2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) | 978-1-6654-6152-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SEAA56994.2022.00043

Analyzing Programming Effort Model Accuracy of High-Level Parallel Programs for Stream Processing

Gabriella Andrade*, Dalvan Griebler*, Rodrigo Santos‡, Christoph Kessler§, August Ernstsson§, Luiz G. Fernandes*
* School of Technology, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil.
‡ Department of Applied Informatics, Federal University of the State of Rio de Janeiro (UNIRIO), Rio de Janeiro, Brazil.
§ Department of Computer and Information Science (IDA), Linköping University, Linköping, Sweden.
Email: gabriella.andrade@edu.pucrs.br, {dalvan.griebler,luiz.fernandes}@pucrs.br, rps@uniriotec.br,
{christoph.kessler,august.ernstsson}@liu.se

***Abstract*—Over the years, several Parallel Programming Models (PPMs) have supported the abstraction of programming complexity for parallel computer systems. However, few studies aim to evaluate the productivity reached by such abstractions since this is a complex task that involves human beings. There are several studies to develop predictive methods to estimate the effort required to develop software applications. In order to evaluate the reliability of such metrics, it is necessary to assess the accuracy in different programming paradigms. In this work, we used the data of an experiment conducted with beginners in parallel programming to determine the effort required for implementing stream parallelism using FastFlow, SPar, and TBB. Our results show that some traditional software effort estimation models, such as COCOMO II, fall short. In contrast, Planning Poker could contribute toward a parallel-aware effort model.**

***Index Terms*—Accuracy; Development effort; Parallel computing; Programmer productivity; Software metrics.**

I. Introduction

The inability of the silicon industry to increase the performance of single-processor CPUs using traditional techniques has driven the emergence of multi-core architectures, which integrate multiple processing cores into a single chip. Multi-core is currently one of the most popular parallel architectures, from smartphones and personal computers to servers. This architecture has been widely used to increase the performance of sequential codes by executing them in parallel on multiple cores [1]. However, this is not an easy task as the developer must deal with low-level architectural details as well as address parallelism-specific aspects, such as load balancing, synchronization etc. Over the years, several Parallel Programming Models (PPMs) have been created, providing abstractions to relieve programmers from dealing with lower-level implementations and architecture-specific optimizations.

In the parallel programming domain, most studies aim to evaluate the performance of the PPMs, considering only technical factors such as execution time and speedup. Programmer productivity is another critical factor when evaluating PPMs. From productivity (or efficiency), effectiveness, and user satisfaction, it is possible to determine the usability of the PPMs [2]. It is known that developing more productive PPMs and refining existing ones from usability evaluation is possible. However, this is a complex task since measuring productivity involves human beings. Experiments with test persons must be well planned and controlled [3], and it is difficult to find participants experienced in parallel programming. Therefore, many parallel programming researchers instead use established code metrics to facilitate productivity evaluation.

To evaluate and compare PPMs concerning productivity, some researchers use code metrics based on code size (Source Lines of Code (SLOC) [4]–[8], Number of Characters (NOC) [9], and Tokens of Code (TOC) [4], [10], [11]), complexity evaluation (Cyclomatic Complexy Number (CCN) [5], [6], [10], [11] and Information Flow Complexity (IFC) [12]), and development effort (Halstead [4]–[6], [10] and Constructive Cost Model (COCOMO) [7], [13]–[15]). However, these metrics target the evaluation of general-purpose software without considering a specific domain. In this context, in our previous work [16], we aimed to evaluate the feasibility of these metrics when evaluating stream parallelism in multi-core systems using different PPMs: FastFlow, Pthreads, SPar, and TBB.

