Teaching Performance Metrics in Parallel Computing Courses

Sandino Vargas-Pérez
Department of Computer Science
Kalamazoo College
Kalamazoo, MI, USA
Email: sandino.vargasperez@kzoo.edu
https://orcid.org/0000-0002-5778-6142

***Abstract*—As high-performance computing technologies advance, the significance of parallel programming in various domains is becoming increasingly evident since it allows us to harness the power of heterogeneous computing and solve complex problems more efficiently. However, for students to master this type of computation and be able to apply it in different contexts, it requires understanding how measuring and optimizing parallel code impacts its performance. This paper presents an approach to enhancing students' comprehension of parallel performance metrics through an interactive exercise that complements lectures on parallel performance and improves assessment.**

***Index Terms*—Parallel Computing, HPC, Undergraduate Education,**

I. Introduction

Educators in the field of Parallel Computing often encounter challenges when teaching students about performance metrics [1]. Measuring and analyzing parallel programs requires a solid grasp of various performance metrics, such as speedup, efficiency, and scalability. With a thorough understanding of these concepts, students can identify and address bottlenecks and inefficiency in their code. Measuring performance also helps students optimize resource utilization, ensuring the available processing power is effectively used.

Moreover, CS students in undergraduate Liberal Arts institutions, such as Kalamazoo College, often double-major or minor in other fields, where they often tackle problems by applying computational methods. A comprehensive understanding of Parallel Computing and program performance enables them to effectively utilize computing resources that can handle large datasets, resource-intensive simulations, or complex calculations and to gain a richer insight into the given problem [2].

Another challenge is the issue of scalability in the era of Big Data [3]. Understanding performance measurements allows students to make informed decisions about the impact of scaling their parallel programs to accommodate larger datasets, more powerful systems, or greater user demands.

This paper presents an interactive exercise that facilitates a holistic comprehension of parallel performance metrics and helps address the challenges mentioned above. In the exercise, students evaluate the parallel performance of a program and are encouraged to think critically about the relationship between program design, execution efficiency, and performance. Two learning outcomes related to measurements of parallel runtime, speedup, and efficiency, as well as understanding the relationship between problem size and the number of processing units, were identified for assessment. The assessment of these learning outcomes was done using both quantitative and qualitative data collected through written assessments. The results indicated that the exercise improved students' understanding of these concepts and that they gained an appreciation for the importance of performance analysis in parallel computing.

II. Related Work

Significant efforts have been made to incorporate parallel and distributed systems concepts into the undergraduate computer science curriculum [4], [5], [6], [7], ranging from module-based introductions of parallel concepts to activity-based and interactive approaches. However, conveying the complex interplay of factors affecting system performance, such as speedup, efficiency, scalability, and overhead, is a significant challenge that has yet to be found as the sole focus of a proposed approach.

Various pedagogical approaches have been developed to teach parallel performance metrics as part of a broader curriculum, ranging from hands-on lab exercises using parallel computing platforms like clusters of Raspberry Pi computers [8], [9], [10], or cloud-based resources [8], and software simulations [11]. Nevertheless, most research on parallel computing education focuses on the effectiveness of different approaches such as project-based learning (PBL), parallel programming patterns, and parallel computing educational games [12].

The cost-effectiveness of the hardware used for teaching parallel computing is also a significant area of research, with single-board computer clusters and cloud services like Google Colab being the most extensively studied methods. To the author's knowledge, no study has emphasized the importance of teaching parallel performance metrics and measurements to motivate and keep students engaged in the learning process.

III. Methodology

In the latest iteration of the 400-level course "COMP 481 – Parallel Computing" at Kalamazoo College, a new interactive exercise was introduced to the students to improve

their understanding of parallel performance metrics since the primary goal of writing these programs is to compare their performance to that of their sequential counterparts [13]. The Computer Science department offers this course every other academic year[1], and it is designed to blend interactive lectures and readings with group or individual work and learning assessment activities such as short quizzes, in-class exercises, and mini-labs to put the theory into practice. The course concludes with a comprehensive final project or research presentation.

