Peachy Parallel Assignments (EduPar 2023)

Alina Lazar
Department of SCSIET
Youngstown State University
Youngstown, OH, USA
alazar@ysu.edu

Virginia Niculescu
Department of Computer Science
Babeş-Bolyai University
Cluj-Napoca, Romania
virginia.niculescu@ubbcluj.ro

David P. Bunde
Department of Computer Science
Knox College
Galesburg, IL, USA
dbunde@knox.edu

Abstract—**The presentation of Peachy Parallel Assignments at parallel and distributed computing education workshops is an effort to promote the reuse of high-quality assignments, both saving precious faculty time and improving the quality of course assignments. These assignments must have been used in class and are selected for being easy to adopt by other instructors and for being "cool and inspirational" so that students spend time on them and talk about them with others. The assignments and their materials are also archived on the Peachy Parallel Assignments website.**

In this paper, we present two new assignments. The first has students implement the Mandelbrot set in Python, combining an interesting image with Python's ease of use. The second assignment is a substantial project to implement a programming contest judge. It requires that students use many parallel and distributed computing concepts, with the added benefit of solving a "real problem" and creating software with which students may have personally interacted.

Keywords-**Peachy Parallel Assignments, Parallel computing education, Parallel programming, Curriculum Development, Mandelbrot set, Programming contest**

I. Introduction

The Peachy Parallel Assignments effort aims to promote the development and spread of high-quality homework assignments and laboratory exercises to teach concepts in Parallel and Distributed Computing (PDC). We specifically seek assignments that are not only well-designed, but also motivating for students to encourage them to spend time working on the assignment and also spread positive messages about the field by excitedly showing the results to their friends, roommates, and family members. Developing such assignments is not easy, requiring both creativity and hard work. Our goal is to recognize this effort and also, by sharing the assignments themselves, to allow other instructors to leverage this effort through reuse.

With these goals in mind, Peachy Parallel Assignments are solicited with a public call for submissions and selected competitively using the following criteria:

- Tested: All assignments must have been successfully used with real students
- Adoptable: The assignments must be useful to other instructors, with clear descriptions and the resources needed for adoption by others (handouts, given code, references for more information, etc.). Ideally, they focus on core PDC topics using widely-used languages and toolsets, with suggested customizations that can make them suitable for students at a variety of levels.
- Cool and inspirational: The assignments must motivate students through the artifacts they create (e.g., images) or the concepts taught. Ideally, students should want to talk about the assignment with friends and show it off to others.

Assignments selected as Peachy Parallel Assignments join a series published at the Edu* (i.e., EduPar and EduHPC) workshops, e.g., [6], [10], [11]. The assignments are also archived at the Peachy Parallel Assignments webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy), along with all the materials needed to adopt them. The assignments are meant to be used as-is or adapted by other instructors to fit the context of their class. The assignments can also serve as inspiration for other assignments.

This paper describes two new Peachy Parallel Assignments selected for presentation at EduPar 2023. Section II describes an assignment to create a representation of the Mandelbrot set in parallel. This task has been used in assignments previously [7] because it is visually interesting but this version improves on prior work by better integrating visualization and also demonstrating the use of Python, which is seeing increased usage in parallel and high-performance computing contexts. It is also widely used in introductory Computer Science courses, allowing the assignment to be used very early in the CS curriculum. Section III describes a larger assignment that requires students to combine many PDC concepts to create a server to judge programming contest problem submissions. Unlike most programs developed for programming assignments, this is practical software that students might have already used, giving it a "real-world" appeal.

II. Parallel Mandelbrot Set Implementation

In this section, we present a parallel programming assignment that is suitable for an introductory programming course and, with minor modifications, also well-suited for upper-division undergraduate courses. The assignment asks students to execute sequential and parallel

Python code to produce an image of the Mandelbrot set. Python is a very popular programming language and is widely used in introductory programming courses. To support parallel computing, we use IPython Parallel, a Python library that provides a simple and flexible way to execute code in parallel across multiple processors or machines. It is well-suited for prototyping and experimentation, and provides standard APIs compatible with many other implementations. By utilizing IPython Parallel framework and mpi4py, students can gain a deeper understanding of parallel and distributed processing. Moreover, this task provides students with a fascinating example to grasp the basics of parallel computing and teaches them how to assess the execution time of both sequential and parallel implementations.

