Resource Consumption of Different Programming Languages

Alex Saelee
University of Anchorage Alaska
Anchorage, United States
acsaelee@alaska.edu

Brian Fuentes
University of Anchorage Alaska
Anchorage, United States
bcuevasfuentes@alaska.edu

Killian Hull
University of Anchorage Alaska
Anchorage, United States
kghull2@alaska.edu

Abstract

This study evaluates the performance and energy efficiency of Python, C++ and C# by executing equivalent algorithms across each language. The languages were measured by benchmarking runtime and estimated the energy consumption for five common tasks. The results indicate surprising differences with the data. The goal of this paper is to provide a general view of the performance to inform programmers of the resource consumption of these three major programming languages.

Keywords Programming Languages, Runtime, Energy Efficiency

1 Introduction

In modern computing, the performance and efficiency of software is becoming increasingly important by the choice of programming languages. While high-level languages offer rapid development, ease of use, and readability, they can introduce drastic overhead that affects the speed of runtime and energy consumption. With the growing scale of data-intensive applications, such as artificial intelligence, the rise of energy-conscious computing such as embedded systems, mobile devices, electric vehicles, and largescale cloud systems has become ever more significant for the prosperity of businesses and the planet.

Although many developers rely on benchmarks to compare programming languages, these comparisons focus mainly on the runtime performance which neglects the emphasis of energy efficiency. This paper aims to address this barrier by comparing three modern, popular languages: Python, C++, and C#. We will explore how these languages handle the same identical computational tasks and analyze the runtime and estimated energy consumption.

We will reflect on these algorithms for their relevance in everyday coding and their ability to highlight performance trade-offs in different types of computation. There are a lot of factors that programmers are concerned with such as memory management, readability, scalability, and much more, however, our results are intended to allow developers, researchers and students to choose the most efficient programming language for time efficiency and energy-sensitive applications.

2 Methodologies

To evaluate the runtime and energy efficiency of Python, C++ and C#, we implemented five benchmark programs in each language. The chosen algorithms represent common programming tasks found in everyday software development:

1. Summation Loop: A simple loop that iterates from 1 to n, summing the values.
2. Recursive Fibonacci: A classic recursive function to compute the *nth* Fibonacci number.
3. Linear Search: A function that iterates through an unsorted list to find a specific value.
4. Quick Sort: A function that organizes an

unsorted list.

5. Matrix Multiplication: A function that multiple dimensions of an array.

Each of the five algorithms were written using native syntax and constructs without optimization to ensure the least amount of biasness. Programs were designed to run the same logic flow and algorithmic structure. Because each language has different paradigms and have different interpretations, runtime results are expected to be different. The impact of these factors will play a crucial role in determining the most effective language for certain applications.

Table 1. Paradigms of programming languages

Features	**Python**	**C++**	**C#**
Object Oriented Programming	Yes	Yes	Yes
Memory Management	Automatic	Manual	Automatic
Compilation Type	Interpreted	Machine Code	Intermediate Language (JIT)
Procedural	Yes	No	Yes

2. 2 Hardware Systems

The following experiments and test runs were used:

- Python: Visual Studio using Python interpreter
- C++: Visual Studio Code with g++ compiler
- C#: Visual Studio Code with .NET

The same input size was maintained across all algorithms and languages to ensure fair conditions when executing. Furthermore, programs were loop over a value N to ensure the average time was accurate. The experiments were executed on a personal computer using Windows 11. The computer has a Snapdragon® X10-core X1P64100 processor clocked at 3.40 GHz, 16 GB of RAM, and 512 GB storage drive.

2. 3 Runtime and Power Measurements

Each of the programs executed used internal timers to measure runtime in seconds and microseconds. Since measuring exact power draw per program is complex without additional hardware-level tools, we assumed power consumption was proportional to runtime under consistent CPU load.