The Parallel Performance module of this course [14], scheduled for the second week of the term, starts with a lecture to introduce parallel programming metrics and runtime measurements, encompassing the following topics:

- Parallel performance metrics.
- Measuring runtime in parallel programs.
- Speedup.
- Efficiency.
- Scalability.
- Amdahl's Law.

To prepare for the lecture and encourage participation, students read the corresponding book sections [13] [15] and complete a set of discussion questions that asked them to define concepts related to the topic. The interactive exercise is given right after the lecture and discussions. This approach gives students confidence in solving the exercise and solidifies the importance of theoretical foundations for their learning experience.

A. "Measuring Performance": Interactive Parallel Performance Exercise

The exercise is laid out on a Jupyter Notebook and contains two simple implementations of a histogram program using C++. First, a sequential version is provided where they can see the algorithm in action: it creates nine datasets of size $N = 10^k$, where $k = 1, 2, 3, ..., 9$. The datasets are randomly populated with integers between 0 and 9. The sequential program will count and store the frequencies of each integer in 10 bins. Lastly, the program will output the sequential runtime to generate the datasets and calculate their histograms. A sample output of the execution can be seen in Fig. 1.

Second, a parallel version of the histogram coded in OpenMP. At this point in the course, students have yet to be exposed to the different APIs used in parallel computing, so they are reminded of the exercise's emphasis, which is to put into practice the theoretical knowledge acquired during the lecture, reading assignments, and discussion questions. The parallel histogram will calculate the histogram of 9 datasets (same as its sequential counterpart), using an increasing number of threads (P), where the values of P are powers of 2 going from 1 to 512 threads. Finally, the program prints a comma-separated list with the parallel execution time of each run, e.g., dataset $N = 10$ will be executed using

[1]For context, Kalamazoo College divides the academic year into three terms of ten weeks each.

```
In [4]: !./histogram

        Processing histogram sequentially...
          calculating histogram for 10 elements
          calculating histogram for 100 elements
          calculating histogram for 1000 elements
          calculating histogram for 10000 elements
          calculating histogram for 100000 elements
          calculating histogram for 1000000 elements
          calculating histogram for 10000000 elements
          calculating histogram for 100000000 elements
          calculating histogram for 1000000000 elements

        Done!
        Printing results:

        n (size)    time (s)
        10          2e-06
        100         3e-06
        1000        2.8e-05
        10000       0.000205
        100000      0.002077
        1000000     0.012784
        10000000    0.109844
        100000000   1.09698
        1000000000  10.8249
```

Fig. 1. Output after execution of sequential histogram program.

$P = 1, 2, 4, ..., 512$. Fig. 2 shows a snippet of the parallel code. Students use the comma-separated list containing the resulting runtime and generate graphs to show the execution times of each combination of threads P and problem size N. This part of the exercise helps students visualize the effects of adding processing units as the problem size increases.

The Jupyter Notebook containing the exercise is hosted in *Jigwe*[2], a high-performance workstation fitted with two Intel Xeon Platinum processors at 2GHz (52 cores total), 192 GB of DDR4 memory, and an NVIDIA Quadro P6000 GPU with 3,840 CUDA cores. This equipment is sufficient (but should be a baseline) for a class of approximately 20 students working and interacting with the assignment, concurrently playing around with different input and resource configurations. The exercise instructs them to execute the parallel code, for example, by setting a range of threads P beyond the HPC system's capabilities, generating dataset inputs of various sizes N, or fixing either N or P to observe the effects on scalability.

Finally, students create a report detailing their experience working with the exercise, according to the following instructions:

1) Create three separate tables containing your calculated speedup and efficiency for three of the datasets (size $N = 10, 100000, 1000000000$).
2) Write about your experience executing the program and calculating the Speedup and Efficiency, more specifically:
   - What can you say about the T_{par} obtained with respect to the T_{seq}?
   - What can you say about the speedup (S) obtained?