A. Motivation

Parallel computing is becoming increasingly important in many fields, from scientific simulations to machine learning. As multi-core hardware platforms evolve and become widely available, people will have high-performance computing at their fingertips. To prepare students for careers in computing and particularly in parallel and distributed computing (PDC), CS educators are finding it increasingly challenging to decide what material to cover and in what order. The typical college-level introductory CS course sequence only exposes students to sequential type programming and data structures concepts, no matter the programming language used. PDC is mainly taught in upper-division courses such as parallel computing, algorithms, operating systems, and computer architecture. The community has long recognized [18], [22], [23] the benefits of exposing students to PDC concepts as early as possible during their studies.

Currently, Python is the most popular programming language according to the TIOBE Index [4] and one of the favorite programming languages to teach in the college introductory programming courses CS1 and CS2. Python's simpler syntax lets students start faster and provides an easier learning curve for those without programming experience. It also sets the stage for later coursework in object-oriented programming, parallel computing, data science, AI, and machine learning. Additionally, integrated development environments (IDEs) such as Jupyter Notebooks and VS Code have helped increase Python's popularity. At the same time, as all modern computing devices have multiple cores and many have GPUs, there is a push to integrate parallel and distributed computing in the early CS courses [20]. However, in most cases, CS1 and CS2 still teach students to solve problems using only sequential thinking. Even in the best case, concurrency and parallel problem-solving are only demonstrated using unplugged activities [13]. Many Python libraries for Just In Time (JIT) compilation, multiprocessing, and high-performance computing exist, however IPython Parallel [3] makes explicit parallel computations interactive, which means it is very appealing for beginners because it removes the barrier of accessing remote computers. IPython Parallel is a Python library that provides a simple and flexible way to execute code in parallel across multiple processors or machines. It is built on top of the IPython interactive computing framework and provides an easy-to-use, flexible interface for parallel computing tasks.

With IPython Parallel, students have all the power of IPython's inspection, interactivity, magic, and debugging at their fingertips, no matter of where they run their code. This makes it especially well-suited to prototyping and experimentation. IPython Parallel also presents standard APIs such as Python Executors that are compatible with many other implementations to make it easy to migrate to and from IPython Parallel, enabling developers to write code on a single multi-core laptop but to deploy it to a thousand cores on an HPC cluster.

For this assignment, we use the IPython Parallel package together with the mpi4py package. IPython Parallel is a lightweight framework that efficiently manages clusters of IPython processes. It can utilize all cores of a single machine to execute computations in parallel. Once you develop your parallel code locally, IPython Parallel lets you scale it up resiliently and elastically to clusters with many nodes for speeding up your solution or for solving even larger problems. Under the Anaconda platform, both IPython Parallel and mpi4py can be installed with a conda or pip install command, under a separate conda environment. This installation includes an extension for both the classic Jupyter Notebook and JupyterLab. IPython's parallel computing architecture has been purposefully built to seamlessly integrate with the Message Passing Interface (MPI). In order to incorporate MPI functionality with IPython, the mpi4py package must be also installed.

B. The Assignment

This assignment aims to use the IPython Parallel and mpi4py packages to introduce students in introductory programming classes to parallel computing concepts. The starting point of this assignment is the sequential pseudocode in Listing 1, which computes and displays the Mandelbrot set shown in Figure 1. The Mandelbrot set is considered one of the most popular fractals. The Python code generates an image of the Mandelbrot set by iterating the function $z = z^2 + c$ for each complex number in a grid spanning the specified range of the complex plane. The number of iterations is limited to a maximum value, and a threshold is used to determine when a point has diverged. The resulting image can be plotted using Matplotlib. Generating the image of the Mandelbrot set is a widely-used benchmark to test computer systems as well as programming language implementations because it is computationally intensive and a speed up can be easily observed.

The Mandelbrot set computation is relatively easy to parallelize because the value of each pixel in the image can be calculated independently, without any information about the values of surrounding pixels. The main idea of this embarrassingly parallel example is to group the pixels together and assign them to be processed by the same process or core.

To generate an image of the Mandelbrot set faster, we employ MPI to distribute the work among multiple processes. First, the image is split into rows, and each process works on a subset of rows. The results are then gathered on the root process (rank 0), which combines them into the full image. Finally, the root process plots the Mandelbrot set using Matplotlib or other plotting packages. Note that the MPI code is more complex than the non-parallel code, but it can significantly speed up the computation for larger images or when running many iterations.