3 Results

The energy consumption of a program can be expressed using the following equation where:

$$Energy(J) = Power(W) * Time(S)$$

W = Watts

J = Joules

S = Seconds

This equation implies that by reducing the runtime of a program, the total energy consumed decreases. Assuming if the power drawn remains constant for each execution dependent on the programming language. However, this assumption can be misleading in real-world scenarios. Measuring the exact amount of power consumed by a program is challenging because the difference is often miniscule or requires additional hardware especially for short or lightweight tasks. Moreover, modern CPUs and runtime environments frequently scale power consumption dynamically based on the program's workload so something as simple as a for loop may have negligible values. Hence, a general assumption will be made about the average amount of energy used for each language.

From our benchmark experiments, we observed drastic differences in performance between Python, C++, and C#, particularly when executing compute – intensive tasks such as generating the Fibonacci sequence. Python consistently performed the slowest, which implies a higher total energy usage due to its longer runtime, even if the power draw per unit of time is low. To estimate the amount of energy used by either of the three programs, and to prove that compiled languages are more efficient, we will imply that C++ consumes 1J, C# consumes 1.5J, and Python consumes 3J. Given the equation for energy, we will use Figure 1 for the time values at N = 10000 to complete Table 2.

Table 2. Estimated energy for summing loop

Programming Language	Estimated Consumed Power(J)	Time(S)	Energy(W)
Python	3	$1.* 10^{-8}$	$3 * 10^{-8}$
C++	1	$4 * 10^{-10}$	$4 * 10^{-10}$
C#	1.5	$5 * 10^{-10}$	$7.5 * 10^{-10}$

From a recent paper [1] that explores energy consumption with accurate, controlled, and efficient experiments, the authors states that compiled languages, in general are "the fastest and most energy efficient ones" where they consume an average of 120J. On the other hand, interpreted languages consumed either 576J or 2365J.

These results align with expectations because Python is an interpreted language with a significant amount of overhead. It does not compile directly to machine code like C++ or benefit from the ahead-of-time (AOT) compilation or just-in-time (JIT) optimizations used in C#. Additionally, Python often requires more system resources and more time to execute equivalent logic.

C++ and C# performed significantly better in CPU-bound operations like Fibonacci and sorting algorithms. C++, being a compiled language with direct memory access and no runtime overhead, generally yielded the fastest results. C#, while managed, benefitted from just-in-time compilation and performed closely to C++ in many scenarios.

3.2 Python Performance Analysis

In our experiment, Python was one of the three languages used to implement our sample programs. The expectation that Python being more likely to have the longest execution time of our languages, as indicated in Figures 1-5, are largely due to three reasons - Python being an interpreted language, Python's Global Interpreter Lock, and Python being dynamically typed. Further research into articles supported this belief, such as in the article *Energy Efficiency across Programming Languages* where the authors' research and figures display Python's poor execution times, nearly coming in last in most test cases.

The standard implementation of Python is known as CPython, and it compiles the source Python code into bytecode, which is then interpreted at runtime. Despite the compilation process, Python itself is still interpreted at runtime, making it an interpreted language. It's also been widely observed that the compilation phase is mostly concerned with load times rather than the execution speed, which is more in line with the focus of our project. As an interpreted language, it was expected that the execution time of the Python programs would be noticeably worse than the other tested languages, as compiled languages generally have much better execution times.

One of Python's biggest limitations, however, is its Global Interpreter Lock, which is a mutex used to prevent multiple threads from executing bytecode at once. This constraint is in place due to Python's memory management not being thread-safe, and as a result, it can cause noticeable loss of performance in multi-threaded programs. While no multi-threaded Python programs were tested in our project, it's still worth mentioning as it directly speaks to the general performance of Python in a broader context than our project.

In languages like C++, it's necessary to specify the type of variable when it's being declared, which allows the compiler to allocate memory and perform other optimizations. Python, however, is a

Summing Loop			
N	Python	C++	C#
10	9.2001E-06	3.00E-06	3.581E-04
100	1.88999E-05	5.00E-06	4.149E-04

1,000	1.402E-04	8.00E-06	4.050E-04
10,000	1.312E-03	4.40E-05	4.075E-04
100,000	1.31163E-02	4.06E-04	4.854E-04

Fibonacci

N	Python	C++	C#
20	3.400E-03	5.1214E-05	5.922E-05
30	6.093E-01	7.13543E-03	7.51365E-03
40	57.3372	8.70318E-01	9.4752929

