Academic Journal of Nawroz University (AJNU), Vol.10, No.3, 2021
This is an open access article distributed under the Creative Commons Attribution License
Copyright ©2017. e-ISSN: 2520-789X
https://doi.org/10.25007/ajnu.v10n3a1145

Python Parallel Processing and Multiprocessing: A Review

Zena A. Aziz[1], Diler Naseradeen Abdulqader[2], Amira B. Sallow[3], Herman Khalid Omer[4]
[1] Technical College of Information Akre, Duhok Polytechnic University, Kurdistan Region-Iraq
[2,3,4] Department of Computer and Communication, Nawroz University, Kurdistan Region-Iraq

ABSTRACT

Parallel and multiprocessing algorithms break down significant numerical problems into smaller subtasks, reducing the total computing time on multiprocessor and multicore computers. Parallel programming is well supported in proven programming languages such as C and Python, which are well suited to "heavy-duty" computational tasks. Historically, Python has been regarded as a strong supporter of parallel programming due to the global interpreter lock (GIL). However, times have changed. Parallel programming in Python is supported by the creation of a diverse set of libraries and packages. This review focused on Python libraries that support parallel processing and multiprocessing, intending to accelerate computation in various fields, including multimedia, attack detection, supercomputers, and genetic algorithms. Furthermore, we discussed some Python libraries that can be used for this purpose.

KEYWORDS: Parallel processing, multiprocessing, Python, CPU, Multicore CPU, GPU.

1. Introduction

In recent years, the Python programming language has gained momentum for scientific computing. Often conventional tools like MatLab are replaced. [1]. It is open to all at no cost because Python is open source, and its portability makes its usability possible on many platforms. The language itself is lightweight, abridged, and highly suitable for quick prototyping, although it is strong enough to write significant applications. Some people don't give it enough credit for its usability and flexibility.

Python can very well be integrated with the C/C++ so that external performance or code-based modules can be easily invoked. Besides, it offers a wide range of scientific libraries, e.g. Processing and analyzing data, plotting and graphical user interfaces [2][3][4]. All these feature makes Python attractive to the scientific public but it has to be parallel to the languages used in large projects. CPython is the default implementation and most commonly used. [5], due to the global look in the interpreter, several threads cannot be run at once. A variety of options have been developed to create many Python processes, Shared and elastic infrastructure environments, including networks, clusters, and clouds.

Parallel computing, on the other hand, is a computational paradigm where several instructions are performed concurrently. It is based on the premise that significant problems can often be broken into separate ones and solved simultaneously (parallel). Bit-level parallelism, instruction-level parallelism, data parallelism, and task parallelism are the four types of parallel computation [6].

Hiotas been used for many years, especially in high-performance computing; however, interest in this field has recently increased due to physical hardware constraints on CPU frequency, such as shared-memory and distributed services, as well as infrastructure networks, clusters, and clouds [7][8]. Furthermore, the use of such resources and the generation of heat by computers has become a focus of recent technological advancement. As a result, parallel computing has established itself as a key concept in computer architecture, specifically in multi-core processors.

The Python multiprocessing module [9] allows processes to be spawned in SMP machines with an API like the module for threading, explicit calls for process generation, declaration passing, and implementation, cooperation, and result selection. The GIL problem is avoided by the multiprocessing module, which launches sub processes rather than threads through a fork system call. Parallel Python (PP) is a Python module that implements frameworks for parallel Python code execution on SMP and clusters. It is based on an API that includes explicit functions for specifying the number of workers to be used, submitting jobs for execution, obtaining worker results, Etc. Similarly to the multiprocessing module, the programmer is in charge of parallelism management, which combines the actual algorithm parallelism management. [10].

The paper is organized as follows. Sections 2, 3, 4, and 5 include context theory; Section 6 addresses related work. Section 7 contains a discussion of the analysis. Section 8 concludes with observations and future work.

2. Parallel Processing

Most modern PCs, workstations, and even handheld devices contain several central processing unit (CPU) cores. These cores are self-contained and can execute various instructions at the same time. Programs that use parallelization to take advantage of multiple cores run faster and allow better use of CPU resources. Parallel Processing is another term for speeding up the efficiency of running a program by dividing it into smaller pieces that can be performed simultaneously on multiple processors. [11], In general, each component has its processor. A program running on Q processors can complete Q times faster than a program running on one processor. [12].