Our previous results [16] showed that while code metrics based on code size and complexity can be helpful for evaluating PPMs, it is impossible to predict the effort required to develop a parallel application based on these factors alone. Other factors influence the parallel development cycle, such as the development environment, and developer experience. Thus, Halstead and COCOMO II showed more promise for evaluating PPMs, although they also have limitations. In this study, we make initial efforts to overcome some of these limitations. We propose an approach to evaluate parallel applications using Halstead and a refined version of COCOMO II reuse model. There are other predictive metrics that, to our knowledge, have not yet been used to evaluate parallel applications: Function Points (FP), Planning Poker, Putnam, SEER for Software (SEER-SEM), and Use Case Points (UCP). Therefore, the goal of this study is to evaluate the *accuracy* of such metrics compared to the actual effort required to develop parallel stream applications using FastFlow, SPar, and TBB.

The scientific contributions provided in this study are: (i) Accuracy analysis of programming effort models in high-level parallel programs for stream processing on multi-core systems; (ii) An extension of the Halstead's measures (Parallel Halstead - PHalstead); (iii) An extension of the COCOMO II reuse model (Parallel COCOMO II Reuse Model - PCRM).

II. Development time estimation techniques

Over the years, several quantitative metrics have been proposed to estimate software effort, quality, and reliability [15] based on code size, parametric methods etc. Putnam's model is one of the first to estimate the development effort [17].

978-1-6654-6152-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SEAA56994.2022.00043

This model is simple and easily calibrated because it estimates the development time (in years) based on the SLOC and the productivity parameter (PP). PP can be derived from the SLOC, development effort and time of previous projects.

Halstead proposed a series of measures based on TOC, classified as operators or operands [18]. Programming languages have several operators such as the keywords, arithmetic (e.g., $+$, $-$, $*$, $/$, $\%$), logical (e.g., !, $\&\&$, $||$) etc. On the other hand, operands are constants, variables, identifier etc. From counting the number of operators and operands, and the total occurrence of operators and operands, it is possible to measure the code length, vocabulary, volume, programming difficulty, development effort, and development time (in seconds).

COCOMO II is an updated version of a model to estimate the cost and effort required to develop software using source code size in thousand of SLOC (KSLOC), a set of cost drivers, and scale factors for calibrating it [19]. There are two types of cost drivers according to the development phase: the early design model (incept and elaboration) and the post-architecture model (elaboration and construction). Each of them, as well as the scale factors, are evaluated (from extra low to extra high) and used to calculate the effort and time (in months) required to develop a software from scratch. However, variants of the model aim to model the development effort from an existing software (reuse), and for improvements or corrections to an already developed software (maintenance). For this purpose, the KSLOC is measured again. The maintenance size is calculated from the number of lines added to the original code (ASLOC), the number of lines modified in the original code (MSLOC), and the adjusted maintenance factor. On the other hand, the reuse size is based on the ASLOC together with the automatically translated and adaptive modifier factors.

FP [20] is a method independent of the technology and programming language used because it is based on the analysis of the software functional requirements. It is based on five function types: internal logic file, external interface file, external output, external inquiry, and external input. To considers the internal technical complexity of these functions, the FP also evaluates 14 general systems characteristics (from zero to five). From this adjusted, it is possible to estimate the effort and time (in months) needed to develop the application.

UCP [21] is a model inspired by FP, but focusing on the analysis of software use cases. First, the model measures the number of actors (users or systems with which the application communicates) and use cases, and their complexities (simple, average or complex) to calculate the unadjusted UCP. Then, the impact (from zero to five) of a series of technical and environmental factors is evaluated in order to obtain the adjusted UCP. Finally, it is possible to measure the development effort, and consequently estimate the development time (hours).

SEER-SEM [22] is a commercial tool to estimate, and analyze the effort, cost, staffing, schedule, and risk of a software project. It has a database about industry-wide standards for application type, platform, reuse/modification intent, and development method. The code size can be used to improve the estimation as well as other factors such as developers' capabilities, environment, integration level, and volatility.

