011*, pages 129–138.

Galorath, D. D. e Evans, M. W. (2006). *Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves*. Auerbach Publications.

Gandomani, T. J., Faraji, H., e Radnejad, M. (2019). Planning poker in cost estimation in agile methods: Averaging vs. consensus. In *KBEI 2019*, pages 66–71.

Griebler, D., Danelutto, M., Torquati, M., e Fernandes, L. G. (2017). Spar: A dsl for high-level and productive stream parallelism. *Parallel Processing Letters*, 27(1):1–14.

Haugen, N. C. (2006). An empirical study of using planning poker for user story estimation. In *AGILE 2006*, pages 1–9.

ISO (2018). ISO 9241-11:2018 – Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts.

Kirk, D. B. e Hwu, W.-m. W. (2016). *Programming massively parallel processors: A hands-on approach*. Morgan Kaufmann.

Larose, D. T. e Larose, C. D. (2014). *Discovering knowledge in data: An introduction to data mining*. John Wiley & Sons, 2nd edition.

Mahnič, V. e Hovelja, T. (2012). On using planning poker for estimating user stories. *Journal of Systems and Software*, 85(9):2086–2095.

Mallidi, R. K. e Sharma, M. (2021). Study on agile story point estimation techniques and challenges. *International Journal of Computer Applications*, 174(13):9—14.

McCool, M., Reinders, J., e Robison, A. (2012). *Structured parallel programming: Patterns for efficient computation*. Morgan Kaufmann Publishers.

Miller, J. e Arenaz, M. (2019). Measuring the impact of hpc training. In *EduHPC 2019*, pages 58–67.

Moløkken-Østvold, K., Haugen, N. C., e Benestad, H. C. (2008). Using planning poker for combining expert estimates in software projects. *Journal of Systems and Software*, 81(12):2106–2117.

Nanz, S., West, S., Da Silveira, K. S., e Meyer, B. (2013). Benchmarking usability and performance of multicore languages. In *ESEM 2013*, pages 183–192.

Putnam, L. H. e Myers, W. (1991). *Measures for Excellence: Reliable Software on Time, within Budget*. Prentice Hall PTR.

Raith, F., Richter, I., Lindermeier, R., e Klinker, G. (2013). Identification of inaccurate effort estimates in agile software development. In *APSEC 2013*, pages 67–72.

Razali, N. M., Wah, Y. B., et al. (2011). Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of Statistical Modeling and Analytics*, 2(1):21–33.

Sudarmaningtyas, P. e Mohamed, R. B. (2020). Extended planning poker: A proposed model. In *ICITACEE 2020*, pages 179–184.

Tamrakar, R. e Jørgensen, M. (2012). Does the use of fibonacci numbers in planning poker affect effort estimates? In *EASE 2012*, pages 228–232.

Voss, M., Asenjo, R., e Reinders, J. (2019). *Pro TBB: C++ parallel programming with threading building blocks*. Apress.

Wazlawick, R. (2013). *Engenharia de Software: Conceitos e Práticas*. Elsevier.

Wohlin, C., Runeson, P., Høst, M., Ohlsson, M. C., Regnell, B., e Wesslén, A. (2012). *Experimentation in software engineering*. Springer.

Zahraoui, H. e Idrissi, M. A. J. (2015). Adjusting story points calculation in scrum effort & time estimation. In *SITA 2015*, pages 1–8.