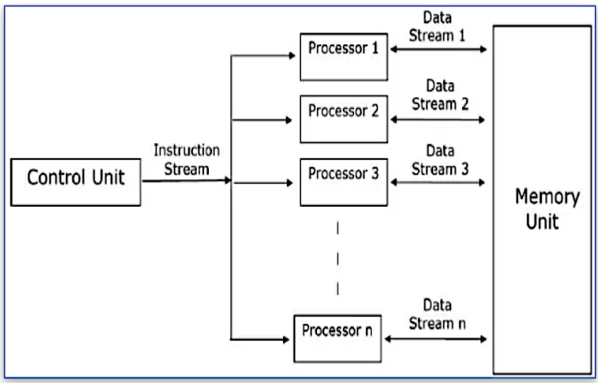

Figure 1: Structure of Parallel Processing [13]

2.1 Parallel Processing benefits

Just one program could be run on the original computers at a given time. The intensive operational program for one hour would take two hours to complete, and a tape-collection program that lasted for one hour would take two hours. In parallel, all programs are run simultaneously at the beginning of the parallel processing. The machine begins an input and output instructions first, and while waiting to complete the mission, the intensive operations program would be executed. It will take less than one hour to complete the two tasks. [14] .

2.2 Applications for Parallel Processing

- Parallel processing systems are used to ensure the security and dependability of the United States' remaining nuclear weapons arsenal. In the absence of nuclear testing, either above or below ground, very fine-grained numerical simulation is needed to evaluate and forecast potential problems caused by long-term storage of nuclear products. [15].
- Parallel processing is used to create computer-generated vehicles and railings to monitor the strength and endurance of the railings in the event of a collision. Executing one model on a single processing system will take up to five days, while it only takes a few hours on a parallel machine.
- Airlines use parallel processing to analyze customer data, estimate requests, and determine the fees to charge.

- MRI images and models of bone implantation systems are examined using medical parallel processing equipment.
- Other uses include broken coding, geological research, animated graphics, computer fluid dynamics, chemistry, the science of physics, electronic styling, and climatology.

3. Multiprocessing

The capacity of a device to support more than one processor at the same time is referred to as multiprocessing. In a multiprocessing method, applications are divided into smaller chunks of code that run independently. The operating system assigns these threads to the processors, which improves system performance. It does two things at once: it runs code on multiple CPUs at the same time, or it runs code on the same CPU and achieves speedups by using "wasted" CPU cycles while the software is waiting for external resources such as file loading, API calls, and so on.

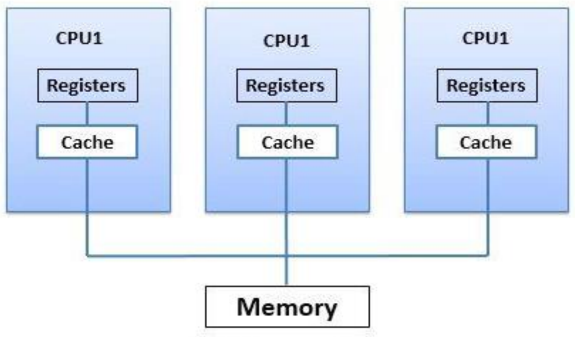

Figure 2: Structure of Multiprocessing

3.1 The Benefits of Multiprocessing

- Enhanced Throughput: More work can be completed in the same amount of time by increasing the number of processors.
- Saving money by sharing memory, buses, peripherals, and so on: When opposed to multiple single systems, a multiprocessor system saves resources. Furthermore, if numerous programs run on the same data, it is less expensive to store the data on a single disk shared by all processors in the system rather than using several copies of the same data.
- In this method increase reliability, since the ability is spread over many processors, the reliability is increased. If one of the processors fails, the system's speed will be slightly slowed, but the system will continue to function normally.

4. Python

For learning as well as actual world programming, Python is an appropriate language. Python is Guido van Rossum's strong object-oriented language of programming. The language designs allow the user to write simple programs on large and small scales. [5]. Python supports many programming paradigms, including object-oriented, mandatory, functional, or procedural types. Python supports the essential feature. Python supports an automatic memory management system of a dynamic kind and has broad and extensive standard libraries. Many operating systems have Python interpreters available.

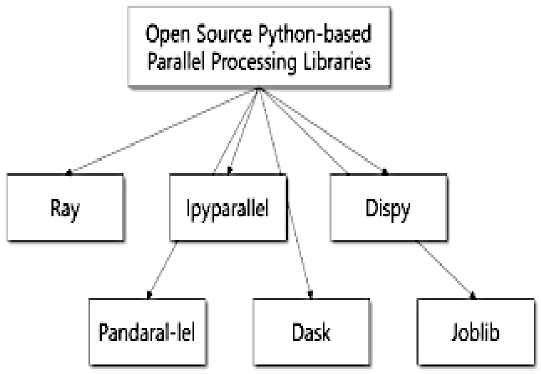

Figure 2: Python-based Parallel Processing Libraries [16] .