Linear Search

Find N	Python	C++	C#
0	2.38937E-07	4.67E-10	3.4637E-08
10	4.33484E-07	6.36E-10	4.4477E-08
20	1.51012E-06	7.36E-10	7.8226E-08
30	1.87392E-06	1.291E-09	8.9447E-08
40	2.35049E-06	2.14E-09	1.04044E-07
50	3.28722E-06	3.604E-09	1.27116E-07

Quick Sort

Trial	Python	C++	C#
1	1.50718E-04	5.01632E-06	2.65735E-06
2	1.5822E-04	5.53733E-06	2.70491E-06
3	1.5662E-04	4.71997E-06	2.57426E-06
4	1.5446E-04	4.84765E-06	2.87147E-06
5	1.5371E-04	4.75072E-06	2.66367E-06
6	1.4993E-04	5.12064E-06	2.71682E-06

Matrix Multiplication

N x N	Python	C++	C#
10 x 10	4.00E-04	6.26E-06	5.50E-06
50 x 50	4.50E-02	6.491E-04	6.3295E-04
100 x 100	3.603E-01	3.78858E-03	1.030126E-02
250 x 250	5.6253	5.8911E-02	8.168193E-02
500 x 500	46.5864	0.55617	1.49073032
1000 x 1000	373.0545	7.483818	23.9482176