[2]*Jigwe* (pronounced cheeg-weah) is the Pottawatomi word for Thunderbird, as spelled by the Gun Lake Tribe, members of the Match-E-Be-Nash-She-Wish Band of Pottawatomi Indians of Michigan. With the tribe's permission, this name honors the Indigenous people who inhabited the land on which Kalamazoo College currently stands.

```
// Execute each dataset size (N) with each number of threads (P)
while (threads <= MAX_THREADS){
    cout << "  Using " <<  threads << " threads" << endl << endl;

    n = 10;
    while (n <= MAX_SIZE)
    {
        int t_histogram[BINS] = {0};

        // start timer
        double start_time = omp_get_wtime();

        #pragma omp parallel num_threads(threads)
        {
            // Initialize a seed for each individual thread.
            unsigned int seed = 42 + omp_get_thread_num();
            int t_point;

            // Generate random data points from 0 to 9
            #pragma omp for
            for (int i = 0; i < n; i++) {
                t_point = rand_r(&seed) % 10;
                data[i] = t_point;
            }

            // Organize data by placing data points
            // into corresponding bins
            #pragma omp for reduction(+:t_histogram[:10])
            for (int i = 0; i < n; ++i) {
                t_point = data[i];
                t_histogram[t_point]++;
            }
        }
        // stop timer
        double duration = omp_get_wtime() - start_time;
        par_durations[row][col] = duration;

        row++;
        // Next dataset size
        n *= 10;
    }

    // Next number of threads
    threads *= 2;
    row = 0;
    col++;

}
```

Fig. 2. Code snippet of the parallel version of histogram code.

- Is there a combination of processing units (threads, P) that resulted in the best speedup?
- Did you observe a superlinear speedup? If so, what do you think it means?
- What can you say about the efficiency (E) obtained?
- Is there a combination of processing units (threads, P) that resulted in the best efficiency?
- Was the efficiency closed to 1 at all times, or did it decrease as more processing units were added?
- Did you observe $E > 1$? If so, what do you think it means?

3) Optionally, choose the dataset $N = 1000000000$, and with your speedup (S) results, plot a line chart: "Speedup vs. Processing Units," placing in the Y axis your speedup values and in the X axis the number of processing units or threads (P) used. With your efficiency (E) values, create a second line chart, "Efficiency vs. Processing Units," with the Y axis containing the efficiency (E) and the X axis the number of P's used. Add the "ideal case" for reference ($S = P$ and $E = 1$) on each chart.

Two learning outcomes were identified to assess the student's written report (Bloom's Taxonomy shown in brackets):

- Students demonstrate an understanding of how to take parallel runtime and calculate the speedup and efficiency of a parallel program [Knowledge].
- Students demonstrate an understanding of how the size (N) and the number of processing units (P) affect the performance of a parallel program and its scalability [Comprehension].

B. Quiz on Parallel Performance

After the report was submitted, evaluated, and discussed with the class, a quiz was given to assess the learning outcomes established for the exercise. The quiz has three main problems to solve and an optional bonus point. The first problem evaluates students' ability to calculate speedup and efficiency, and the second and third evaluate their comprehension of scalability and parallel performance:

1) Suppose you have parallelized an algorithm with sequential runtime $T_{seq} = N$ and the theoretical parallel runtime is $T_{par} = N/P + \log_2(P)$. Calculate the speedup S of your parallel algorithm and its efficiency E if the problem size is $N = 65,536$ with processing units $p = 2, 4, 8, 16, 32, 64$.
2) Based on your efficiency E results above as P doubles and N stays fixed, what can you say about the scalability of this algorithm?
3) Based on the relation of the obtained Speedup S and P, what kind of speedup is this algorithm exhibiting?

The bonus point deals with Amdahl's Law. Students must develop a formula that better represents T_{par} and S, given that the parallel algorithm in the first question has 20% of its code always running sequentially.

IV. Assessment and Analysis

The Computer Science department at Kalamazoo College has offered "COMP 481 – Parallel Computing" three times as of the academic year 2022-2023. The first offering was in the Winter term of 2019, the second in the Winter term of 2021 (during the COVID-19 global pandemic, online), and the third in the Spring term of 2023. The interactive exercise was introduced in the last offering (Spring 2023). For context, this course is mainly offered to Seniors. The gender distribution in each class was as follows:

1) **Winter 2019** 20% female, 80% male, 0% non-binary (or gender non-conforming).
2) **Winter 2021** 16% female, 84% male, 0% non-binary (or gender non-conforming).
3) **Spring 2023** 26% female, 74% male, 0% non-binary (or gender non-conforming).

