Choosing a library for the Python programming language for visualizing the operation of parallel algorithms

Sergii Sharov[1*], *Yurii* Sitsylitsyn[2], *Oleksii* Naumuk[2], *Dmytro* Lubko[1], and *Vira* Kolmakova[3]

[1]Dmytro Motornyi Tavria State Agrotechnological University, 66 Zhukovsky Str., Zaporizhzhia, 69600, Ukraine
[2]Bogdan Khmelnitsky Melitopol State Pedagogical University, 59 Naukovoho mistechka Str., Zaporizhzhia, 69000, Ukraine
[3]Pavlo Tychyna Uman State Pedagogical University, 2 Sadova Str., Uman, 20300, Ukraine

Abstract. The research compares the capabilities of several libraries for the Python language, which allow creating a test application and visually demonstrate the operation of a parallel program in real time. It was found that the Python language is often used to develop parallel programs with internal and external libraries. To provide multithreading and parallelism, applications created in Python use external libraries, including mpi4py.futures, PETSc for Python, MPI for Python, d2o, Playdoh, PyOMP, and others. Visualization and animation of the operation of parallel programs will help to understand the principles of parallel computing. We compared test applications created with the use of Matplotlib, Seaborn, Plotly, Bokeh, Pygame, PyOpenGL libraries. According to the results of the observation, it was found that the Seaborn library is the best choice for developing a test application for animating the operation of a parallel program. Keywords: Python, parallel computing, programming, libraries.

1 Introduction

In the conditions of rapid technogenic development and growing requirements for data processing in real time, the speed of processing large volumes of information becomes a critically important factor in the operation of parallel programs. The most common programming languages used to develop parallel programs are Fortran, C, C++, Java and Python. Each of them has unique capabilities and syntactic features, as well as approaches and tools for implementing parallel algorithms. The multi-paradigm and powerful Python programming language is a powerful tool for many tasks. It is used for website creation, machine learning [1], [2], big data analysis [3], scripting, etc. A comparison of Python with other programming languages was made in the work by [4]. The research by [5] compared the performance of parallelism in Python and C++.

* Corresponding author: segsharov@gmail.com

At the same time, novice programmers often face difficulties with understanding parallel paradigms. Parallel programming, where the execution of multiple processes simultaneously creates a new level of complexity and interaction between processors, is quite different from sequential programming with its linear and one-dimensional processes. This difference often causes misunderstandings and mistakes in approaches to program development and analysis, especially in the context of multiprocessor and distributed systems. As a result, a parallel computing course, along with the graphic design software [6], is one of the important components of the training of future software engineers. It is crucial to understand the key concepts of parallel computing, such as synchronisation, resource management, and scaling, when preparing students for effective work in a modern software environment. In this context, methodological approaches of teaching parallel computing are reflected in the works by [7], [8].

For the development of parallel programs, the Python language has several libraries, the capabilities, and performance of which are highlighted in research of various scientists. The paper [9] analyzes the software components of MPI for Python and PETSc for Python. The capabilities of mpi4py.futures, which works on the basis of the Message Passing Interface (MPI), are shown in the work by [1]. The effectiveness of the Playdoh library is reported in the work by [10]. A PyOMP module that supports OpenMP is described by [11]. In the work by [12], individual libraries (JMetalPy, Parsl, Ray, PyWren, PyNetLogo) were compared for the implementation of parallel and multiprocessor processing according to various criteria.

To facilitate the study of the parallel programming principles, one can use educational applications that demonstrate the process of the parallel program operation in a visual (static or dynamic) form. They give future programmers an opportunity not only to read about the operation mechanisms of parallel algorithms, but also directly observe their execution in real time. In addition, a visual approach allows detecting and analyzing errors in parallel programs, which is an important skill in modern programming.

In addition to the standard Tkinter library, the Python language has numerous libraries and frameworks designed for 2D and 3D data visualization, multidimensional array processing [9], interactive work with the Integrated Development Environment (IDE) in the application development process. Given the importance of studying the capabilities of the Python language for performing parallel computing based on visual representation of data, the goal of the research is to select a library for the Python language that allows creating an educational application and visually demonstrate the operation of a parallel program in real time.