Planning poker [23] is a method that relies on experts' opinions to guess the development effort. To do so, a moderator prepares a deck of cards with a valid sequence of numbers written on each card (e.g., Fibonacci). Each number must have a meaning, such as the hours required to develop an application. Then, each expert receives a deck of cards, and selects a card representing their estimation opinion. If the experts disagree with the estimate, they can repeat the process until the results converge, or average to avoid too many rounds.

III. Extending the Development Effort Metrics for Parallel Programming

In our previous study [16], we identified some limitations when using the Commented Code Detector (CCD)[1] tool to obtain Halstead measures in parallel stream processing applications developed in C++. CCD does not consider any of the PPMs' keywords as operators (e.g., `spar`, `ff_node`, and `tbb`), because its focus is not on evaluating parallel applications. In addition, CCD was also discontinued in 2014.

There are other more recent tools for getting Halstead measurements on code written in C++, such as Testwell CMT++ and IBM Rational Test RealTime. However, like the CCD tool, these other tools were not developed to evaluate parallel code. To overcome these limitations, we developed the *PHalstead*[2] tool, a Python script to obtain Halstead's measures in C++ applications parallelized with FastFlow, SPar, or TBB. PHalstead considers each of the keywords of C++, libraries and PPMs used to develop the video processing applications.

In our previous study [16], we identified COCOMO II as a good metric for evaluating parallel applications. However, some of its parameters are not usually applied in the development cycle of such applications. Some efforts have been made by Wienke et al. [15] to extend it to parallel domain. However, this model is challenging to apply in practice because it uses linear regression and the dataset used was not available.

In this study, we also make an initial effort to refine the COCOMO II model, focusing on the reuse model because it was designed to evaluate development effort from existing applications, such as parallel applications. This modification was called *PCRM*. To do so, we removed some cost and scale factors that are not relevant in parallel application development when calculating[3] the development effort and time estimated by this model. The reuse model applies only the cost drivers of the post-architecture model, so the cost drivers identified were: required reusability, analyst capability, and personnel continuity. The scale factors identified were: architecture/risk resolution, team cohesion, and process maturity level.

IV. Methodology

In this study, we evaluated the accuracy of development effort estimation models applied to the parallel programming

[1]Available in: https://github.com/dborowiec/commentedCodeDetector.

[2]Available at: https://github.com/GMAP/phalstead.

[3]See [19] to see more specific information about the equations used by COCOMO II to estimate development effort and development time.

domain. To do so, we used the data of an experiment conducted with beginners in parallel programming to collect the time required to implement stream parallelism using FastFlow, SPar, and TBB on multi-core systems. The participants were 15 graduate students of the Pontifical Catholic University of Rio Grande do Sul (PUCRS) in Brazil. They were divided into 3 groups just to vary the order of the PPMs used: SPar, TBB and, FastFlow (first group); TBB, SPar and, FastFlow (second group); and SPar, FastFlow, and TBB (third group). They were given the task of parallelizing a C++ OpenCV video processing application [24], whose objective is to extract the green channel. The activity was considered completed if the parallelization achieved a speedup greater than or equal to 3 and produced the correct result. To do so, they used multi-core workstations, FastFlow 2.1.3[4], TBB 4.4.6, and SPar[5].

We used the following tools to measure the code metrics: PHalstead to get Halstead, SLOCCount[6] to get COCOMO II variations, Function Point Calculator[7] to get FP, Use Case Point Calculator[8] to get UCP, SEER-SEM[9] trial version, and a spreadsheet for the other metrics. The Planning Poker estimates were obtained by averaging the guesses of three stream processing experts. We used questionnaires to collect the time spent by each of the participants to parallelize the applications. In addition, we used the Mean Magnitude Relative Error (MMRE) and the Percentage Relative Error Deviation (PRED) [25] to assess the accuracy.

V. Results

Table I shows the average results of the 15 participants for SLOC and the actual development time for FastFlow, SPar, and TBB. From these results, we can assume that SPar requires less effort to develop parallel stream processing applications than the other PPMs. However, since the averages alone cannot determine which PPM offers the best productivity, further analysis of the results with a hypothesis test is required. This study aims to evaluate the accuracy of the effort estimation methods, so productivity analysis is outside this study's scope.