Table 3. Runtime of each program

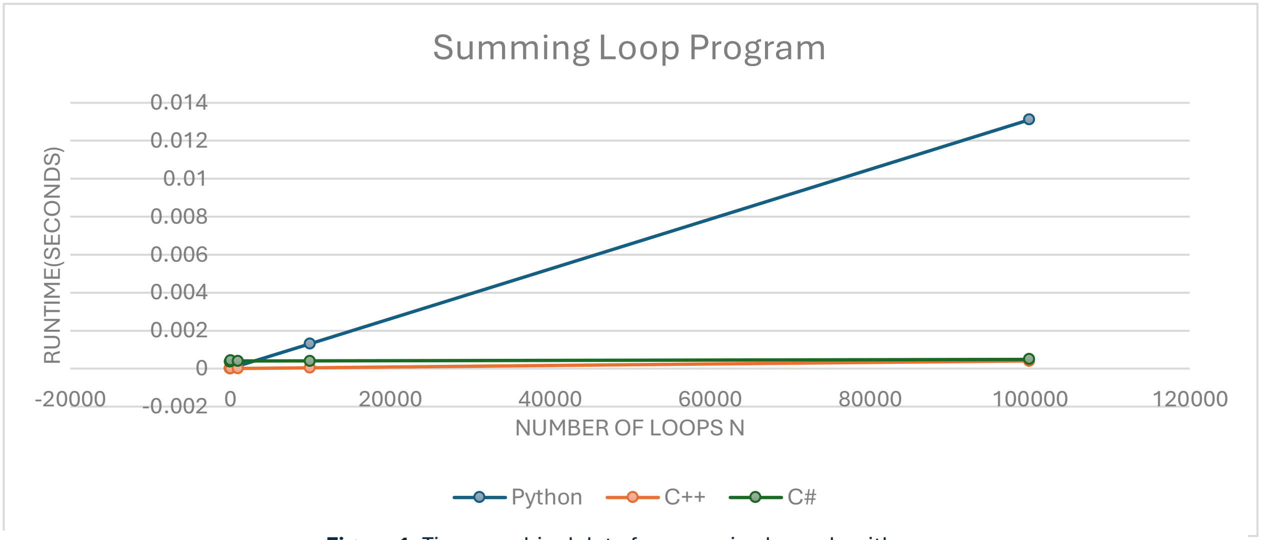

Figure 1. Time graphical data for summing loop algorithm

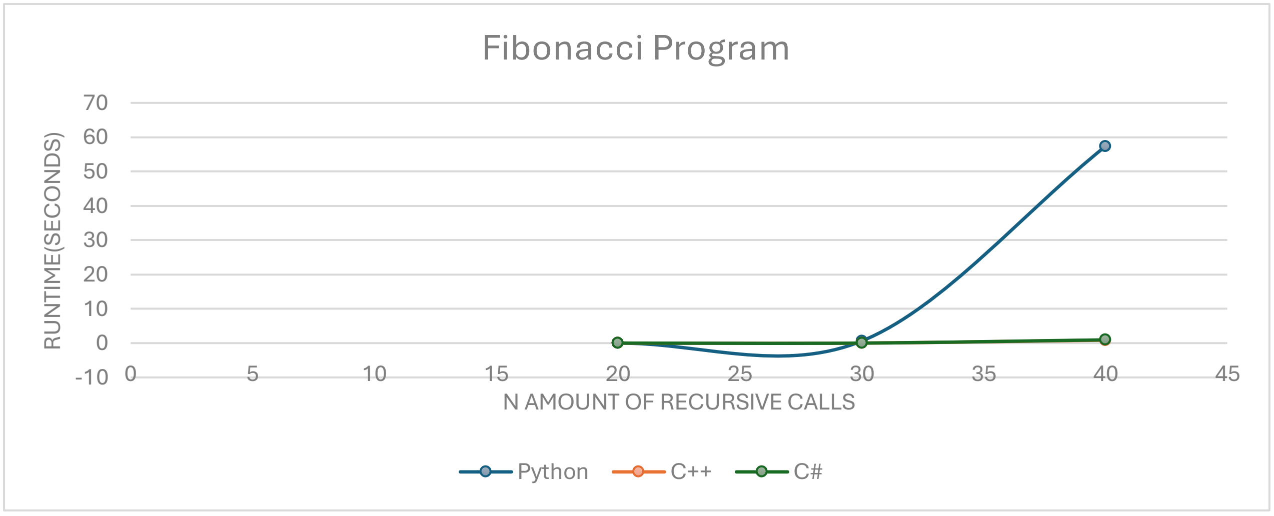

Figure 2. Time graphical data for Fibonacci algorithm

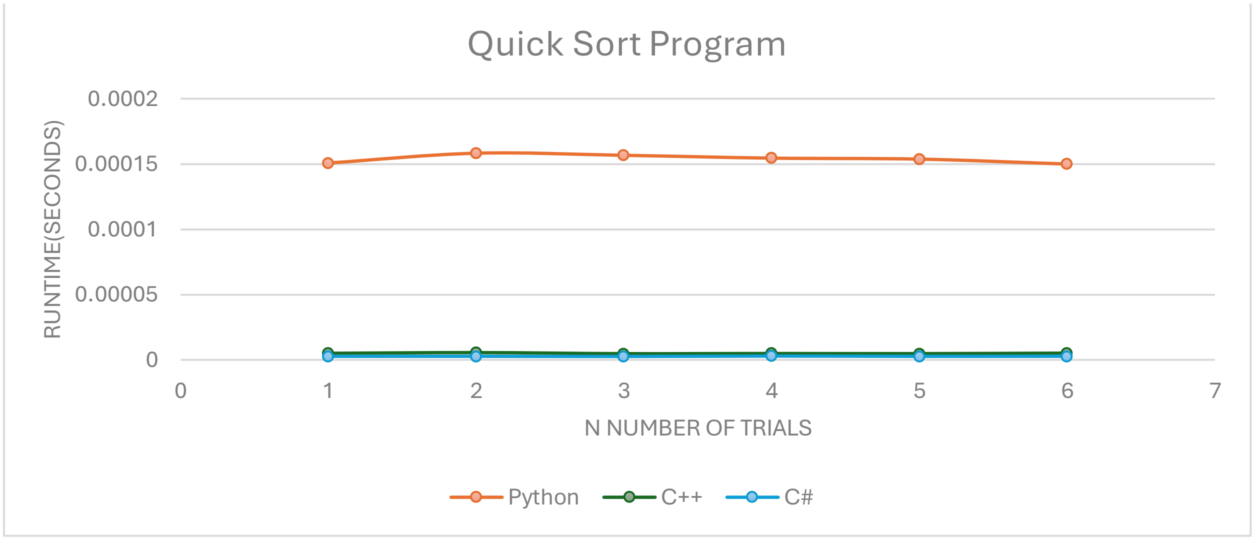

Figure 3. Time graphical data for linear search algorithm

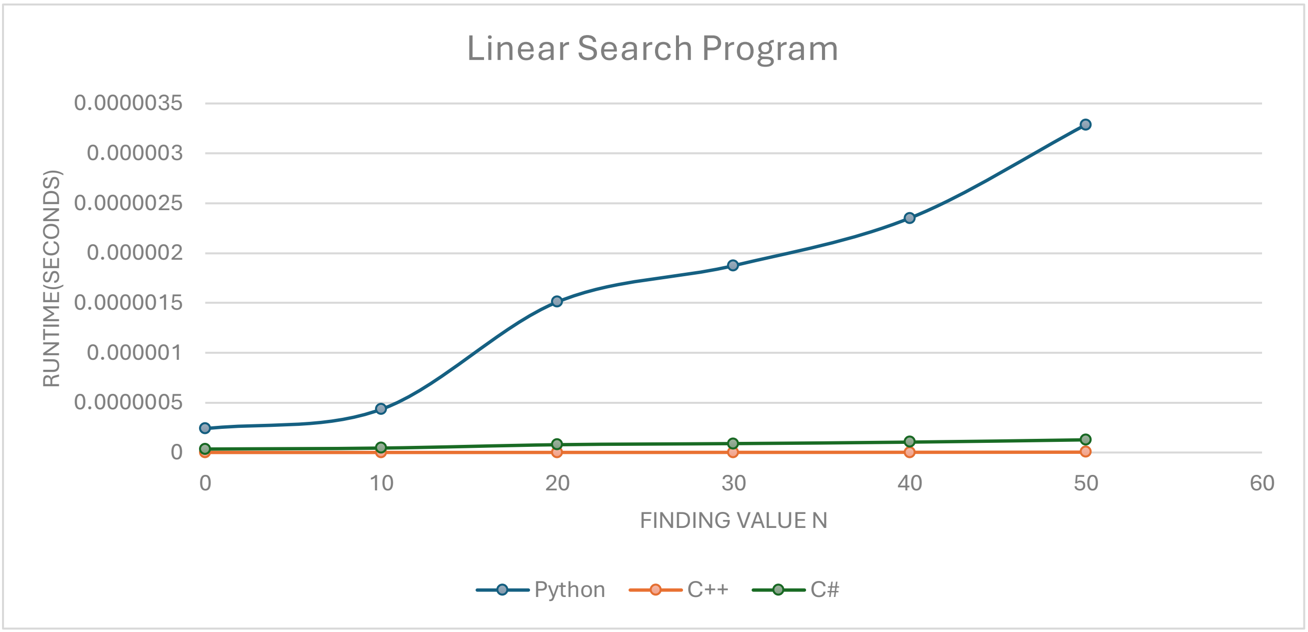

Figure 4. Time graphical data for quick sort program

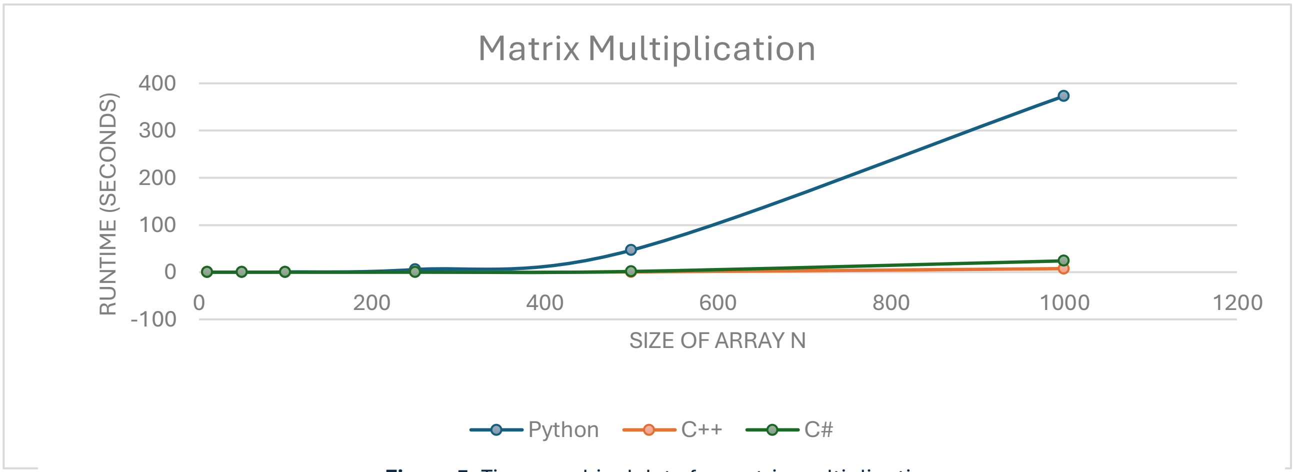

Figure 5. Time graphical data for matrix multiplication

a requirement to declare a variable's type. This feature can make code more readable and easier to write, which is a big selling point for Python, but it comes at the cost of the interpreter performing all the necessary checks during the execution which can cause distinct performance loss, especially in large loops or when working with large amounts of data. In the context of our project, Python being dynamically typed is another main reason as to why I believed it would have a worse execution time than the other languages.

3.3 C++ Performance Analysis

C++ was capable of consistently outperforming Python and C#, from Figure 1-3 and from Table 3, because of the low-level design, minimal overhead, and compiled-time optimizations. Although the paradigms, as shown from Table 1, have an influence on the runtime, major contributions that allow C++ to be faster and consume less power is a compiled language, where the source code is translated to machine code through ahead-of time compilation. This removes the necessity for interpretation or JIT compilation during runtime, resulting in an instantaneous