2 Theoretical Background

Today, the general-purpose programming language Python holds high positions in international rankings, such as TIOBE (1st place) and Stack Overflow (3rd place in the Technology section). Its popularity is based on several advantages. Firstly, the Python language is characterized by a simple syntax that allows even beginners to learn how to create applications quickly. Secondly, it supports several programming paradigms, including functional, imperative, object-oriented. Support of the object-oriented programming concept [13] allows structuring a program in the form of modules and classes, simplifying its extension and maintenance. Third, Python is a cross-platform programming language. This means that a developed application will run on different operating systems, such as Windows or Linux, without a need to make significant changes to the code or adapt it to a specific platform.

The work by [14] gives the following reasons for the popularity of the Python language, these are: simplicity and ease of mastering the language syntax; versatility, which allows

using Python in different IDEs; a substantial amount of educational materials available in the public domain; a wide community of Python programmers; one of the main areas of Python usage is web development, data analytics, and machine learning, which are currently considered as the most popular in the IT industry. As a result, Python is taught in higher education institutions, in particular, within Computer Science specialties [7], as well as independently with the help of various online resources, such as Massive Open Online Course (MOOC) [15].

It is worth mentioning that performance of modern applications requires the implementation of parallelism and multithreading. The parallelism paradigm involves dividing a large task into several parts that will be executed simultaneously on different processors [12]. In the Python language, parallelism is ensured by the use of the multiprocessing module. However, it cannot provide high performance due to several reasons. First, the Python language uses a program code interpreter, which does not allow you to get the advantages of optimization when using a compiler. Second, using the Global Interpreter Lock (GIL) prevents multiple threads from executing simultaneously, which negatively affects performance [11]. Based on the parallel and sequential implementation of several algorithms, F. J. M. Arboleda et al. [5] found that the multiprocessing module had worse performance than the OpenMP directives used in the C++ programming language. Therefore, to create Python applications that require parallel computing, one can use external compiled modules, i.e. libraries [16]. Let us consider some of them.

Module mpi4py.futures is considered to be part of the mpi4py standard library, so it has a shared memory limit. However, due to the use of MPI for communication between processors, the module can be scaled to several processors. The Mpi4py.futures interface is similar to the concurrent.futures package [1].

MPI for Python is a general-purpose package that allows parallel programs written in Python to use multiple processes simultaneously. The package can be used in any environment that supports MPI [9].

PETSc for Python is a portable toolkit based on the object-oriented programming paradigm that uses modern data structures to solve large tasks [9].

d2o is an open-source Python module designed to process multidimensional numeric arrays without sacrificing performance. The module interface is similar to the numpy.ndarray interface. Since part of the module is written in Python, the advantages of d2o are compactness, ease of use and modifiability [16].

Playdoh is a Python library designed to distribute computations between a small number of multi-core computers. It is based on NumPy6, has a clear interface, and it can be ported to most platforms [10].

PyOMP is a module based on using OpenMP in Python. It contains the most prominent OpenMP Common Core elements (each thread executes the same code, loop-level parallelism, divide and conquer with tasks) [11].

If you take a more thorough look at the mechanism of a parallel program, you will see that it involves the creation and simultaneous execution of several parallel threads, each of which moves towards a common endpoint where it completes its activities. These threads operate at different speeds, which depend on the data they process and the specifics of the algorithms used to process the data. While executing, threads can pause their work depending on algorithmic conditions. Such suspension points are critical for understanding the behaviour of parallel programs and can affect the overall efficiency and performance of program execution. In addition, it is worth paying attention to flow control mechanisms, such as blocking, conditional variables, and barriers. They regulate interactions between threads, ensuring the correct order of operations and preventing problems such as race conditions or mutual blocking.