4.1 Python Libraries for Parallel Processing and Multiprocessing systems.

In this section, some Python Libraries will be discussed. Python Programming Language provides a standard library as well as a variety of libraries for parallel processing and multiprocessing system.

a. **The multiprocessing library:** It allows parallel processing in which multiple processes with different input arguments can be produced from a single function. On the other hand, the process library allows external processes such as another

Python script or a C/C++ execution from a Python script. [17].

b. **JMetalPy:** Create an environment for solving multi-objective optimization problems using traditional meta-heuristics and techniques for preference articulation and emotional problems, as well as a rich set of features and real-time and interactive visualization. JMetalPy also supports parallel computing in multicore and cluster systems. [18].

c. **Parsl:** It is a Python-based parallel scripting library that facilitates data-oriented workflows that are both asynchronous and implicitly parallel. Using Swift's model as a foundation [12], Parsl extends Python scripts (or applications) with advanced parallel workflow capabilities. Parsl scripting links selected python functions, and external applications (called apps), with shared input/output data objects in versatile parallel workflows. Parsl summarizes the execution environment for multi-core processors, clusters, and supercomputers [19].

d. **Ray**: Is an open-source Python parallel and distributed library. Ray provides a cohesive interface for expressing in cooperation calculations that are parallel to the role and actor based on a single dynamic implementation motor. Ray monitors the system's control state using a distributed scheduler and a parallel fault-tolerant store to meet performance requirements. [20].

e. **PyWren**: an open-source project that runs user-supplied Python code and dependencies as server-less activities on a server-less platform. PyWren performs server-less actions at a large scale and tracks the effects without needing awareness of how they are invoked and run. PyWren provides a client that operates locally and a runtime deployed as a server-less action in the cloud. PyWren uses object storage to communicate between the client and server sides. PyWren, on the client-side, serializes Python code and related data and stores it in object storage. The client instructs the stored actions to run concurrently and then awaits the output. PyWren takes the code and processes the related data from object storage on the server-side, saving the output. [21].

f. **PyNetLogo**: Is a connector. The Python general-purpose programming language will be used to handle NetLogo. Due to Python's increasing demand in the field of IT in general, the analysts and modelers have the ability to choose within many different selections. PyNetLogo features include monitoring using one of NetLogo's example frameworks in an interactive Python environment to conduct a global data analysis with parallel processing. [22].

5. Literature Review

This paper reviewed many papers related to multiprocessing and parallel processing issues that Python solved. And demonstrates how multiprocessing and parallel processing can significantly reduce calculation time by using python libraries.

D. Meunier *et al.*, [23] NetLogo can be controlled through the programming language of Python. Given Python's growing popularity in computer science, modelers and analysts now have more choices. PyNetLogo features include controlling one of NetLogo's example models from an integrated Python environment and performing a global sensitivity analysis with parallel processing.

J. Kready *et al.*, [24] This paper proposes an implementation using multiprocessing from Python to process a parallel application for the YouTube Data API. First, parallel data collection from YouTube. The tests show that multiprocessing increases the output by 400 percent with parallel processing for YouTube data collection. These enhancements minimize calculation time by using multi-threaded CPUs.

J. Niruthika *et al.*, [25] The output of the parallel Aho-

Corasick algorithm was compared to that of the serial version. Aho-Corasick is a well-known algorithm that solves the problem of exact string matching, which is a significant problem in the field of computer science. The results show that parallel Aho-Corasick implemented in Python has a lower time performance than its serial counterpart, while parallel Aho-Corasick implemented in C has a higher time performance than its serial counterpart. As a result, Python is unsuitable for parallelizing the Aho-Corasick algorithm since the algorithm's CPU consumption may be significant compared to its I/O usage.

A. Benítez-Hidalgo *et al .,* [26] jMetalPy is implemented in a Python-based multi-objective optimization system with meta-heuristics. It is distributed under the MIT license and is freely available to the public on GitHub. They presented and discussed the central architecture of the NSGA-II program and some of its variants as ample examples of how to use this framework. Dynamic optimization, parallelism, and data processing decision-making are all assisted by Metal.

Y. Babuji et al., [27] Parsl is a parallel script library that extends Python through fast, scalable, and adaptable encoding parallels. Experimental results on computing in Blue Waters show that Python scripts can run components of just 5 MS overhead, scale to over 250,000 employees across more than 8,000 nodes, and process up to 1200 tasks a second. It has shown multitasking, collaborative, web-based, and machine learning skills in biology, cosmology, and materials science.