execution without any overhead. With programs that require repetitive algorithms such as loops, C++ compilers offer advanced optimizations such as loop unrolling which drastically reduces the execution time and access overhead memory. Additionally, C++ offers manual control over memory management which consequently improves the cache hit rate especially for matrix multiplication or recursive sorting algorithms which require heavy computational requirements. For paradigm, C++ offers zero cost abstractions which allow programmers to freely access features, such as templates or object-oriented design without additional delays in the execution. There are some cases where C++ will remain slower as shown in Figure 4. This is because C# can optimize the program to improve the runtime. Although this is unlikely due to C++ having more improved key features Table 1.

4 Limitations

While this study aimed to remove biasness when comparing Python, C++ and C#, there were several limitations worth acknowledging.

First, although efforts were made to implement equivalent algorithms, differences in programming experiences, syntax, and optimizations may have introduced unfair experiments. A possible solution to this could have been better communication when designing the programs.

Second, energy consumption was estimated based on runtime and assumed power, rather than directly using a specialized tool or software. Although the assumption provides an approximation [1], the true amount of power used could be significantly higher or lower depending on the hardware and environment.

Third, this study focused on a limited number of algorithms and did not include more complex algorithms such as multi-threading or file I/O. As a result, these results could be different in real-world applications. More programming languages could have been explored but had limited knowledge.

Finally, hardware factors may have contributed to the unfair testing grounds for the experiment. Small changes in temperature and additional processes could have a contribution in altering the runtime of the results leading to unstable and unreliable data.

Conclusion

In this study, we analyzed the runtime and estimated energy consumption of three modern and popular programming languages: Python, C++, C#. By implementing identical algorithms across each of the languages, we could generate reliable results that indicated the performance of each language. The tables and figures conclude that compiled languages, like C++ and C#, were capable of outperforming interpreted languages greatly in terms of execution time and estimated energy usage. The efficiency of C++ was due to the low-level memory control, optimization capability, and lack of overhead. C# was almost as efficient as C++ but lacked the ability due to its compilation and memory management. Python performed the slowest across all benchmarks due to its interpreted language. Python's performance indicates that it is unreliable in terms of energy or computational-intensive tasks. Hence, C++ or C# would be more appropriate for these cases.

While Python is slow in terms of heavy computational tasks, Python remains efficient in some cases such as using functions that could greatly reduce the runtime of a program such as NumPy.