Table I shows the grade distribution of the assessment quiz on parallel performance. The grades were normalized and grouped into categories. The total weight of the quiz was 10 points plus two bonus points (not included in the grades shown in Table I). Students improved their grades

TABLE I
STUDENTS' GRADE DISTRIBUTION OF THE PARALLEL PERFORMANCE QUIZ IN EACH COURSE OFFERING (QUIZCO)

Grade	QuizCO1	QuizCO2	QuizCO3
10	3	3	14
9	6	15	0
8	4	1	3
7	3	0	1
6	3	0	0
5	1	0	1

significantly during the spring term of 2023 (QuizCO3) by introducing the parallel performance exercise. Approximately 74% of this class achieved a perfect score of 10 compared to the previous offerings (QuizCO1 and QuizCO2). This suggests that the exercise helped the Spring 2023 cohort perform better. However, it is worth mentioning that the class of Winter 2021 took the quiz as a take-home, open-book assessment, having 24 hours to complete it. In this modality and given the accommodations granted to the students by the disruptive nature of remote learning, students did very well, most of them (around 79%) scoring 9 out of 10.

Nevertheless, 79% of Spring 2023 students attempted the bonus point successfully, compared to 53% in Winter 2021. All of the students during the Winter 2019 course offering attempted the bonus portion of the quiz, but around 50% did so successfully.

While the data suggests a positive impact, additional analysis or a controlled study design might be necessary to establish a causal relationship between the introduction of the exercise and the improved performance. Other factors could also contribute to the observed quantitative differences.

After completing the parallel performance exercise, a qualitative assessment was taken based on the student's written report in Spring 2023. Most students demonstrated a great deal of understanding regarding their calculations of speedup and efficiency. A student noticed: "*Executing the program and calculating the Speedup and Efficiency has demonstrated the concepts we talked about in class, allowing me to understand the concepts further. For example, in class we talked about how adding more processing units doesn't always guarantee faster Speedups and that the added overhead can actually cause it to run slower. When analyzing the data given by the program, I saw evidence of this situation being true, because while we would expect the speedup to be greater with a greater number of processors being used [...] we can see that once we surpass 16 processors for a problem of size n = 100000, the Speedup slows down considerably.*"

Another student talked about the parallel performance regarding the smallest dataset: "*From these results, I can see that T_{par} always improved compared to T_{seq}. For the smaller problem sizes, it seems that increasing P decreases the Speedup. This is probably due to the significant effect of overhead for adding new threads to compute such small amount of data points. On the other hand, increasing P dramatically increased the Speedup with the larger problem sizes.*"

TABLE II
STUDENT'S MEASUREMENTS OF PARALLEL RUNTIME AND PERFORMANCE CALCULATIONS FOR A DATASET OF SIZE $N = 1,000,000,000$ AND $T_{seq} = 7.87894$ SECS

P	T_{par} (secs)	S	E
2	4.00123	1.969129493	0.984564746
4	2.06517	3.815153232	0.953788308
8	1.29542	6.082150963	0.760268870
16	0.798353	9.868992789	0.616812049
32	0.497659	15.83200545	0.494750170
64	0.311599	25.28551119	0.395086112
128	0.272777	28.88418012	0.225657657
256	0.259524	30.35919607	0.118590609
512	0.244141	0.030986528	0.000060520

Reflecting on the speedup obtained in their calculations, one student noted "*$S = P$ implies perfect scalability; when $N = 100,000$ and $N = 1,000,000,000$, the Speedup is close to the number of processors, especially when $N = 1,000,000,000$ and $P = 2$. None of S exceeded P, therefore, they did not go beyond the limit of speedup, in other words, there is no superlinear speedup. When $P = 32$, the Speedup started making a big gap from P. It can be because the overhead is too high or the load balance between processors is poor.*"

Table II shows the calculations performed by one student. Answering the report questions "Was the efficiency closed to 1 at all times, or did it decrease as more processing units were added? Did you observe $E > 1$? If so, what do you think it means?", the student wrote "*Based on my data as P increased, the efficiency decreased and did not always stay close to 1. I didn't observe that $E > 1$. Which could mean that the speedup achieved by the parallel execution is less than the ideal $S = P$, because I think that to observed $E > 1$ you need superlinear speedup. My data is probably inefficient in the parallel execution for $P = 512$, I think the server was probably overloaded.*" Here, the student refers to *Jigwe* being asked to use more processing units than physically available (it maps each computing core to a thread) and having to share those resources with the rest of the students in the class. Fig. 3 shows the line chart the student built to visualize the calculated efficiency E.

