uc3m | Universidad Carlos III de Madrid

Bachelor's degree in Computer Science and Engineering

Bachelor Thesis

"Evaluating performance and energy impact of programming languages"

Author

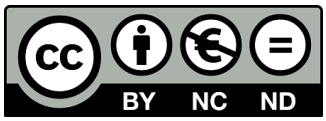
Eduardo Alarcón Navarro

Advisor

Jose Daniel García Sánchez

Leganés, Madrid, Spain

September 2025



This work is licensed under Creative Commons

Attribution - Non Commercial - Non Derivatives

To reach the moon, you should aim for the stars.

—

ACKNOWLEDGEMENTS

I would like to thank my family, for their continuous support and encouragement, specially this last year, where it has been the hardest for all of us. Not only they have provided me with the best education possible, financially and emotionally, but they have also taught me the importance of hard work and dedication.

I would like to thank my parents, my father for always being there when I needed him, for listening to me rant about many university projects, or crazy ideas. Furthermore, I would also like to thank my mother, for always being there for me, and for being the best mother I could have ever asked for, and for always supporting me in everything I do, even if we don't always agree on everything.

I would not have been able to finish my degree without the help of my friends, who have always been there for me, adopting me as part of their family and being there when I needed them most. During the four years of my degree, I have met many people, some have persevered in completing their degree, while others have chosen to take a different path in life. I would like to thank all of them for being there for me, and for being part of my life.

Another person I would like to thank is my thesis advisor Jose Daniel. Thank you for giving me the opportunity to work with you, after an extreme late request for a thesis. I am grateful for the trust you placed in me from the very beginning and for guiding me when I was most lost. From the admiration, it has been a great pride to close this stage with you as my advisor.

ABSTRACT

Nowadays, the importance of energy efficiency is increasing as more computationally expensive programs are being used by more and more people. The energy impact of running these programs is directly related with the programming language used to create the program as well as the design specifics.

This thesis aims to bring a specific example of the power efficiency of three programming languages: Python, Go and C++. Each one having its differences and properties, ease of use and execution speed. Each of these languages has been selected as each one has a particular characteristic that can be representative of their respective category of language.

Compiled languages with no garbage collection and no managed runtime have usually had the best execution speed as they can reach byte-code for each specific platform, but in the last years, other methods have improved significantly, such as JIT (Just in Time) compiling

To achieve realistic results, these languages were tested in multiple configurations, on different hardware, core count and operating systems to be able to eliminate any outliers.

Thus, this work will try showing the differences in energy consumption of different programming languages in a real world task, rendering a ray-traced image of multiple spheres with different materials and reflectivity.

Keywords: Compiled Language • Energy Efficiency • Interpreted Language • JIT • Ray-Tracing

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. Motivation

Energy consumption in the software industry has been raising over the years up to a point that it is now significant at a world energy consumption.

About 1.5% of the world's energy consumption is to be blamed on data centers and server farms. These numbers may not represent much, but from the total 183,230 TWh produced in 2023 [1] only 23,746 TWh come from a renewable source as it can be seen from Figure 1.1, which comes to 12.96%.

Knowing which language to use for each project is decisive not only in regards to the expertise one or their team might have on that language, but also the performance and language characteristics. If you want to develop a high-performance stock trader you would never think about using a high level language such as python or Perl, but you would try sticking to compiled languages such as Java C, C++, Java or Rust.

Thus, the main motivation for this project lies in studying 3 different programming languages, with each one having peculiar characteristics, to test their respective speed and power consumption in different platforms and architectures.

This comes from the idea that the program efficiency does not come from the language itself, but the implementation of the algorithm that the programmer chooses. The language helps, but choosing the optimal algorithm is much more important.

My personal take in this project comes from my hesitation in choosing a topic to specialize in inside the Computer Science area. Having seen and used many of these languages in multiple courses along these 4 years has made me realize the importance of choosing the correct language for each problem.

2023
in terawatt-hours

Other renewables	2,428 TWh
Modern biofuels	1,318 TWh
Solar	4,264 TWh
Wind	6,040 TWh
Hydropower	11,014 TWh
Nuclear	6,824 TWh
Natural gas	40,102 TWh
Oil	54,564 TWh
Coal	45,565 TWh
Traditional biomass	11,111 TWh
Total	**183,230 TWh**

Fig. 1.1. Global electricity generation by source in 2023

1.2. Objectives

The main goal of this project is the study and analysis of three implementations of a ray-tracer program, measuring the energy consumption as well as the time each program takes to complete. It should be also noted that the platform in which the program is being run affects the energy consumption of the program.