D. S. Wahyuni [28] The BayesFactorFMRI tool, written in R and Python, was presented to enable Bayesian second-level analysis and Bayesian meta-analysis for multiprocessing fMRI image data by neuroimaging researchers. This tool accelerates computer-intensive Bayesian fMRI analysis by using multiprocessing. Its graphical user interface (GUI) enables researchers to conduct Bayesian fMRI analysis without the need for computer programming expertise. BayesFactorFMRI can be downloaded for free from Zenodo and GitHub. Neuroimaging researchers who wish to analyze their fMRI data with Bayesian analysis will usually use it, as it is more sensitive than conventional analysis and increases efficiency by spreading analytical tasks across multiple processors.

G. Heine, T *et al.,* [29] Introduced a method for asynchronous streaming. Stream subscriptions are proposed as a tool for monitoring public opinion. A prototype is presented that integrates Twitter sources, Python text processing, and Cassandra storage methods, with three main points elaborated on: 1) A comparison of results in writing techniques. 2) Data parallelization and asynchronous concurrent database writes are used in multiprocessing procedures. 3) Monitoring of public opinion by noun extraction.

D. Datta *et al.,* [30] The performance of parallelized CPUs was compared. Python's Ray library is used to parallelize multicore CPUs. In this project, the benchmark image classification algorithm used is based Convolutional Neural Network. The author attempted to demonstrate the Parallelization of a CPU's multicores which allows for faster training of a model. In this paper, a comparison analysis was conducted between three different Convolutional Neural Network models.

T. Shaffer *et al.,* [31] Native Python functions were proposed at scale, and techniques for dynamically evaluating a minimal collection of dependencies and assembling a lightweight function monitor (LFM) that captures the software environment and manages resources at the granularity of single functions were introduced. The author tests these approaches in various settings, from a campus cluster to a supercomputer, and demonstrates that their advanced dependency management planning and complex resource management strategies outperform the competition.

E. Jonas *et al.,* [32] Introduced the MPI Python connect with the standard MPI communication API, known as mpiPython. The author discussed the design issues associated with the implementation of the mpiPython API in this paper. The second part of the paper addressed the node/parallel output to compare mpiPython with other MPI bindings on a Linux cluster.

Galvez *et al.,* [33] CharmPy was introduced as a parallel programming model and application based on the Python programming language. It had many distinguishing features, including a simpler model and API, improved flexibility, and writing anything in Python. Another example is a general-purpose distributed map function that can run independent jobs on multiple nodes simultaneously and supports load balancing. The authors also demonstrated how to use CharmPy to write parallel Python applications that scale to massive core counts on supercomputers and perform similarly to MPI or C++ versions.

R. Eggen *et al,.* [34] The effect of the global python interpreter lock (GIL) has been examined. To show the effect of GIL, the authors analyze a comparison of python threads to python processes. The GIL leads to sequential execution of threads, while concurrent processing is executed. Processes need more start-up time; it answers the amount of data needed to execute processes faster than threads.

M. R. Rizqullah *et al.,* [35] The middleware in this paper was developed using the Python parallel programming language and installed on a Raspberry Pi 3. The console frame was designed to help people learn the basics of IoT through the transmission and receipt of control data to access sensors or actuators. This middleware transforms a command line for running or accessing the various IoT module features. In order to increase program operating time performance, Python employs multiprocessing or multithreading.

V. Skorpil et al., [36] The paper discussed various methods for parallelizing genetic algorithms with subsequent implementation. For example purposes, the Python programming language is used. Various models of genetic algorithm parallel processing are also provided and described. The Python implementations of the models are then defined and compared using iteration count as a criterion. While individual model output can only be compared to a certain degree, all parallel models outperform the simple serial model.

H. Jan *et al.,* [37] In this article, the NetLogo connector was initially introduced, which connects the NetLogo modeling agent to a Python environment. This was illustrated with one of NetLogo's sample versions. The library SALib Python was used as an example of the more complex tests given in a Python GUI in Sobol's variance-based structural reliability analysis of the model. For better results in the study, the ipyparallel library was used to parallel sequential simulations.

Zhang et al., [38] This paper proposed Quant Cloud, a program that integrates a parallel Python framework with a C++-coded Big Data system. This extensive data framework is built in C++, and the user methods are written in Python. A coprocessor-based parallel strategy underpins the automatic parallel execution of Python code. They have put the program into two popular algorithms: moving window and self-adjusting average movements (ARMA). The Intel Xeon E5 and Xeon Phi processors are thoroughly compared. Their approach to parallelization is almost linear and is suitable for today's multicore processors. The findings show that their method is almost linear.