Overall, students in this class (Spring 2023) showed a great understanding of parallel performance measures, which continued to show beyond this course module. Their Parallel Computing vocabulary, the usage of correct terminologies, and the ability to formulate better questions significantly improved compared to the cohorts in previous course offerings. They kept applying what they learned to different assignments in the class. For example, an assignment later in the term tasked

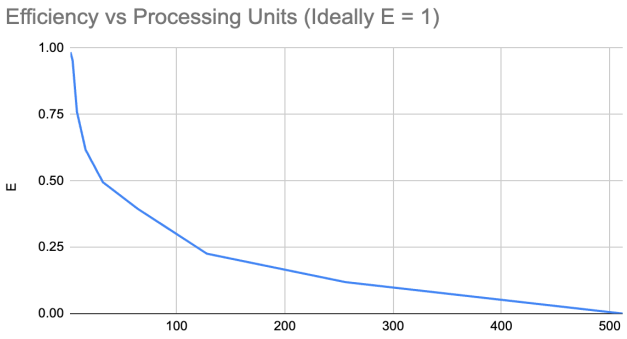

Fig. 3. Student's line chart representing the efficiency E vs. Processing Units P for a dataset of size $N = 1,000,000,000$

them with parallelizing a sequential sorting algorithm using OpenMP. In their attempt, many students identified issues based on the obtained speedups and efficiency, ranging from race conditions and the overuse of critical sections to declaring shared variables that should have been private, recognizing bottlenecks in concurrent memory access by multiple threads in their program.

Furthermore, after the conclusion of the Spring 2023 course offering, several students pursued research and independent study opportunities with other faculty members, as well as internships where they applied parallel computing to accelerate tasks, evidencing that they can apply the skills they have learned to their work in different contexts. It is worth mentioning:

- A Math and Computer Science double-major student presented a research talk titled "Computational Techniques for Finding Virus Transitions That Preserve Icosahedral Symmetry" at the 2023 West Michigan Regional Undergraduate Science Research Conference (WMURUGS). A snippet of the abstract states: "*Finding [virus] transitions requires a large amount of computation for which we have employed a cluster and parallel computing techniques. We have reproduced previously discovered transitions for the Cowpea Chlorotic Mottle Virus that preserve 2 and 3-fold symmetry and more importantly have created a comprehensive list of what symmetries can be preserved between any possible combination of the 55 standard point arrays and their combinations.* "
- A Computer Science student worked with the author in the summer of 2023, running experiments and measuring the runtime performance of an updated version of a parallel program [16] [17] that is capable of accessing compressed data without the need for a complete decompression. Besides helping identify bottleneck issues in the program, the student designed, implemented, and tested several of the program's core functions. This project is under development and will soon be submitted to a peer-reviewed journal.
- A student completed an internship during the summer of 2023 where they had the opportunity to identify and implement a parallel solution for one of the company's existing projects. Their experience served as the basis for their Senior Integrated Project (SIP) titled "Engineering and Parallelizing Synthetic Aperture Radar (SAR) Image Environment Viewing Engine (SIEVE)". A snippet of the abstract states: "*In the summer of 2023 spanning the months of May – August, I had the pleasure of working for Maxar Technologies (Maxar), a company in the GEOINT and Space sector.[...] These formats are used in practice for extremely large images with lots of metadata as well. This caused a few issues in the speed and efficiency of the program when loading in the images for analysis. Prior to this summer I engaged in coursework relating to using parallelization techniques to maximize efficiency in programs. This is was the motivation to utilize this opportunity and conduct research on parallelizing these loading processes. Through this, there were not drastic improvements to the speed and efficiency of the program, but there were improvements, nonetheless.* "

V. Conclusion

The data analysis shows significant improvement in students' performance and comprehension of parallel computing performance after introducing an interactive exercise that allows them to play around with different configurations of problem size and processing units. The quantitative and qualitative assessments support the exercise's positive impact on students' communication skills and critical thinking. It is essential to emphasize that while these observations point towards a positive influence, establishing a causal relationship between the introduction of the exercise and the improved performance demands more rigorous analysis.