Listing 1: Sequential Mandelbrot set Python code

```
MAX_ITER = 100

def mandelbrotSet(complex):
    z = 0
    i = 0
    while abs(z) <= 2 and i < MAX_ITER:
        z = z**2 + complex
        i += 1
    return i
```

The parallel MPI implementation is presented in a Jupyter notebook using IPython Parallel to make it simple and easier for students. A static-type MPI parallel implementation of the Mandelbrot set is also provided to the students. The main goal of the assignment is to check if this parallel implementation scales well when the number of processes or cores increases (e.g., 2, 4, 8, 16). Students are asked to run these experiments and to plot a bar plot to show the time it takes to finish the computation using different numbers of computing cores. Students will use the generated plot to decide if the output scales linearly with the number of cores. Another question highlights the relationship between the output and Amdahl's [5] and Gustafson's [15] laws. More details about this assignment can be found at https://github.com/alinutzal/EduPar-23_Peach.

C. Setup and Learning Outcomes

Before assigning this example as homework or as an in-class lab, the instructor should review fundamental concepts such as modern CPU architecture, Amdhal's law, Gustafson's law, task distribution, and speed-up. Also, the students should have access to a computer with multiple cores, have miniconda installed, and be able to

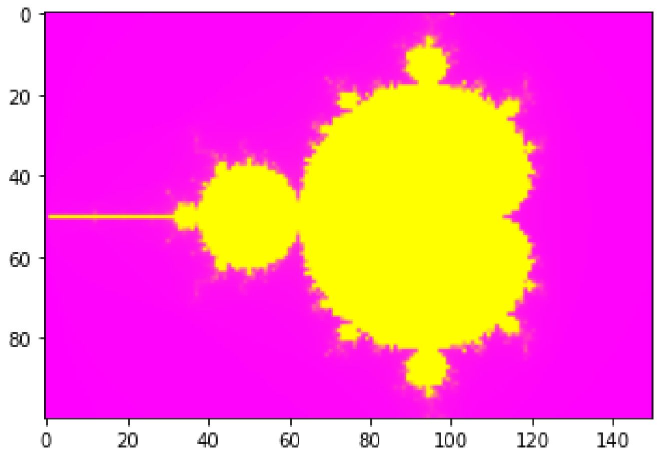

Figure 1: Mandelbrot set

install IPython parallel and mpi4py in a conda environment. Other possible resources are Google Colab, which provides environments with up to 2 cores, or access to a supercomputer. For example, Ohio Supercomputing Center [12] gives access to HPC resources for instructors and students in Ohio. Instructors can request Classroom Projects for individual courses and generate accounts for students.

In terms of learning outcomes, after completing this assignment students, should be able to:

- Understand the basics of parallel computing and how it can be used to speed up computationally intensive tasks.
- Explain the concept of embarrassingly parallel tasks and how they can be easily parallelized. Also, develop an understanding of how to split work into subtasks to be processed in parallel.
- Use IPython Parallel and mpi4py packages to implement parallel tasks in Python.
- Apply the Mandelbrot set iterative process as a benchmark to test sequential and parallel Python implementations.
- Explain the process of distributing work among multiple cores using MPI.
- Implement Python code to distribute work among multiple cores using MPI.
- Demonstrate how to gather results from multiple cores and combine them into a single final result.
- Understand the limitations and benefits of parallel computing in terms of performance, complexity, and running time.

These learning outcomes align well with the parallel computing learning outcomes highlighted in the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates [20] for CS1, CS2, Data Structures and Algorithms, and Parallel Computing courses.

D. Student Feedback

To understand the experience of CS1 and CS2 students in using IPython parallel with MPI, it is important to consider their background knowledge and familiarity with basic programming concepts. Students with more experience and motivation in programming may find the assignment straightforward and engaging, while those with less experience may need additional help, especially with the implementation details. Therefore, it is crucial to provide appropriate guidance and support to all students, regardless of their prior knowledge. This can include providing clear lecture notes, instructions, examples, and documentation, as well as offering office hours, tutoring, or peer support. Encouraging students to work collaboratively and share their experiences can also enhance the learning experience and build a sense of community.

Regarding the benefits of the assignment, students can gain valuable skills in parallel computing, which is becoming increasingly important in many fields, such as scientific computing, machine learning, and data analysis. Additionally, the Mandelbrot set computation provides a tangible example of how parallel computing can significantly improve performance and efficiency, which can motivate students to explore further related applications.