An important aspect of parallel programs is load balancing between threads. Different threads can perform different tasks or process different sections of data. Their effective distribution is the key to optimizing performance. With improper load distribution, some threads may remain underutilized while others are overloaded, resulting in a decrease in overall system performance. This aspect is especially important in the case of working with large data and in complex computational tasks. Another important aspect is error processing in parallel programs. Parallel systems are often more prone to errors due to the complexity of interactions between threads. Detecting and processing errors such as data loss, data inaccuracy, or synchronization errors are critical to ensuring the reliability and stability of parallel programs.

Understanding these aspects of the operation of parallel programs is key to creating efficient and reliable parallel systems. Therefore, demonstrating these aspects in training/test applications will not only help novice programmers to understand how they arise, but also to learn how to solve them. In general, the development of an application for visualizing the operation of parallel programs provides a unique opportunity not only to study the theory of parallel programming, but also to visually observe the practical application of the obtained information.

3 Research Methods

The purpose of the research was to choose a Python library that could be used to develop a training application and visually demonstrate the operation of a parallel program in real time. This application is not intended as a tool for developing full-fledged parallel programs. Its main purpose is to serve as a learning tool that facilitates the understanding of parallel programming concepts through visualization and animation of the operation of parallel programs. To achieve this goal, we compared several open-source and free libraries that have visualization and animation capabilities:

Matplotlib. It is one of the most popular data visualization libraries in Python. It allows creating static, animated and interactive visualizations, which makes it useful for demonstrating dynamic processes in parallel programs.

Seaborn. This library is based on the Matplotlib library, with additional features for creating more complex and aesthetically appealing visualizations. It can be useful for highlighting key aspects of data in parallel programs.

Plotly. This library supports the creation of high-quality interactive graphs. Plotly is especially useful for creating interactive visualizations that allow users to better interact with the presented data.

Bokeh. It is a library for creating interactive visualizations mainly in web browsers. Bokeh works well for creating detailed interactive charts and graphs.

Pygame. Although Pygame is typically used for game development, this library can also be useful for creating animations and graphical user interfaces. It allows developing complex animation scenes that can display processes in parallel programs.

PyOpenGL. It is a Python-interface for OpenGL that can be used to create visualizations using 3D graphics. This can be particularly useful for demonstrating complex parallel processes in three-dimensional space.

The comparison of the capabilities of the libraries was carried out on the practical test, which involved the parallel calculation of matrix multiplication by a number. Mathematically, this can be described using formula (1):

$$b_{i,j} = k \cdot a_{i,j} \quad (1)$$

Where $b_{i,j}$ are the elements of the resulting matrix, $a_{i,j}$ are the elements of the original matrix, and k is the multiplier

To perform the calculation, we developed small parallel programs using selected libraries. Each of the programs uses four threads to multiply parts of the matrix by a number. Since each thread will process its own part of the matrix, this allows visualizing this process in real time. Animation of the parallel calculation process was implemented in the test programs. When evaluating libraries for developing the parallel program animation application, priority was given to those that have good integration with Tkinter (the standard graphical user interface (GUI) library for Python) [17].

4 Results and Discussion

The operation of the test example of the parallel program developed with the Matplotlib library is shown in Figure 1.