Table I also shows the average development times estimated by each of the models used in this study, which have been converted into development hours. To compare these results with the actual development time we used the MMRE and PRED metrics (Table II). Our results show that the Planning Poker got the best result than the other estimation metrics from the MMRE and PRED values analysis. However, the estimated value is considered acceptable only for TBB. According to Port and Korte [25], MMRE less than or equal to 0.25, and PRED greater than or equal to 0.75 are values considered an acceptable accuracy level for models and effort estimation.

[4]Available at: http://calvados.di.unipi.it/

[5]Available at: https://gmap.pucrs.br/spar-wiki/

[6]Available at: https://dwheeler.com/sloccount/.

[7]Available at: https://w3.cs.jmu.edu/bernstdh/web/common/webapps/oop/fpcalculator/FunctionPointCalculator.html

[8]Available at: http://groups.umd.umich.edu/cis/tinytools/cis375/f17/team9-use-case-pts/Use_Case_Point_Calculator/

[9]Available at: https://galorath.com/seer-for-software/.

TABLE I
Results for the number of SLOC, ASLOC, and MSLOC, as well as, the actual and estimated development times (in hours).

Code Metric	Sequential	FastFlow	SPar	TBB
SLOC	75	108.27	79.87	117.53
ASLOC	-	33.40	4.87	42.67
MSLOC	-	20.67	1.67	20.00
Actual development time	-	1.72	1.01	2.17
COCOMO II (post-archit.)	264.34	570.44	525.75	582.90
COCOMO II (early design and post-archit.)	198.24	550.12	507.74	562.11
COCOMO II maint. model	-	506.81	287.41	526.80
COCOMO II reuse model	-	428.50	253.15	454.30
PCRM	-	447.32	267.69	473.57
FP	-	127.54	101.62	137.32
PHalstead	-	10.73	7.16	10.83
Planning poker	-	0.50	0.50	2.00
Putnam's model	-	2.51	2.00	2.68
SEER-SEM	31.00	58.07	42.60	63.33
UCP	-	162.06	129.72	175.14

Putnam's model showed the second-best result. The time estimated by Putnam's model is close to the real time, although the estimated values did not meet the criteria of the accuracy metrics (MMRE $\geq$ 1.19 and PRED $\leq$ 0.07). This model can be useful for evaluating parallel applications, but there is a limitation. It uses the PP obtained through the development effort of previously developed applications. However, there is currently no database composed of such applications.

PHalstead's development effort was the third-best result, although it does not meet the accuracy criteria (MMRE $\geq$ 7.22 and PRED $=$ 0). It considers only the TOC in its evaluation, without taking into account any of the factors that impact the development effort of parallel applications. So, this metric can be useful for measuring code size, as well as the SLOC.

FP and UCP showed similar results because these models were designed to evaluate user interaction systems in which there is an interface where users enter, delete, and query data. Neither considers any aspect of the programming language used, so they are unsuitable for evaluating PPMs.

The traditional COCOMO II (post-architecture alone, and early design and post-architecture together) showed the worst results. In our previous work [16], we have seen that traditional COCOMO II estimates the effort required to develop an application from scratch, so these results were already expected (MMRE $\geq$ 427.99 and PRED $=$ 0). To address this limitation, we used the COCOMO II maintenance and reuse models because the parallel applications were implemented from a sequential application rather than from scratch.

TABLE II
Accuracy results for the development times estimated by each of the code metrics evaluated using MMRE and PRED metrics.