Overall, incorporating IPython Parallel and MPI into introductory programming courses can be a valuable addition to the curriculum and can inspire students to pursue more advanced topics in parallel computing and CS in general. In this context, most feedback received from students solving this assignment was very positive. The following are a few of the comments collected from students:

"I found this assignment creative and very useful. In addition, the instructor was helpful in clarifying the hard concepts."

"This assignment taught me a ton about parallel programming. Python is a little trick[y] at first, but I can try programming other algorithms now."

"Very challenging and fun. Enabled me to incorporate parallel programming skills into problem-solving."

E. Conclusions

The proposed assignment provides an approachable introduction to parallel and distributed computing for CS students. Before attempting this assignment, students are expected to have a working knowledge of Python and Numpy, usually gained earlier in the semester. This assignment requires access to a multicore computer and a miniconda environment. Students can use either Jupyter Notebooks or VS Code to run the notebooks. The skills acquired while solving this assignment will be useful in understanding how parallel code is designed. This assignment can be extended to present a dynamic task assignment parallel solution for the Mandelbrot set. The same approaches can be used to show the generation of other fractals, such as Julia, or extended to solve different problems requiring a high amount of computation, such as Monte Carlo simulations and random numbers generation.

III. Programming Contest Judge

Now we describe our second Peachy Parallel Assignment, the implementation of a judge for programming contests. Our idea is that using a real life problem will attract students' interest to the problem even if it is as complicated as those in parallel and distributed programming.

This assignment was used in "Parallel and Distributed Programming", a mandatory course in the curricula of the third-year undergraduate students pursuing the Computer Science specialization at "Babeş-Bolyai" University. The course classes contain lecture hours, but also laboratories where students must do assignments that allow them to put the principles, concepts, and mechanisms they are learning about into practice. The course uses Java and C++ to demonstrate the general programming concepts and mechanisms. For this project, we required that students do their implementation in Java based on previous years insights' into the students' preferences.

The assignment is a complex project that covers many of the previously presented topics:

- client-server,
- thread-pools,
- balanced distribution,
- producer-consumer pattern,
- conditional synchronization,
- data race,
- fine-grain vs. coarse-grain synchronization,
- synchronized methods vs. locks,
- signalization of the end of an operation/computation in multi-threaded and client-server environments,
- the impact on the performance of right splitting of the resources (number of threads).

Details about the assignment could be found at https://www.cs.ubbcluj.ro/~vniculescu/pdp/.

A. The problem

The high-level problem statement given to students is the following: *An international programming contest proposes several assignments to very many participants from different countries. For each assignment some points could be gained and finally the competitors that gain more points will be the winners. There are several ethical rules that should be respected; if a competitor doesn't respect one of these rules, then he/she is eliminated from the competition. At the end of the competition, the results for each assignment are sent (by each country client) to a central server that stores them in a specific directory. There are several prizes and in order to* *determine the winners, a list ordered descending based on the total points is needed. The server is responsible for delivering this list as soon as possible.*

Performance analysis should be conducted based on the resources' allocation for different tasks, on synchronization granularity, and on the size of the blocks sent by the clients.

They are also given the following information and constraints:

- There are the same number of participants in each country.
- Configuration data are read from a given file that contains:
  #_assignments, #_countries, #_competitors, #_prizes
- For each assignment and each country there is an associated file that contains the results. A result for a specific assignment is specified using a pair (id, *#_points*), where id represents the id of a competitor. The file that contains the results of a specific assignment contains all the corresponding pairs (an id appears only once), in an unspecified order. For each assignment the maximum number of points is 100.
- If a competitor breaks an ethical rule for an assignment, then a pair $(id, -1)$ is added in the corresponding assignment file.
- The list should be represented as an ordered linked list.
- For the list construction at the server side, a maximum of p threads can be used (the number p will be different for different test cases).
- The final result is represented by an output file that contains the country and the id of each winner in descending order of their accumulated points, each on a separate line.

B. Design

Students are also told to create the following two components:

1) Client-Server component
   - Each client is associated to a country and sends the files of the corresponding assignments results.
2) Classification list construction component
   - This is done at the server side using multiple execution threads.

The result data from each country for each assignment are sent to the central server through the Client-Server application. The results for each assignment are processed as soon as they are completely received. The server receives files and stores them into directories that correspond to each assignment, and concurrently executes the second component. A visual description of its behavior is depicted in Figure 2.