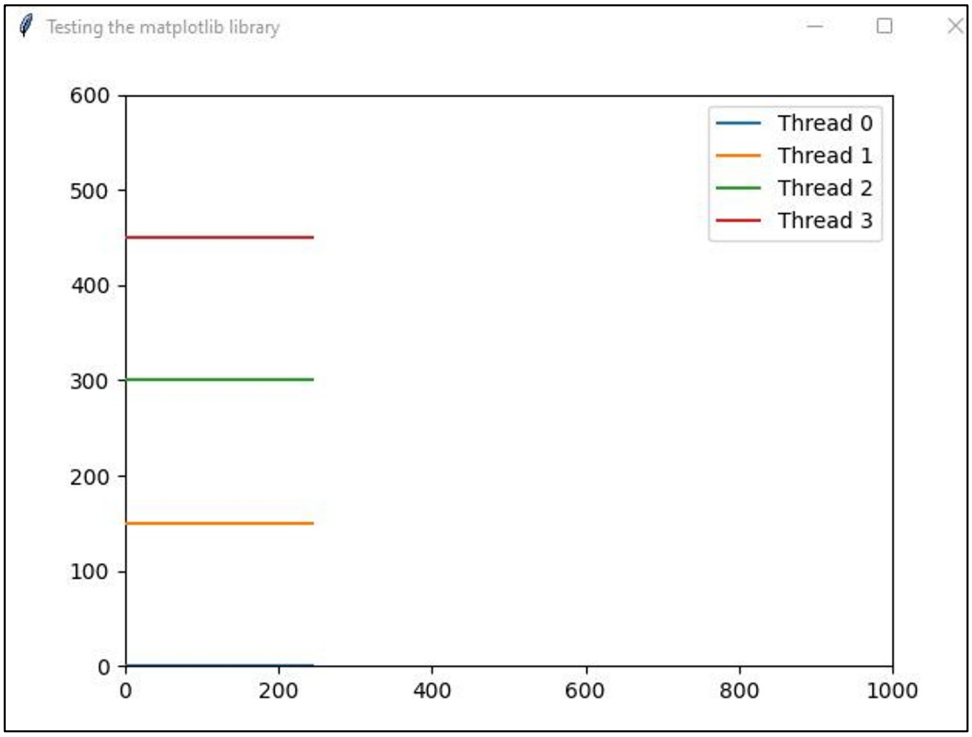

Fig. 1. The example of the program developed with the Matplotlib library

The use of the Matplotlib library with Tkinter to develop the test program for parallel calculation of matrix multiplication by a number revealed several key aspects. Matplotlib was easily integrated into Tkinter via the FigureCanvasTkAgg widget. This allowed integrating Matplotlib graphs directly into the Tkinter graphical interface. Thanks to this integration, it was possible to create a user interface with buttons, menus, and other control elements that interact with the visualization. Conclusion: using Matplotlib with Tkinter enables the development of an application that not only demonstrates the process of parallel computing, but also provides the flexibility and interactivity necessary for effective learning.

The operation of the test example of the parallel program developed with the Seaborn library is shown in Figure 2.