To perform this, I have improved the code from a well-known book called Ray Tracing in One Weekend [2], translating it to go and python, updating the code so that it could handle parallel rendering.

Once the code is created, the methodology for testing the different codes need to also be created.

- x86 Intel Xeon Based
- ARM Apple Icestorm & Firestorm
- x86 Zen 2 AMD ????
- ARM Cortex-A76 CPU - Raspberry Pi 5

1.3. Document Structure

The document contains the following chapters:

- Chapter 1, *Introduction*, details the motivation of the project.

- Chapter 2, *State of the Art*, describes the main points of interest in order to fully understand the project. Theoretical and technological issues are addressed.
- Chapter 3, *Problem Statement*, general description of the project and its requirements.
- Chapter 4, *Design and Implementation*, describes the most relevant design decisions with the multi-language renderers and their multi-threaded implementation.
- Chapter 5, *Evaluation*, the analysis and benchmarks are performed and the results are exposed and discussed.
- Chapter 7, *Socioeconomic environment*, provides a comprehensive account of the project's developmental costs and its associated socio-economic implications.
- Chapter 6, *Planification*, describes the organization of the project along the development.
- Chapter 8, *Regulatory Framework*, indicates the licenses under which the project is distributed.
- Chapter 9, *Conclusions and Future Work*, briefly analyzes the results obtained and states the possible future objectives of the project.

CHAPTER 2

STATE OF THE ART

This chapter describes the paradigms and characteristics of different programming languages. Thus concepts of compiled languages, interpreters optimizations and parallelism are discussed with respect of the different programming languages.

The purpose is to provide background information necessary to understand the study and present a clear justification for the decisions made

2.1. Energy Efficient Systems

An energy efficient system is defined as a system designed and optimized to performs its functions while consuming the minimum amount of energy possible, without compromising its performance, safety and reliability.

As Muralidhar, Borovica-Gajic, and Buyya [3] put it, the average power a system draws is:

$$P_{avg} = P_{dynamic} + P_{leakage}$$

The dynamic power depends on the V supply, the clock frequency, the node capacitance and the switching activity. This power can be reduced by reducing the load on the chip or by manually setting a limit on how much voltage the chip can draw (known as undervolting [4])

2.2. System Architectures

While the energy efficiency of a system is significantly affected by connected devices (e.g., a graphics card or an AI Accelerator), this study excludes any external devices and expansion cards. Therefore, the processor architecture is the primary factor determining

energy consumption on the system.

2.2.1. x86 Architecture

The x86 architecture is the most widely adopted in the world of desktop and server computers, whose market share is almost entirely shared by AMD and Intel who created it in 1978. Originally called x86-16, due to the 16 bit word size, it debuted in the Intel 8086 a single core, a $3\mu m$ node processor.

Nowadays, the technology has improved, the architecture is now called x86-64, a 64 bit extension, created by AMD, and releases the full specification in August 2000. From 2006 onward, the two companies have been developing multi-core processors, adding further Single Instruction Multiple Data (SIMD) Extensions such as AVX-512 [5]. Then came the integrated graphics and finally Power Efficiency Focus.

Then, a new paradigm came, instead of having a homogeneous set of cores, cores focused on performance and efficiency were added to the same package, the big.LITTLE architecture. This set of heterogeneous cores meant the scheduler had to be changed in the operating systems, to better allocate more demanding programs on high performing cores and lower important tasks, such as background jobs to the highly efficient cores. This technology was released by Intel on the 12th generation Intel core processors, using Intel 7 (a $7nm$ node). This approach was revolutionary for power efficiency as Padoin, Lima Pilla, Castro, *et al.* [6] state.

2.2.2. ARM Architecture

The ARM (Advanced RISC Machine) is the newest architecture that has reached the global scale. Developed in 1986, the goal of this new 32 bit architecture was the simplicity. As Moir [7] puts it, the energy efficient came later. This allowed the ARM architecture to dominate on the mobile sector, specially on smartphones, which run on batteries.

The characteristics of this ISA are a reduced set of instructions (RISC), which allows processors to have fewer transistors than CISC architectures such as x86, resulting in lower cost, lower temperatures and lower power consumption.

Currently, this technology is not only used in low-power light devices, but many laptops, and even desktops are using ARM chips due to their power efficiency and performance.