The following technical constraints [2], [8], [9], [16] should be followed:

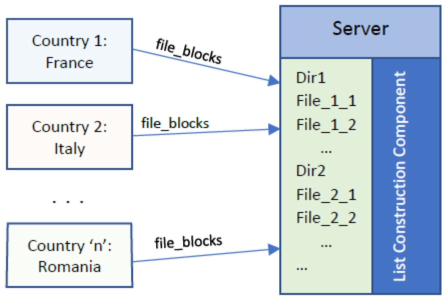

Figure 2: Visual description of the client-server application.

- The server uses a thread pool executor defined with t threads (`Executors.newFixedThreadPool(t)`).
- The connection should be based on Java sockets.
- A client sends a file split in blocks of predefined size.

The ordered list construction component is executed at the server side and uses the files that contains the results for each assignment sent by each country.

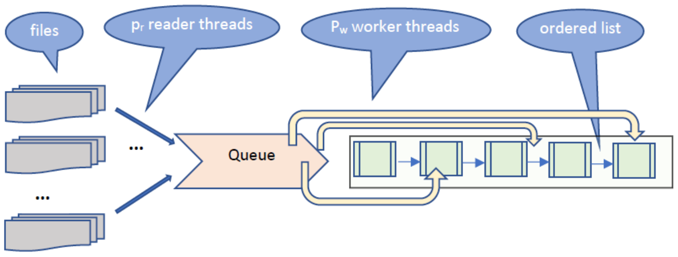

Figure 3: Visual description of the ordered list construction.

The p threads that can be used for the list creation are split into two groups: *readers* (p_r) and *workers*(p_w) such that $p = p_r + p_w$. In addition, the assignment specifies the following:

- The *readers* are responsible for reading pairs from the files and add them into a queue Q that has a maximum capacity.
- A *worker* takes a pair from Q and adds it into the list L (initially this list is empty and eventually it will contain the descending ordered list).
- The *producer-consumer* pattern [21] should be used! Readers are the producers and the workers are the consumers.

Figure 3 describes the steps and the entities involved in this activity.

For adding a pair into the list, the synchronization should be based on the following:

- Variant W_A: mutual exclusion by blocking the entire list,
- Variant W_B: mutual exclusion at the node level.

C. Methodology and Analysis

The project is very complex and it should be developed in stages. Because of this, two approaches could be considered:

1) *Waterfall approach.* Inform the students from the beginning about the entire goal of the application, and then split its development into stages.
2) *Agile approach.* Split the problem into independent stages. This means to starting first with the problem that creates the ordered list from a given set of input files, then, develop a client-server application that receives files from clients, and then, integrate these two components.

The first approach has the advantage of rising the students' interest by focusing on the real-life application, but it may increase the risk of failure due to the estimated complexity. The second is an agile approach that could lead the students to the final results following a gentler path, and in the same time, allowing better focus on different concurrency mechanisms. This was the approach that we followed.

Following the agile approach, we broke the problem into the following stages:

1) Development of the component that creates the ordered list from some input files that contain the pairs of type $(id, points)$.
   a) Development of the first variant that uses W_A: mutual exclusion by blocking the entire list,
   b) Development of the second variant that uses W_B: mutual exclusion at the node level.
   c) Comparison of the two approaches.
2) Develop a client-server application in which files are sent from clients to the server by splitting the files in blocks.
3) Integrate the first component into the client-server application.
4) Testing with different test-cases.

In order to make the first stage independent, the files that finally will be received from the clients are generated by using random numbers.

At the first stage, the *producer-consumer* pattern is discussed and analysed in more details. For the second stage, we discuss the implications of using *fine-grain synchronization* over *coarse-grain synchronization* [14].
The general *producer-consumer* pattern and its implementations were previously presented at the lectures, so the students just have to identify the fact that this pattern fits to the problem *readers=producers, workers=consumers*), and put it to work.

The workers should add new nodes into the ordered list, update the points of some nodes, or delete some nodes. These operations could have a *data race*, and so, the implementation uses mutual exclusion. This can be done for the entire list, which allows a very simple implementation. For large volumes of data, however, the performance can be much better if synchronization is done at the node level. Synchronization at the node level is not so simple to understand and implement and a detailed analysis of this solution should be provided to the students from the beginning (concrete examples, scenarios based analysis, recommendations as using head and tail empty nodes, etc.). The process illustrates the need to use *fine grain synchronization* even if it implies a much complex solution compared to *coarse grain synchronization* [1], [14].