With the observations provided in earlier sections and based on previous course offerings, anecdotal evidence, and informal conversations with students, the author encourages other computer science educators to prioritize teaching parallel performance metrics in Heterogeneous Computing courses. Doing so will encourage students to learn more and explore the possibilities of applying parallel computing solutions to various computational problems, particularly in the Big Data and Artificial Intelligence realms where many CS students are heading. The broader implications of these findings are evident in students' continued application of parallel performance knowledge beyond the scope of the course.

Acknowledgment

The author wants to thank all the COMP481 students across the years, for their dedication to learning and their interest in Parallel Computing.

References

[1] R. Muresano, D. Rexachs, and E. Luque, "Learning parallel programming: a challenge for university students," *Procedia Computer Science*, vol. 1, no. 1, pp. 875–883, 2010.

[2] R. K. Raj, C. J. Romanowski, J. Impagliazzo, S. G. Aly, B. A. Becker, J. Chen, S. Ghafoor, N. Giacaman, S. I. Gordon, C. Izu *et al.*, "High performance computing education: Current challenges and future directions," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, 2020, pp. 51–74.

[3] C. Dobre and F. Xhafa, "Parallel programming paradigms and frameworks in big data era," *International Journal of Parallel Programming*, vol. 42, no. 5, pp. 710–738, 2014.

[4] A. Qasem, D. P. Bunde, and P. Schielke, "A module-based introduction to heterogeneous computing in core courses," *Journal of Parallel and Distributed Computing*, vol. 158, pp. 56–66, 2021.

[5] D. J. Conte, P. S. L. de Souza, G. Martins, and S. M. Bruschi, "Teaching parallel programming for beginners in computer science," in *2020 IEEE frontiers in education conference (FIE)*. IEEE, 2020, pp. 1–9.

[6] 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, keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731517300163

[7] P. Chitra and S. K. Ghafoor, "Activity based approach for teaching parallel computing: An indian experience," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 290–295.

[8] Z. Xu, "Teaching heterogeneous and parallel computing with google colab and raspberry pi clusters," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 308–313. [Online]. Available: https://doi.org/10.1145/3624062.3624095

[9] E. Shoop, R. Brown, J. Adams, and S. Matthews, "Teaching distributed computing fundamentals using raspberry pi clusters," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*, ser. SIGCSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1201. [Online]. Available: https://doi.org/10.1145/3478432.3499161

[10] A. A. Younis, R. Sunderraman, M. Metzler, and A. G. Bourgeois, "Developing parallel programming and soft skills: A project based learning approach," *Journal of Parallel and Distributed Computing*, vol. 158, pp. 151–163, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731521001611

[11] R. Tanaka, R. F. da Silva, and H. Casanova, "Teaching parallel and distributed computing concepts in simulation with wrench," in *2019 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*. IEEE, 2019, pp. 1–9.

[12] J. Zhu, K. Alderfer, A. Furqan, J. Nebolsky, B. Char, B. Smith, J. Villareale, and S. Ontañón, "Programming in game space: how to represent parallel programming concepts in an educational game," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, pp. 1–10.

[13] P. Pacheco, *An Introduction to Parallel Programming*, 1st ed. Elsevier, 2011.

[14] Computer Science Department. COMP-481: parallel computing. HTML. [Online]. Available: http://www.cs.kzoo.edu/cs481/schedule.html

[15] R. Trobec, B. Slivnik, P. Bulić, and B. Robič, *Introduction to parallel computing: from algorithms to programming on state-of-the-art platforms*. Springer, 2018.

[16] S. Vargas-Pérez and F. Saeed, "A hybrid mpi-openmp strategy to speedup the compression of big next-generation sequencing datasets," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2760–2769, 2017.

[17] ——, "Scalable data structure to compress next-generation sequencing files and its application to compressive genomics," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2017, pp. 1923–1928.