Sindhu *et al.,*[39] A Python multi-processing library has implemented a simultaneous implementation of the Max-p problem. The author achieves speeds up to 12 and 19 times with the best sequential algorithm for developing and improving phases utilizing an intuitive multi-lock data structure. In order to validate the algorithms, the author provides detailed experiential results.

Real et al., [40] This paper has presented Auto Parallel which is a Python module that facilitates parallelism and runs on distributed infrastructures. It is built on top of PyCOMP and is sequential, This helps in making it easy to scale up to hundreds of cores for creative purposes. Users can specify the affine loop on sequential methods using the @parallel annotation instead of testifying sequential python code. As it turns out, the generated codes for Choles, LU, and QR algorithms can achieve similar performance without any effort from the programmer. Thus, taking the Auto Pip parallelizes distributed systems one step further.

Z. Rinkevicius *et al* ., [41] Prsened an Open source software named VeloxChem that was created to measure electronic complicated, real linear response functions for functional theories of Hartree–Fock and Kohn–Sham density. Points to an objective software framework written in Python/C++ layered fashion, VeloxChem enables the time-efficiently prototyping of new techniques without cooperating computational achievement.

V. Canh Vu et al,. [42] In parallel, a genetic programming technique for classifying data patterns for wireless attack detection was presented. The author performed tests on the same computer system configuration, parameters and datasets in order to associate the performance of Karoo GP and standard GP. Karoo GP was, however, implemented alongside the high-speed GPU processing mechanism when the mainstream GP for multi-core CPUs has been used. Karoo GP is much faster than its average GP, according to performance.

S. Khan and A. Latif [43] Proposed solution eliminates this constraint and allows a single machine to run several instances. The SIME method for the measurement of critical clearance time (CCT) and the stability of the rotor angles is measured on a piece of single infinite system equipment (SIME). This method reduces computational time as a parallel factor and dramatically improves the handling and aggregation of the tasks. The approach is generic and possible.

A. V. M. Barone et al., [44] Introduce a broad and diverse Parallel Corpus with its documentation strings ("docstrings") created by scrapping open source repositories on GitHub, with a hundred thousand Python functionalities. The paper defined the fundamental results in neural machine-created translations for the code documentation and code generation tasks. To further increase the number of training information

Table 1: Summary of Review Papers Based on Parallel Processing and Multiprocessing in Python

Ref.	Year	Objectives	Methods / Tools	Research Problem	Applied Field
[23]	2020	Create parallel processing pipelines that can be shared and keep track of all analyses.	NeuroPycon	Multi-modal and decided reproducible brain connectivity pipelines	Health Care (Brain Pipelines)
[24]	2020	Reduce computation time of YouTube Data API request in parallel.	Python	The requests from the YouTube Data API take time	Multimedia (YouTube)
[25]	2019	Checking the performance of the parallel version of the Aho-Corasick algorithm against its serial version	Pyrhon.	Problem of Exact String Matching	Electronic Dictionary (Aho–Corasick)
[26]	2019	Create multi-object optimizations like quick prototyping facilities and a vast number of data libraries available, and support multi-core and cluster systems for parallel computing processing, analyzing, and viewing.	jMetalPy	Multi-Objective Optimization with Met-Heuristics	Engineering, Economics and Logistics
[27]	2019	Build a dynamic component dependency graph that can then run effectively on one or more processors	Parsl	Encoding parallelism	Biology, cosmology, and materials science.
[28]	2021	Comparing Bayesian meta-analysis of fMRI image data with multiprocessing with serial analysis.	BayesFactorFMRI	Perform Bayesian second-level analysis and Bayesian meta-analysis	Image Processing
[29]	2018	Combining Twitter streams, by Python, Multiprocessing procedures employing data parallelization.	Python	Asynchronous streaming Stream subscriptions	Multimedia (Twitter)
[30]	2020	Comparing the performance of parallelized CPUs	Python's Ray library	Huge Convolutional Neural Networks	Image Processing