The readers should read pairs from files. The files are organized into $\#\_assignment$ directories, and there are p_r threads that should fulfill this task. It could be assumed that each directory contains a similar number of pairs distributed into $\#\_countries$ files stored in each directory. A *balanced distribution* of the work should be done and this mainly depends on the relation between $\#\_assignment$ and p_r.

Another interesting problem, which is not so obvious at the first analysis, and proved to be difficult for the students, is the need to signal the end of the computation. For the workers, an empty queue does not necessarily mean that the work is finished; they should be informed when no other data is going to be added to the queue. It is important for the students to understand that this happens only when all the readers finish their job. The *Poison Pill* pattern [17] was discussed in this context. On the other hand, atomic variables could be used, and so, *low-level synchronization* was used, too.

The solution should be tested for correctness and also performance. The correctness testing is suggested to be done by implementing a sequential solution, and then comparing the outputs of the sequential and of the parallel solutions. The students are informed that this method just increases the level of confidence in correctness.
Different values for p, p_r, p_w are considered for the performance analysis, and the execution times are compared with the sequential time.

This first stage is oriented especially to synchronization problems. Regarding this, an important observation has been noticed: initially the students are not good at identifying all the data-race situations or the synchronizations imposed by the conditional waiting. After they realize the importance of sychronization, they tend to have the opposite behavior and to overuse synchronization and mutual exclusion to the detriment of performance.

The second stage imposes the development of a basic client-server application, in which the clients send several files to the server that stores them into different directories based on their names. For sending files the students are told to define clients that send chunks (binary blocks) of files in sequence. The problem of *signaling the end of the computation* also appears here, but in the context of the distributed memory environment, where the solution should be based on messages. The testing should evaluate the impact of the block size on the performance. The possible sizes of these blocks should be discussed, and performance analysis could be done for different values.

In the integration stage, the two main components are put together. This also involves some challenges due to the requirement of concurrency. The readers of the component that constructs the list should start reading files associated to an assignment as soon these are all received. When all the files that correspond to an assignment are received, the directory that contains them may be used by the readers. For implementing this, flags defined using atomic variables should be used.

Finally, in the last stage, the whole application is tested in several scenarios. The scenarios are defined based on different values for the total number of threads used by the server (t), how are these distributed for the different tasks (p_r, r_w), and the size of the blocks into which the clients split the files are split to send them. All these scenarios were tested for the both variants W_A and W_B. The testing activity is very important for this project since it depends on many parameters that may have impact on the performance.

Depending on the server hardware, the total number of threads should be split between the threads that work for thread pool that receives the files, the threads that read pairs from the files, and the threads that add the pairs to the list.

D. Variations

A significant possible variation of this assignment is to eliminate the reader threads and to have the clients send result pairs (not only one, but a bunch of pairs) which are put into the queue (Q) directly by the server's executor tasks. The advantage of this approach is that the threads at the server side could be used more efficiently. If both variants are implemented than they could be compared from the performance point of view.

Another variation is to implement the list construction using only CAS operations [14].

This problem could also have different real-life wrappers; in general, it could be applied to problems where many applications are submitted from different locations, and they have to be aggregated based on specified criteria in order to select the best of them.

Another variation, not as much a real-life problem but appropriate for students interested in Mathematics, is to consider the computation of the sum of several polynomials that are represented using monoms— pairs of (coefficient, exponent). We have used this for the students from the specialization "Mathematics and Computer Science".

E. Conclusion

Overall, the problem emphasizes that there are many factors on which the performance depends, and these factors are correlated. The analysis of the possible optimization should be done first at the component level and then for the entire application.

The problem forces students to use many of the mechanisms and paradigms they learned related to parallel and distributed programming, and shows that deep understanding and analysis are important in order to obtain good performance.

Even though the problem is complex, the students managed to complete the assignment. We believe that the most important reason for this was the agile approach that split it into stages that could be developed and tested independently. Agile methodology has important advantages in software development and it can be successfully used in teaching and learning activities [19]. The students' feedback emphasized that this assignment allows them to better understand the patterns, mechanisms, and principles of parallel and distributed computing that were previously discussed, and gave them a rewarding practical experience.