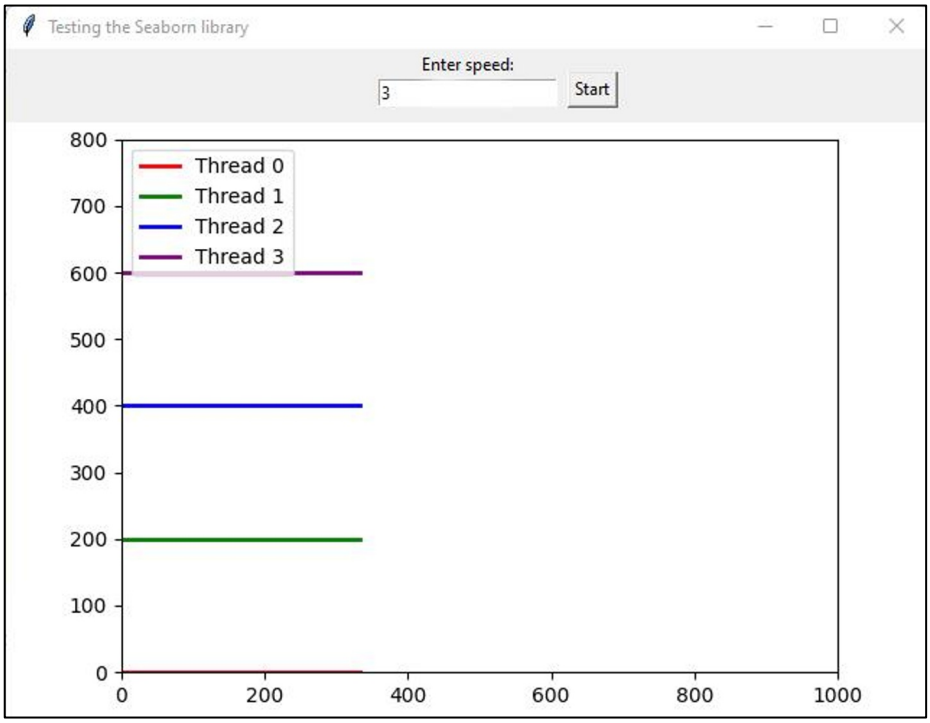

Fig. 2. The example of the program developed with the Seaborn library

Using the Seaborn library with Tkinter for developing the test case revealed more features than Matplotlib. Especially when it comes to creating more attractive and intuitive visualizations. This is achieved through better management of styles and colour palettes in the Tkinter graphical interface. Seaborn can be used to create more complex visualizations, which can include heat maps or other types of graphs. Unlike Matplotlib, Seaborn is not interactive. However, integrating with Tkinter allows the addition of interactive controls, such as buttons and sliders, for managing visuals. Seaborn simplifies the process of creating complex graphs, providing a higher level of abstraction, which allows us to develop our application faster with richer functionality. Conclusion: Using Seaborn together with Tkinter can give visualizations more aesthetic appeal and clarity. Thus, you can make information more comprehensible for programmers who are just starting to get acquainted with parallel computing.

The work of the test example of the parallel program developed with the Plotly library is shown in Figure 3.

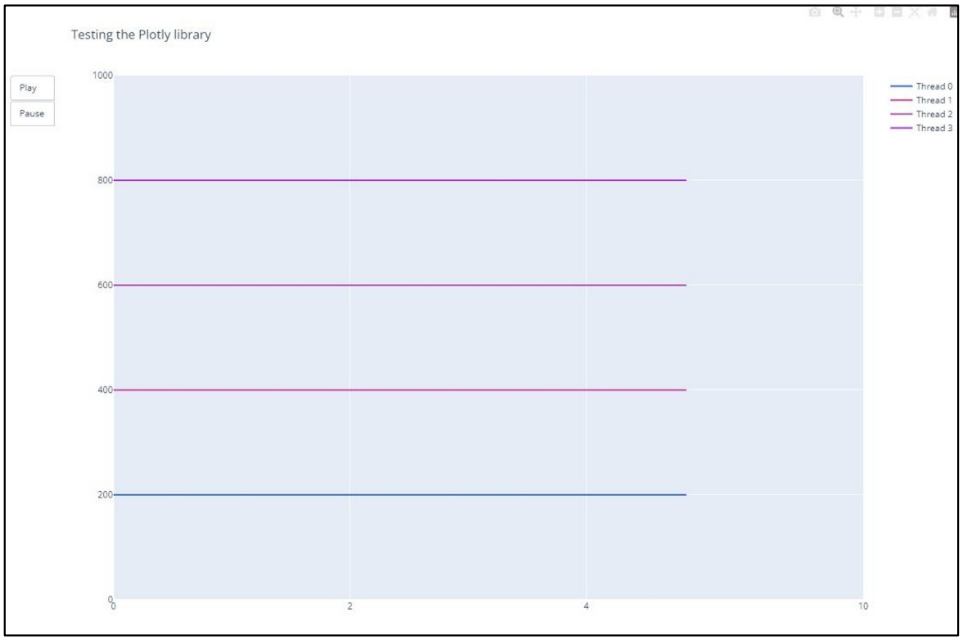

Fig. 3. The example of the program developed with the Plotly library

Using the Plotly library with Tkinter to develop a parallel application requires a special approach. This can be explained by the fact that the Plotly library was primarily developed for creating interactive visualizations in a web environment. One of the advantages of Plotly is its interactivity, which allows users to interact with visualizations (e.g. zoom in, view specific data, etc.). However, using Plotly in a desktop application requires more resources, especially when integrated with web technologies. It is also necessary to consider dependencies and compatibility with different versions of browsers or web widgets. Conclusions: Using Plotly with Tkinter may require additional efforts to integrate and optimize the performance of a program.