[31]	2021	Resource management in a distributed system raises issues relating to granular parallelism, management of software environments, and	Parsl	Dependency management planning and complex resource management strategies	Complex application at supercomputer - scale.
[32]	2020	Creates a message passing Python linking interface to facilitate parallel	Python	Big Data	Media, Data Science, Physics, Healthcare.
[33]	2018	Write parallel Python apps with CharmPy, which	CharmPy	Distributed asynchronous execution-	Computation and Communicatio
[34]	2019	Examine Thread and multi-Process Efficiency in	Python	Thread and Process Efficiency in	Security
[35]	2019	Creating an App, A console program to help people	Python	Difficulties of command line command for	Console Application
[36]	2019	The use of genetic algorithm parallel processing that is	Master-Slave	Algorithm speed and load distribution	Genetic Algorithms
[37]	2019	To manage one of NetLogo's examples from a Python-interactive environment, perform a parallel processing global sensitivity analysis.	PyNetLogo	global sensitivity analysis of Net Lgo	Controling the Communication and Linking
[38]	2018	Coding Big Data system in C++.	Pyrhon	Data Analysis and	Finance
[39]	2018	Max-p problem parallel implementation	Python	Max-P Area Efficiency and Synergy	Geospatial
[40]	2019	facilitates parallelism and runs on distributed infrastructural infrastructures	Python	Improving how users cannot deal with distributed and parallel computing problems directly.	Programming language
[41]	2020	Calculating real and complex electronic linear responses at the Hartree–Fock and Kohn–Sham density stages. Computer effectiveness sacrifice	Python	Execution in cluster environments with high efficiency.	Spectroscopy simulations
[42]	2018	Parallel to the classification of data patterns for the identification of wireless attacks	(Karoo GP)	Classify data patterns for wireless attacks	Security
[43]	2019	Reduce computer time as a parallel factor and greatly increase handling and aggregation of results	Python	Software Instance Modular and Scalable.	PowerFactory
[44]	2017	Introduce the extensive and diverse parallel corpus with its documentation ("docstrings") of one hundred 000 Python functions, provided by removing the GitHub open source repository.	Python	The nature and the production of code is not reprehensible by the current company.	Documentation and code generation

6. Discussion

Increased use of Python and other high-level programming languages calls for intuitive interfaces in libraries written in different components in the lower languages and applications. In combination with the growing need for parallel computation (for example because of big data and the end of Moore's law), this change to orchestration instead of execution calls for a revision on how parallelism in programs is interpreted. In order to compare the performance of the python libraries in parallel processing and multiprocessing fields, we reviewed some papers which used python libraries for the purpose of parallel processing and multiprocessing. As a result, some of the researcher proposed a python based new software such as VeloxChem to be used for Actual and complex electronic response functions calculation. Moreover, Parsl, a parallel scripting library is used by the author Babuji [19] for constructing a dynamic dependency graph of components. On the other hand, Shffere [31], worked with Parsl scripting for issues relating to granular parallelism, management of software environments, and adaptation to computing. Foremother, Python parallel scripting language used for Internet of Things (IoT) console applications. This survey aims to concentrate on python open sources language libraries used in different parallel and multiprocessing systems. There are numerous feedbacks and so, they can be used widely in the future.

7. Conclusion

This paper demonstrated that Python is a new, mature, complete, and scalable scripting language that is well-suited to scientific research and education in power system analysis. Python's programming language provides the resources needed to run parallel code on multicore machines. Throughout the paper, several Python libraries were discussed that are used in parallel and multiprocessing in various approaches.

Multimedia, websites, massive core counts on supercomputers, genetic algorithms, attack detection, and so on were all reviewed in the related work section. It was stated that Python has features for spreading work between multiple processes, allowing it to take advantage of multiple CPU cores and larger quantities of usable machine memory.