References

[1] "The Java tutorials, Oracle Java documentation — Concurrency," https://docs.oracle.com/javase/tutorial/essential/concurrency/.

[2] "The Java tutorials, Oracle Java documentation — Networking," https://docs.oracle.com/javase/tutorial/networking/.

[3] "Using IPython for parallel computing — ipyparallel 8.4.1 documentation," https://ipyparallel.readthedocs.io/en/latest/, accessed: 2023-1-27.

[4] "TIOBE index," https://www.tiobe.com/tiobe-index/, Dec. 2021, accessed: 2023-1-27.

[5] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the afips conference proceedings, vol. 30 (atlantic city, nj, apr. 18–20), afips press, reston, va., 1967, pp. 483–485, when dr. amdahl was at international business machines corporation, sunnyvale, california," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 3, pp. 19–20, 2007.

[6] H. Bücker, H. Casanova, R. F. da Silva, A. Lasserre, D. Luyen, R. Namyst, J. Schoder, and P.-A. Wacrenier, "Peachy parallel assignments (EduPar 2022)," in *Proc. 12th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*, 2022.

[7] D. Bunde, "Modules for introducing threads," in *Topics in parallel and distributed computing: Introducing concurrency in undergraduate courses*, S. Prasad, A. Gupta, A. Rosenberg, A. Sussman, and C. Weems, Eds. Morgan Kaufmann, 2015, ch. 4, pp. 59–82.

[8] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*.

[9] K. L. Calvert and M. J. Donahoo, *TCP/IP Sockets in Java, Practical Guide for Programmers*, 2nd ed. Morgan Kaufmann, 2008.

[10] R. Carratalá-Sáez, A. Gonzalez-Escribano, A.-S. Iliopoulos, C. Leiserson, C. Park, I. Rosa, T. Schardl, Y. Torres, and D. Bunde, "Peachy parallel assignments (eduhpc 2022)," in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2022.

[11] H. Casanova, R. F. da Silva, A. Gonzalez-Escribano, H. Li, Y. Torres, and D. Bunde, "Peachy parallel assignments (EduHPC 2021)," in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2021.

[12] O. S. Center, "Ohio supercomputer center," 1987. [Online]. Available: http://osc.edu/ark:/19495/f5s1ph73

[13] S. K. Ghafoor, D. W. Brown, M. Rogers, and T. Hines, "Unplugged activities to introduce parallel computing in introductory programming classes: an experience report," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science*

Education, ser. ITiCSE '19. New York, NY, USA: Association for Computing Machinery, Jul. 2019, p. 309.

[14] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*. Addison Wesley, 2003.

[15] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures," *IEEE Parallel & Distributed Technology: Systems & Applications*, vol. 1, no. 3, pp. 12–21, 1993.

[16] E. R. Harold, *Java Network Programming*, 4th ed. O'Reilly Media, Inc, 2013.

[17] B. L. Massingill, T. G. Mattson, and B. A. Sanders, *A Pattern Language for Parallel Programming*, ser. Software Patterns Series. Addison Wesley, 2004.

[18] T. Newhall, K. C. Webb, V. Chaganti, and A. Danner, "Introducing parallel computing in a second CS course," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. ieeexplore.ieee.org, May 2022, pp. 321–329.

[19] V. Niculescu, D. M. Suciu, and D. V. Bufnea, "Agile principles applied in learning contexts," in *Proc. 3rd Intern. Workshop on Education through Advanced Software Engineering and Artificial Intelligence*, 2021, pp. 31–38.

[20] S. K. Prasad, A. Chtchelkanova, S. Das, F. Dehne, M. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc, M. Lumsdaine, D. Padua, M. Parashar, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: core topics for undergraduates," in *Proceedings of the 42nd ACM technical symposium on Computer science education*, ser. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, Mar. 2011, pp. 617–618.

[21] P. Raj, A. Raman, and H. Subramanian, *Architectural Patterns*. Packt Publishing — O'Reilly, 2017.

[22] X. Suo, O. Glebova, D. Liu, A. Lazar, and others, "A survey of teaching PDC content in undergraduate curriculum," *2021 IEEE 11th Annual*, 2021.

[23] N. Watkinson, A. Shivam, A. Nicolau, and A. Veidenbaum, "Teaching parallel computing and dependence analysis with python," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. ieeexplore.ieee.org, May 2019, pp. 320–325.