The Bokeh library was also developed for creating interactive visualizations in a web environment, so it has similar disadvantages for developing a test application. Similar to Plotly, Bokeh is not designed for direct integration with Tkinter. Embedding Bokeh visualizations in the Tkinter interface may require the use of web widgets or embedded browsers. It can significantly complicate development, since additional dependencies and integration challenges need to be considered.

Using the Pygame library with Tkinter for developing an application also requires additional efforts. It is due to the fact that Pygame is more focused on developing games and multimedia applications. Integrating Pygame with Tkinter can be a bit tricky, as both libraries have their own event loops and ways of managing windows. Special code may be required to coordinate these libraries to work together. Pygame is well optimized for creating graphical animations and can efficiently use system resources. However, it should be noted that working with Pygame may require more attention to detail in graphic design and animations. Conclusion: Using Pygame to develop an application can provide many opportunities for creative and efficient display of parallel computing, but it requires careful planning and understanding of the specifics of working with both libraries (Pygame and Tkinter).

PyOpenGL turned out to be a less suitable choice for the development of our test application, especially given the need to work with Tkinter and the implementation of interaction with a user. PyOpenGL is an interface for OpenGL that provides advanced

capabilities for working with 3D graphics. Its integration with Tkinter is not trivial and may require special widgets or additional settings to work properly. Conclusion: PyOpenGL provides powerful capabilities for working with 3D graphics, but is not a suitable option for developing our application.

Based on the analysis of various libraries, we came to the conclusion that Seaborn is the best option for developing the application for the animation of the parallel program. It is due to the following reasons:

- The Seaborn library allows creating attractive, informative graphics with advanced settings for styles and colour schemes, making visualizations more perceivable for users.
- Seaborn offers a high level of abstraction and simplifies the process of creating complex visualizations. It makes it more accessible to developers who do not have profound knowledge of data visualization.
- Seaborn supports a wide range of graph types and visualizations. It makes it possible to efficiently display various aspects of data in parallel computing, including resource allocation and execution efficiency.
- Despite the fact that Seaborn itself does not have any specific support for Tkinter, it can be used together with Matplotlib to integrate with Tkinter, which provides flexibility in creating a user interface.

In our opinion, the development of test applications with the ability to animate the operation of the program will allow beginners in programming to better understand how parallel processes interact with each other, how resources are managed, how typical problems such as “race conditions” or “mutual blocking” are solved. They will be able to visually evaluate the efficiency and potential problems in parallel programming structures, make adjustments to the code, and instantly see the results of these changes in the form of animations. In addition, during the creation of parallel applications, programmers will be aware of the capabilities of various libraries, learn to use them, and also get practical experience in programming [16], which is one of the main components of training a professional programmer.

5 Conclusions

Thus, parallel computing is one of the promising areas of technological development. The development of parallel applications can be implemented with the Python programming language, which allows the use of internal and external libraries. To provide multithreading and parallelism, applications created in Python use various libraries, including mpi4py.futures, PETSc for Python, MPI for Python, d2o, Playdoh, PyOMP, and others.

To simplify the understanding of the principles of parallel computing during the development of applications with the Python language, one can visualize their operation using the appropriate libraries. Such educational applications will allow one to see the process of resource management and the result of calculations in real time in the form of animation or graphics. For this purpose, the capabilities of such libraries as Matplotlib, Seaborn, Plotly, Bokeh, Pygame, PyOpenGL were analyzed. After analyzing the possibilities of the considered libraries, we found out that Seaborn is the best choice for developing the test application for animation of the work of a parallel program.

After choosing the Python library for developing parallel program animations, we plan to develop a simplified formal language for writing parallel programs that will be closely related to visualization. This approach to creating applications will allow programmers to see a direct connection between the program code and its execution in the form of animation.