Accuracy	FastFlow		SPar		TBB	
	MMRE	PRED	MMRE	PRED	MMRE	PRED
COCOMO II (post-archit.)	533.68	0.00	600.63	0.00	442.27	0.00
COCOMO II (early design and post-archit.)	511.71	0.00	577.85	0.00	427.99	0.00
COCOMO II maint. model	472.12	0.00	330.96	0.00	398.90	0.00
COCOMO II reuse model	398.97	0.00	291.19	0.00	344.74	0.00
PCRM	416.46	0.00	307.65	0.00	358.94	0.00
FP	133.02	0.00	115.25	0.00	102.61	0.00
PHalstead	9.03	0.00	7.22	0.00	7.33	0.00
Planning poker	0.71	0.00	0.50	0.00	0.08	1
Putnam's model	1.48	0.20	1.29	0.07	1.19	0.27
SEER-SEM	53.38	0.00	47.79	0.00	46.97	0.00
UCP	169.29	0.00	147.40	0.00	116.07	0.00

The COCOMO II reuse model showed better results compared to the maintenance model. There is still a big difference between the development time estimated by the reuse model and the actual development time. Then we proposed the PCRM approach, removing the cost drivers and scale factors not applied to parallel application development. However, this increased development time because adapting existing models to the parallel programming domain is a complex task. It is not possible to just remove some of their parameters.

SEER-SEM also considers several factors that impact the software development cycle in its evaluation. SEER-SEM also uses a database of already developed software to calibrate its parameters. There are video processing applications in its database, but not parallel applications. This may be one of the reasons why the model also fails to estimate a development time close to the real one (MMRE $\geq$ 42.60 and PRED $=$ 0).

VI. Threats to Validity

This study has some threats to validity. The learning effect is a threat to internal validity, because of the order in which the PPMs are used by the participants. Therefore, the time spent by the participants developing applications using FastFlow may have been affected since two groups already knew the target problem before using it. In addition, there is no control group, since the three groups used all three PPMs (construct validity). The study design is a threat to construct validity because SPar is an annotation-based PPM, unlike FastFlow and TBB. The application size is another threat to construct validity, although is a standard application in real-world stream processing. In addition, the sample size of 15 participants and the participants' experience level are threats to the conclusion validity. Therefore, the results can not be generalized.

VII. Conclusion and Future Work

PHalstead has proven to be a useful metric for evaluating parallel code, although it does not get the best results. In future work, we aim to extend PHalstead to consider other PPMs, such as SkePU, and Pthreads. The Planning Poker showed the best results because it relies on the experts' opinions to guess the development effort. Despite the promising results, it is not easy to find professionals to apply this method in practice.

Putnam's model has proven to be a suitable method for estimating development effort for parallel applications. Although the SEER-SEM method did not show the best result, it also uses a similar technique to calibrate its parameters. There are several public domain datasets available in the software engineering area to evaluate the effort estimation models. However, in the parallel programming domain, no such data set is available, making it difficult to use and evaluate these techniques. As future work, we aim to compose a dataset regarding the development effort of parallel applications. So, it will be possible to evaluate the accuracy of predictive models.

Despite our efforts, the PCRM was not suitable for parallelization using high-level PPMs, since the estimated effort is much higher than the actual effort. Moreover, it is not easy to refine its parallel application development scenario parameters. Therefore, creating a parallelism-sensitive model to evaluate applications in this domain is necessary since its development involves factors that are not addressed by the models considered in this study.

Acknowledgment

This research is partially funded by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, FAPERGS 05/2019-PQG project ParAS (Nº 19/2551-0001895-9), FAPERGS 10/2020-ARD project SPar4.0 (Nº 21/2551-0000725-7), and Universal MCTIC/CNPq Nº 28/2018 project SParCloud (Nº 437693/2018-0).