2.3. Programming Languages

In this section, the different programming languages that have been chosen will be discussed, as well as why other similar languages were not.

2.3.1. Go: Compiled Language with an Embedded Managed Runtime

Go is a language developed by Google, released in 2009, focused on concurrency. It has a runtime which manages the goroutines. As described by [8], GO has a Garbage Collector, which means while the program is running, there needs to be a thread checking for unused memory structures.

From Table 2.1, Go is statically typed and compiled, which makes it have a good start as an efficient programming language. But from the Table 2.2, we can observe go has a managed runtime, which means the energy consumption will be higher than other languages that do not have this. This runtime is the section of the program in charge of running and scheduling goroutines. This is why go binaries have a bigger minimum size as the runtime has to be fitted in the binary, which is great for cross compilation, but not great for either energy efficiency or performance.

This language was designed for backend tasks, handling thousands of simultaneous connections, while having an easy syntax for any programmer. Some companies that use this are Uber, Docker, Twitch and, although not mainly, Netflix.

2.3.2. Python: Interpreted Programming Language

Interpreters

TO DO: Execution Strategy: JIT compilers (e.g., PyPy) can optimize hot code paths to machine code at runtime, improving performance but with potential warmup costs.

TO DO: Typing & Compiler Optimizations: can apply runtime type-based optimizations. Type hints do not directly provide runtime performance benefits in CPython.

2.3.3. C++: Directly Compiled, Unmanaged Language

C++ is one of the most famous language when it comes to high performance computing applications. Based on the programming language C, released in 1978 as a high-level language at the time, compared to assembly.

As we can see from Table 2.5, there are many characteristics on why the language is one of the most used for high performance software, for example, blender or nuke. This known examples and the multiple tests performed in multiple courses during the computer science degree.

If we take into account the energy efficiency, from Table 2.6 we can observe that being a compiled language, with multiple optimizations at the compilation level, zero-cost abstractions, no runtime and direct memory management makes it one of the best low energy consumption language in theory. In this section, the different programming languages that have been chosen will be discussed, as well as why other similar languages were not.

Compilers

2.3.4. Other languages not used

2.4. Previous Benchmarks

TABLE 2.1
GO LANGUAGE GENERAL CHARACTERISTICS

General Characteristics	
Characteristic	**Description**
Concurrency Support (Goroutines & Channels)	Lightweight, concurrent execution units (goroutines) and communication primitives (channels) managed by the Go runtime, enabling efficient parallel processing and simplified concurrent programming.
Statically Typed	Types are checked at compile-time, ensuring type safety and early error detection, contributing to robust applications.
Compiled Language	Code is compiled directly to native machine code, resulting in fast execution and typically producing a single, self-contained executable.
Comprehensive Standard Library	Provides a rich set of packages for common tasks like networking, I/O, string manipulation, JSON handling, and cryptography, reducing reliance on external libraries.
Garbage Collection	Automatic memory management simplifies development by handling memory allocation and deallocation, helping prevent common memory-related bugs like leaks and dangling pointers.
Fast Compilation Times	The Go compiler is designed for speed, significantly reducing development iteration cycles and build times, especially for large projects.

TABLE 2.2
GO LANGUAGE CHARACTERISTICS ON ENERGY EFFICIENCY

Characteristics Impacting Performance & Energy Efficiency	
Characteristic	**Description (Impact on Performance/Energy)**
Efficient Concurrency (Goroutines & Channels)	Lightweight goroutines (often KBs in stack size) and efficient channel operations allow for high levels of concurrency with low overhead.
Managed runtime	Go has a managed runtime which manages memory deallocation and goroutines.
Compilation to Native Code	Produces efficient machine code directly executable by the CPU, performing some optimizations.
Optimized Standard Library	Many standard library functions, especially in areas like networking and cryptography, are highly optimized, often using assembly for critical sections.
Static Typing & Compiler Optimizations	Type safety allows for certain compiler optimizations like devirtualization and inlining.
Value Types and Stack Allocation	Go prefers value types and encourages stack allocation where possible, reducing heap allocations and GC pressure. *Stack operations are faster and more energy-efficient than heap operations.*

TABLE 2.3
PYTHON LANGUAGE GENERAL CHARACTERISTICS

General Characteristics	
Characteristic	**Python Description**
Bad Concurrency Support	Offers threading (often limited by the Global Interpreter Lock (GIL) for CPU-bound tasks in CPython), multiprocessing library (for parallelism)
Typing System	Dynamically Typed: Type checking occurs at runtime. Optional static type hinting (PEP 484) for static analysis.
Language Nature	Interpreted Language: Typically compiled to bytecode executed by a Virtual Machine (VM).
Standard Library	Extensive standard library: Offers a vast array of modules for diverse tasks, speeding up development.
Garbage Collection	Automatic memory management via reference counting and a cyclic garbage collector.
Execution Model	Interpretation: Code is interpreted, generally offering platform independence and ease of use.