References

1. M. Rogowski, S. Aseeri, D. Keyes, L. Dalcin, *Mpi4py.futures: MPI-Based Asynchronous Task Execution for Python,* IEEE Transactions on Parallel and Distributed Systems **34(2)**, 611-622 (2023) doi: 10.1109/TPDS.2022.3225481
2. N. Nagy et al., Phishing URLs *Detection Using Sequential and Parallel ML Techniques: Comparative Analysis,* Sensors **23(7)**, 3467 (2023) doi: 10.3390/s23073467
3. S. Chakraborty, A. Cortesi, N. Chaki, *A uniform representation of multi-variant data in intensive-query databases,* Innovations Syst. Softw. Eng. **12**, 163-176 (2016) doi: 10.1007/s11334-016-0275-9
4. Md. G. Rashed, R. Ahsan, *Python in Computational Science: Applications and Possibilities,* International Journal of Computer Applications **46(20)**, 26-30 (2018) doi: 10.5120/7058-9799
5. F. J. M. Arboleda, M. R. Arias, J. A. H. Riveros, *Performance of Parallelism in Python and C++, IAENG* International Journal of Computer Science **50(2)**, 1-13 (2023)
6. H. Chemerys et al., *Fundamentals of UX/UI design in professional preparation of the future bachelor of computer science,* AIP Conference Proceedings **2453(1)**, 030025 (2022) doi: 10.1063/5.0094433
7. N. Watkinson, A. Shivam, A. Nicolau, A. Veidenbaum, *Teaching parallel computing and dependence analysis with python,* In Proceedings 2019 IEEE 33rd International Parallel and Distributed Processing Symposium Workshops (IPDPSW 2019), 320-325 (2019) doi: 10.1109/IPDPSW.2019.00061
8. Y. O. Sitsylitsyn, V. V. Osadchyi, V. S. Kruglyk, O. H. Kuzminska, *Modeling training content for software engineers in parallel computing,* Journal of Physics: Conference Series **2611**, 012017 (2023) doi: 10.1088/1742-6596/2611/1/012017
9. L. D. Dalcin, R. R. Paz, P. A. Kler, A. Cosimo, *Parallel distributed computing using Python,* Advances in Water Resources, Advances in Water Resources **34(9)**, 1124-1139 (2011) doi: 10.1016/j.advwatres.2011.04.013
10. C. Rossant, B. Fontaine, D. F. M. Goodman, *Playdoh: a lightweight Python library for distributed computing and optimisation,* Journal of Computational Science **4(5)**, 352-359 (2013) doi: 10.1016/j.jocs.2011.06.002
11. T. G. Mattson et al., *PyOMP: Multithreaded Parallel Programming in Python*, Computing in Science and Engineering **23(6)**, 77-80 (2021) doi: 10.1109/MCSE.2021.3128806
12. A. Aziz et al., *Python Parallel Processing and Multiprocessing: A Rivew,* Academic Journal of Nawroz University **10(3)**, 345-354 (2021) doi: 10.25007/ajnu.v10n3a1145
13. S. Choporov, S. Gomenyuk, O. Kudin, A. Lisnyak, *Design patterns for object-oriented scientific software,* CEUR Workshop Proceedings **2105**, 441–444 (2018)
14. A. Saabith, M. Fareez, T. Vinothraj, *Python current trend applications-an overview,* International Journal of Advance Engineering and Research Development **6(10)**, 6–12 (2019)
15. S. Sharov et al., *Using MOOC to Learn the Python Programming Language,* International Journal of Emerging Technologies in Learning **18(2)**, 17–32 (2023) doi: 10.3991/ijet.v18i02.36431

16. T. Steininger, M. Greiner, F. Beaujean, T. Enßlin, *D2o: a distributed data object for parallel high-performance computing in Python,* Journal of Big Data **3(17)**, 1-34 (2016) doi: 10.1186/s40537-016-0052-5
17. S. Kurzadkar et al., *Hotel Management System Using Python Tkinter GUI,* International Journal of Computer Science and Mobile Computing **11(1)**, 204–208 (2022) doi: 10.47760/ijcsmc.2022.v11i01.027