8. Reference

1. S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *Int. J. Mach. Learn. Comput.*, vol. 7, no. 1, pp. 9–12, 2017, doi: 10.18178/ijmlc.2017.7.1.611.
2. I. V. Kotenko, I. B. Saenko, and A. G. Kushnerevich, "Architecture of the parallel big data processing system for security monitoring of internet of things networks," *SPIIRAS Proc.*, vol. 4, no. 59, pp. 5–30, 2018, doi: 10.15622/sp.59.1.
3. A. Ebrahim, J. A. Lerman, B. O. Palsson, and D. R. Hyduke, "COBRApy: COnstraints-Based Reconstruction and Analysis for Python," *BMC Syst. Biol.*, vol. 7, 2013, doi: 10.1186/1752-0509-7-74.
4. L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using Python," *Adv. Water Resour.*, vol. 34, no. 9, pp. 1124–1139, 2011, doi: 10.1016/j.advwatres.2011.04.013.
5. A. Rogohult, "Benchmarking Python Interpreters," *KTH, Sk. för datavetenskap och Kommun. (CSC), 2016*, p. 119, 2016.
6. W. Y. Je, S. Teh, and H. R. Chern, "A parallelizable chaos-based true random number generator," 2018.
7. [7] W. K. Lee, R. C. W. Phan, W. S. Yap, and B. M. Goi, "SPRING: a novel parallel chaos-based image encryption scheme," *Nonlinear Dyn.*, vol. 92, no. 2, pp. 575–593, 2018, doi: 10.1007/s11071-018-4076-6.
8. T. Wang and Q. Kemao, "Parallel computing in experimental mechanics and optical measurement: A review (II)," *Opt. Lasers Eng.*, vol. 104, no. June 2017, pp. 181–191, 2018, doi: 10.1016/j.optlaseng.2017.06.002.
9. E. Tejedor *et al.*, "PyCOMPSs: Parallel computational workflows in Python," *Int. J. High Perform. Comput. Appl.*, vol. 31, no. 1, pp. 66–82, 2017, doi: 10.1177/1094342015594678.
10. R. Filguiera, A. Krause, M. Atkinson, I. Klampanos, and A. Moreno, "Dispel4py: A Python framework for data-intensive scientific computing," *Int. J. High Perform. Comput. Appl.*, vol. 31, no. 4, pp. 316–334, 2017, doi: 10.1177/1094342016649766.
11. S. R. M. Zebari and N. O. Yaseen, "Effects of Parallel Processing Implementation on Balanced Load-Division Depending on Distributed Memory Systems Client / Server Principles," vol. 5, no. 3, 2011.
12. M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Comput.*, vol. 37, no. 9, pp. 633–652, 2011, doi: 10.1016/j.parco.2011.05.005.
13. S. Y. Yu, S. R. Chhetri, A. Canedo, P. Goyal, and M. A. Al Faruque, "Pykg2vec: A python library for knowledge graph embedding," *arXiv*, vol. 22, pp. 1–6, 2019.
14. C. Evangelinos and C. Hill, "Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2," *Ratio*, vol. 2, Jan. 2008.
15. K. Asanovíc *et al.*, "The Landscape of Parallel Computing Research : A View from Berkeley," pp. 1–54, 2006.
16. T. Kim, "Survey and Performance Test of Python-based Libraries for Parallel Processing," pp. 1–4, 2020.
17. N. Singh, L. M. Browne, and R. Butler, "Parallel astronomical data processing with Python: Recipes for multicore machines," *Astron. Comput.*, vol. 2, pp. 1–10, 2013, doi: 10.1016/j.ascom.2013.04.002.
18. B. Lewis, I. Smith, M. Fowler, and J. Licato, "The robot mafia: A test environment for deceptive robots," *28th Mod. Artif. Intell. Cogn. Sci. Conf. MAICS 2017*, pp. 189–190, 2017, doi: 10.1145/1235.
19. Y. Babuji *et al.*, "Introducing Parsl: A Python Parallel Scripting Library," pp. 1–2, 2017, [Online]. Available: https://doi.org/10.5281/zenodo.891533#.WdOPKS_nvdE.mendeley.
20. P. Moritz *et al.*, "Ray : A Distributed Framework for Emerging AI Applications This paper is included in the Proceedings of the," *USENIX Symp. Oper. Syst. Des. Implement.*, 2018.
21. J. Sampé, G. Vernik, M. Sánchez-Artigas, and P. García-López, "Serverless data analytics in the IBM cloud," *Middlew. Ind. 2018 - Proc. 2018 ACM/IFIP/USENIX Middlew. Conf. (Industrial Track)*, no. October 2019, pp. 1–8, 2018, doi: 10.1145/3284028.3284029.
22. M. Jaxa-Rozen and J. Kwakkel, "PyNetLogo: Linking NetLogo with Python," *J. Artif. Soc. Soc. Simul.*, vol. 21, Mar. 2018, doi: 10.18564/jasss.3668.
23. D. Meunier *et al.*, "NeuroPycon: An open-source python toolbox for fast multi-modal and reproducible brain connectivity pipelines," *Neuroimage*, vol. 219, no. June, 2020, doi: 10.1016/j.neuroimage.2020.117020.
24. J. Kready, S. A. Shimray, M. N. Hussain, and N. Agarwal, "YouTube data collection using parallel processing," *Proc. - 2020 IEEE 34th Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2020*, pp. 1119–1122, 2020, doi: 10.1109/IPDPSW50202.2020.00185.
25. J. Niruthika and S. Pranavan, "222 implementation of parallel aho-corasick algorithm in python," no. November, pp. 222–233, 2019.
26. A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, and J. Del Ser, "jMetalPy: A python framework for multi-objective optimization with metaheuristics," *arXiv*, 2019.
27. Y. Babuji *et al.*, "Parsl: Pervasive parallel programming in Python," *HPDC 2019- Proc. 28th Int. Symp. High-Performance Parallel Distrib. Comput.*, pp. 25–36, 2019, doi: 10.1145/3307681.3325400.
28. H. Han, "BayesFactorFMRI: Implementing Bayesian Second-Level fMRI Analysis with Multiple Comparison Correction and Bayesian Meta-Analysis of fMRI Images with Multiprocessing," *J. Open Res. Softw.*, vol. 9, no. 1, pp. 1–7, 2021, doi: 10.5334/jors.328.
29. G. Heine, T. Woltron, and W. Alexander, "Towards a Scalable Data-Intensive Text Processing Architecture with Python and Towards a Scalable Data-Intensive Text Processing Architecture with Python and Cassandra," no. November, 2018.
30. D. Datta, D. Mittal, N. P. Mathew, and J. Sairabanu, "Comparison of Performance of Parallel Computation of CPU Cores on CNN model," *Int. Conf. Emerg. Trends Inf. Technol. Eng. ic-ETITE 2020*, pp. 1–8, 2020, doi: 10.1109/ic-ETITE47903.2020.142.
31. T. Shaffer, Z. Li, B. Tovar, and Y. Babuji, "Lightweight Function Monitors for Fine-Grained Management in Large Scale Python Applications," pp. 1–11.
32. H. Park, J. Denio, J. Choi, and H. Lee, "MpiPython: A robust python MPI binding," *Proc. - 3rd Int. Conf. Inf. Comput. Technol. ICICT 2020*, pp. 96–101, 2020, doi: 10.1109/ICICT50521.2020.00023.
33. J. J. Galvez, K. Senthil, and L. Kale, "CharmPy: A Python Parallel Programming Model," *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, vol. 2018-Septe, pp. 423–433, 2018, doi: 10.1109/CLUSTER.2018.00059.
34. R. Eggen and E. M. Eggen, "Thread and Process Efficiency in Python," pp. 32–36.
35. M. R. Rizqullah, A. R. Anom Besari, I. Kurnianto Wibowo, R. Setiawan, and D. Agata, "Design and implementation of middleware system for IoT devices based on raspberry Pi," *Int. Electron. Symp. Knowl. Creat. Intell. Comput. IES-KCIC 2018 - Proc.*, pp. 229–234, 2019, doi: 10.1109/KCIC.2018.8628528.
36. V. Skorpil, V. Oujezsky, P. Cika, and M. Tuleja, "Parallel Processing of Genetic Algorithms in Python Language," *Prog.*