References

[1] D. B. Kirk and W. W. Hwu, *Programming massively parallel processors: A hands-on approach*. Morgan Kaufmann, 2016.
[2] ISO 9241-11:2018, "Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts," *ISO*, 2018.
[3] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer, 2012.
[4] G. Rodriguez-Canal, Y. Torres, F. J. Andújar, and A. Gonzalez-Escribano, "Efficient heterogeneous programming with fpgas using the controller model," *The Journal of Supercomputing*, vol. 77, no. 12, pp. 13 995–14 010, 2021.
[5] M. A. Martínez, B. B. Fraguela, and J. C. Cabaleiro, "A highly optimized skeleton for unbalanced and deep divide-and-conquer algorithms on multi-core clusters," *The Journal of Supercomputing*, pp. 1–21, 2022.
[6] B. Peccerillo and S. Bartolini, "Flexible task-DAG management in PHAST library: Data-parallel tasks and orchestration support for heterogeneous systems," *Concurrency and computation: Practice and experience*, vol. 34, no. 2, pp. 1–20, 2022.
[7] D. Griebler, D. Adornes, and L. G. Fernandes, "Performance and usability evaluation of a pattern-oriented parallel programming interface for multi-core architectures," in *SEKE 2014*. KSIGS, 2014, pp. 25–30.
[8] S. Memeti, L. Li, S. Pllana, J. Kołodziej, and C. Kessler, "Benchmarking opencl, openacc, openmp, and cuda: programming productivity, performance, and energy consumption," in *ARMS-CC '17*, 2017, pp. 1–6.
[9] V. Pankratius, F. Schmidt, and G. Garreton, "Combining functional and imperative programming for multicore software: An empirical study evaluating scala and java," in *ICSE 2012*. IEEE, 2012.
[10] J. Fresno, D. Barba, A. Gonzalez-Escribano, and D. R. Llanos, "Hitflow: A dataflow programming model for hybrid distributed-and shared-memory systems," *International Journal of Parallel Programming*, vol. 47, no. 1, pp. 3–23, 2019.
[11] J. Fernàndez-Fabeiro, A. Gonzalez-Escribano, and D. R. Llanos, "Simplifying the multi-gpu programming of a hyperspectral image registration algorithm," in *HPCS 2019*. IEEE, 2019, pp. 11–18.
[12] J. Miller and M. Arenaz, "Measuring the impact of hpc training," in *EduHPC 2019*. IEEE, 2019, pp. 58–67.
[13] T.-W. Huang, G. Guo, C.-X. Lin, and M. D. Wong, "Opentimer v2: A new parallel incremental timing analysis engine," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 40, no. 4, pp. 776–789, 2020.
[14] V. Narayanan, R. Kavitha, and R. Srikanth, "Performance evaluation of brahmagupta-bhaskara equation based algorithm using openmp," in *ICDAM 2021*. Springer, 2022, vol. 1, pp. 21–28.
[15] S. Wienke, J. Miller, M. Schulz, and M. S. Müller, "Development effort estimation in hpc," in *SC'16*. IEEE, 2016, pp. 107–118.
[16] G. Andrade, D. Griebler, R. Santos, M. Danelutto, and L. G. Fernandes, "Assessing coding metrics for parallel programming of stream processing programs on multi-cores," in *SEAA 2021*. IEEE, 2021, pp. 291–295.
[17] L. H. Putnam and W. Myers, *Measures for excellence: reliable software on time, within budget*. Prentice Hall PTR, 1991.
[18] M. H. Halstead, *Elements of software science*. Elsevier, 1977.
[19] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software cost estimation with COCOMO II*. Prentice Hall PTR, 2000.
[20] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," *Transactions on software engineering*, vol. 9, no. 6, pp. 639–648, 1983.
[21] G. Karner, "Resource estimation for objectory projects," *Objective Systems SF AB*, vol. 17, pp. 1–9, 1993.
[22] Galorath, "Seer for software," 2022, Source: https://galorath.com/products/seer-for-software/, March 2022.
[23] M. Cohn, *Agile estimating and planning*. Pearson Education, 2005.
[24] OpenCV, "Creating a video with OpenCV," Source: https://docs.opencv.org/2.4/doc/tutorials/highgui/video-write/video-write.html, March 2022.
[25] D. Port and M. Korte, "Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research," in *ESEM '08*, 2008, pp. 51–60.