As a result, the amount of energy used by a programming language is dependent on many factors including hardware and compiler optimization. To control these factors and create a fair environment remains difficult due to a lack of tools and resources to facilitate the analysis. More concise experiments are required to further explore the ability to minimize energy consumption but also ensure that programs run the most efficient for vast applications.

Overall, this study aims to inform programmers about the significance of choosing a programming language for a specific task based on requirements and limitations. Programmers who are energy-conscious should consider the ethical usage of advanced programming such as artificial intelligence. As technology begins to expand, the amount of energy consumed increases yearly. By investigating the performance of these languages, programmers can minimize the harmful resource consumption of technology.

Appendix I

Alex Saelee worked on the organization of the paper and is responsible for the C++ programs. He wrote the introduction, abstract, methodologies, results, C++ performance analysis section, limitations, conclusions, and created the tables/figures.

Killian Hull worked on the Python performance analysis section and was responsible for the Python programs. Responsible for revisions of the paper and references.

Brian Cuevas Fuentes worked on the C# performance analysis section and was responsible for the C# programs.

He is also responsible for revising the paper to ensure there is no redundant information.

References

[1] R. Pereira *et al.*, "Energy efficiency across programming languages: how do energy, time, and memory relate?," *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE)*, 2017.

[2] L. Prechelt, "PLOTS AND STATISTICAL METHODS," 2000. [Online]. Available: https://www.cs.tufts.edu/~nr/cs257/archive/lutz-prechelt/comparison.pdf

[3] A. Kwame, E. Martey, and A. Gilbert, "Foundation of Computer Science FCS," *Communications on Applied Electronics (CAE)*, vol. 7, no. 7, 2017. [Online]. Available: https://www.caeaccess.org/archives/volume7/number7/kwame-2017-cae-652685.pdf

[4] I. Camargo-Henríquez, A. Martínez-Rojas, and G. Castillo-Sánchez, "Energy Optimization in Software: A Comparative Analysis of Programming Languages and Code-Execution Strategies," *2024 9th International Engineering, Sciences and Technology Conference (IESTEC)*, Panama City, Panama, 2024, pp. 507–511.

[5] Z. Porkoláb, "Save the Earth, Program in C++!," *2022 IEEE 16th International Scientific Conference on Informatics (Informatics)*, Poprad, Slovakia, 2022, pp. 11–12.

[6] L. Koedijk and A. Oprescu, "Finding Significant Differences in the Energy Consumption when Comparing Programming Languages and Programs," *2022 International Conference on ICT for Sustainability (ICT4S)*, Plovdiv, Bulgaria, 2022, pp. 1–12.

[7] D. Nikolaeva and D. Petrova, "A Survey of the Programming Paradigms used in Programming Languages," *2024 5th International Conference on Communications, Information, Electronic and Energy Systems (CIEES)*, Veliko Tarnovo, Bulgaria, 2024, pp. 1–5.

[8] S. Maleki, C. Fu, A. Banotra, and Z. Zong, "Understanding the impact of object oriented programming and design patterns on energy efficiency," *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, Orlando, FL, USA, 2017, pp. 1–6.

[9] G. G. Magalhães, A. L. Sartor, A. F. Lorenzon, P. O. A. Navaux, and A. C. Schneider Beck, "How Programming Languages and Paradigms Affect Performance and Energy in Multithreaded Applications," *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, João Pessoa, Brazil, 2016, pp. 71–78.

[10] O. Ezenwoye, "What Language? - The Choice of an Introductory Programming Language," *2018 IEEE Frontiers in Education Conference (FIE)*, San Jose, CA, USA, 2018, pp. 1–8.

[11] M. Burger, G. N. Nguyen, and C. Bischof, "Developing Models for the Runtime of Programs With Exponential Runtime Behavior," *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, GA, USA, 2020, pp. 109–125.

[12] X. Cao and X. Zhang, "A cache power model and a runtime power simulation platform," *IET*

International Conference on Information and Communications Technologies (IETICT 2013), Beijing, China, 2013, pp. 100–105.

[13] Singh, S. (2023, September 7). *Compilation process in C#*. Scaler Topics. https://www.scaler.com/topics/csharp/compilation-process-in-c-sharp/