TABLE 2.4
PYTHON LANGUAGE CHARACTERISTICS IMPACTING PERFORMANCE & ENERGY EFFICIENCY

Characteristics Impacting Performance & Energy Efficiency	
Characteristic	**Python Description (Impact on Performance/Energy)**
Concurrency Model Efficiency	Threading (GIL in CPython limits CPU-bound parallelism), Multiprocessing (bypasses the GIL, but has a higher overhead). Overall concurrency overhead is extremely high for CPU-bound tasks.
Managed Runtime	The Python interpreter manages memory and execution. Interpreter overhead and dynamic nature impacts the raw performance and energy use substantially.
Execution Strategy	Bytecode Interpretation: Code is compiled to bytecode and run on a VM.
Optimized Standard Library	Performance-critical modules often implemented in C. Python call overhead still exists.
Typing	Dynamic typing limits static optimizations.
Memory Allocation Strategy	Object Model and Heap Allocation: Predominantly uses heap-allocated objects and references, potentially increasing Garbage Collector (GC) load, memory footprint, and access latency compared to stack-based value types.

TABLE 2.5
C++ LANGUAGE GENERAL CHARACTERISTICS

General Characteristics	
Characteristic	**Description**
Object-Oriented Programming (OOP)	Supports encapsulation, inheritance, and polymorphism, enabling modular and reusable code.
Statically Typed	Types are checked at compile-time, catching errors early and aiding optimization.
Compiled Language	Code is compiled directly to native machine code for fast execution.
Portability	Well-written code can be compiled and run on various platforms.
Rich Standard Library	Provides optimized data structures and algorithms (e.g., vectors, maps, sort, templates).
Support for Multiple Paradigms	Supports procedural, object-oriented, and generic programming.

TABLE 2.6
C++ LANGUAGE CHARACTERISTICS ON ENERGY EFFICIENCY

Characteristics Impacting Performance & Energy Efficiency

Characteristic	**Description (Impact on Performance/Energy)**
Direct Memory Management	Allows fine-tuned memory accesing and managing (e.g., ‘new‘/‘delete‘), avoiding garbage collection. *Leads to fewer CPU cycles for memory tasks, saving energy.*
Low-Level Capabilities	Pointers and bitwise operations enable direct hardware interaction.
Compilation to Native Code & Optimizing Compilers	Mature compilers generate highly optimized machine code. *Faster execution means shorter active CPU time, lowering energy use.*
Zero-Cost Abstractions	High-level features (classes, templates) aim to have no runtime overhead if unused or compiled efficiently.
Templates and Generic Programming	Code specialized at compile-time for specific types, avoiding runtime overhead.
RAII (Resource Acquisition Is Initialization)	Deterministic resource management prevents leaks and improves stability.

CHAPTER 3

PROBLEM STATEMENT

3.1. Project Description

3.2. Requirements

3.2.1. Functional Requirements

3.2.2. Non Functional Requirements

3.3. Use Case

3.4. Traceability

CHAPTER 4

DESIGN AND IMPLEMENTATION

4.1. General Program Design

4.2. Scene

4.2.1. Sphere_data design

4.2.2. Language Specific

C++

Go

Python

4.3. Object

C++

Go

Python

4.4. Renderer

4.4.1. Camera

C++

Go

Python

4.4.2. Multiprocessing

C++

Go

Python

4.5. Output

CHAPTER 5

EVALUATION

5.1. Measurement Platforms

5.1.1. Many Core Platform

Evaluated parameters

Results

5.1.2. Personal Desktop

Evaluated parameters

Results

5.1.3. Personal SOTA Laptop

Evaluated parameters

Results

5.1.4. Raspberry Pi 5

Evaluated parameters

Results

CHAPTER 6

PLANIFICATION

CHAPTER 7

SOCIOECONOMIC ENVIRONMENT

7.1. Budget

7.1.1. Human Resources

7.1.2. Material Resources

Hardware

7.1.3. Software

7.1.4. Indirect Costs

7.1.5. Total Cost

7.2. Socio-Economic Impact

CHAPTER 8

REGULATORY FRAMEWORK

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

Example of figure:

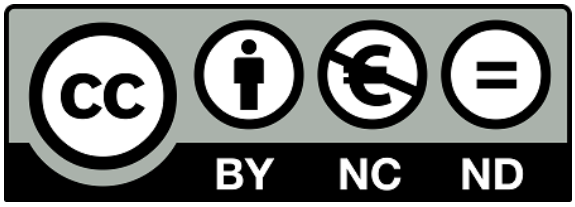


Fig. 9.1. Figure name here

Example of table:

TABLE 9.1
LOREM IPSUM

I		II		III			IV
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Source: BOE

BIBLIOGRAPHY

[1] P. R. Hannah Ritchie and M. Roser. "Energy Production and Consumption." (Jan. 2024), [Online]. Available: https://ourworldindata.org/energy-production-consumption.

[2] S. H. Peter Shirley Trevor David Black. "Ray tracing in one weekend." (Apr. 2025), [Online]. Available: https://raytracing.github.io/books/RayTracingInOneWeekend.html.

[3] R. Muralidhar, R. Borovica-Gajic, and R. Buyya, "Energy efficient computing systems: Architectures, abstractions and modeling to techniques and standards," *CoRR*, vol. abs/2007.09976, 2020. arXiv: 2007.09976. [Online]. Available: https://arxiv.org/abs/2007.09976.

[4] A. Muc, T. Muchowski, M. Kluczyk2, and A. Szeleziński, "Analysis of the use of undervolting to reduce electricity consumption and environmental impact of computers," *Rocznik Ochrona Środowiska*, 2020.

[5] Intel. "Maximize roi and performance for demanding workloads with intel avx-512." (Jun. 2024), [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-avx-512.html.

[6] E. Padoin, L. Lima Pilla, M. Castro, F. Zanon Boito, P. Navaux, and J.-F. Méhaut, "Performance/energy trade-off in scientific computing: The case of arm big.little and intel sandy bridge," *IET Computers & Digital Techniques*, vol. 9, pp. 27–35, Dec. 2014. DOI: 10.1049/iet-cdt.2014.0074.

[7] C. Moir. "The remarkable story of arm." (Jun. 2020), [Online]. Available: https://medium.com/swlh/the-remarkable-story-of-arm-85760399c38d.

[8] M. Rosecrance. "The garbage collector." (Feb. 11, 2019), [Online]. Available: https://www.youtube.com/watch?v=gPxFOMuhnUU (visited on 06/05/2025).

GLOSSARY

A

AI Accelerator

An expansion card that usually has specific hardware to accelerate AI workloads. 5

C

CPython

Pythons default interpreter, with a GIL, until version 3.13 which an experimental free-threaded version was created and version 3.14 will continue its development (both known as 3.13t and 3.14t respectively). 11, 12, 32

cross compilation

Compiling a program with a different architecture than the host machine that is making the compilation.. 7

G

GC

The garbage collector is an automatic memory management whose primary job is to reclaim memory that was allocated by the program but is no longer being used. 12

GIL

The global interpreter lock is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once. 11

goroutines

A goroutine is a lightweight thread managed by the Go runtime. 7, 10

H

Heap Allocation

Heap allocation is the process of reserving a block of memory from a large, flexible pool (the heap) for program data whose size or lifetime is not known at compile time, allowing for dynamic memory management during program execution. As python does not know the sizes of most variables, it needs to perform heap allocations always.. 12

O

OOP

Programming paradigm that organizes code into reusable units called "objects," which contain both data and the functions that operate on that data. 13

P

polymorphism

allows objects of different types to be treated as objects of a common superclass or interface. 13

V

VM

The Python virtual machine is the same as the interpreter, which turns python's code into machine ready instructions. Pythons default VM implementation if CPython, but others exists such as Jython (Java), IronPython (.NET) or PyPy. 11

ACRONYMS

G

GC

Garbage Collector. 12, *Glossary:* GC

GIL

Global Interpreter Lock. 11, 12, 31, *Glossary:* GIL

J

JIT

Just In Time Compiler. 7

O

OOP

Object-Oriented Programming. 13, *Glossary:* OOP

S

SIMD

Single Instruction Multiple Data. 6

V

VM

Virtual Machine. 11, 12, *Glossary:* VM