Electromagn. Res. Symp., vol. 2019-June, pp. 3727–3731, 2019, doi: 10.1109/PIERS-Spring46901.2019.9017332.

37. H. Jan, J. H. Pynetlogo, L. Netlogo, M. Jaxa-rozen, and J. H. Kwakkel, "Article PyNetLogo : Linking NetLogo with Python Reference PyNetLogo : Linking NetLogo with Python," vol. 21, no. 2.
38. P. Zhang, Y. Gao, and X. Shi, "QuantCloud: A software with automated parallel python for Quantitative Finance applications," *Proc. - 2018 IEEE 18th Int. Conf. Softw. Qual. Reliab. Secur. QRS 2018*, pp. 388–396, 2018, doi: 10.1109/QRS.2018.00052.
39. V. Sindhu, "ScholarWorks @ Georgia State University Exploring Parallel Efficiency and Synergy for Max-P Region Problem Using Python," 2018.
40. F. Real, A. Batou, T. Ritto, and C. Desceliers, "Stochastic modeling for hysteretic bit–rock interaction of a drill string under torsional vibrations," *J. Vib. Control*, p. 107754631982824, 2019, doi: 10.1177/ToBeAssigned.
41. Z. Rinkevicius *et al.*, "VeloxChem: A Python-driven density-functional theory program for spectroscopy simulations in high-performance computing environments," *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, vol. 10, no. 5, pp. 1–14, 2020, doi: 10.1002/wcms.1457.
42. V. Canh Vu and T. H. Hoang, "Detect Wi-Fi Network Attacks Using Parallel Genetic Programming," *Proc. 2018 10th Int. Conf. Knowl. Syst. Eng. KSE 2018*, pp. 370–375, 2018, doi: 10.1109/KSE.2018.8573378.
43. S. Khan and A. Latif, "Python based scenario design and parallel simulation method for transient rotor angle stability assessment in PowerFactory," *2019 IEEE Milan PowerTech, PowerTech 2019*, pp. 1–6, 2019, doi: 10.1109/PTC.2019.8810949.
44. A. V. M. Barone and R. Sennrich, "A parallel corpus of Python functions and documentation strings for automated code documentation and code generation," *arXiv*, 2017.