Performance evaluation of concurrent languages: C++, Go & Rust

Florentin Bekier
Department of Computer Science
Illinois Institute of Technology
A20458632

Rémi Blaise
Department of Computer Science
Illinois Institute of Technology
A20453722

Neeraj Rajesh
(PhD candidate supervisor)
Department of Computer Science
Illinois Institute of Technology

***Abstract*—This project aims to evaluate to what extent Go and Rust can be serious replacement of C++ and how well are they performing, through a benchmark analysis of usual operations as well as more specific tasks that are important to high-performance development. This study is intended to provide a reference comparison between C++, Go, and Rust.**

***Index Terms*—performance, benchmark, programming, c++, go, rust**

I. Introduction and motivations

The C language has been created in 1972 and has ruled the field of system programming for almost 50 years now. It is indeed the primary system programming language used in Windows, Linux, and macOS systems. The language brings simple and common abstractions over the machine instruction set and gives full power over the machine to the programmer. But this power comes with a major flaw: the ability to write unsafe code and to lead to undefined behavior. It is the issue of memory-safety. In consequence, operating systems with such a large code-base can easily contain memory leaks and breaches leading to regular discoveries of new exploits.

Its successor, C++, was released in 1985 and brings a lot of commodity features to ease programming such as object-oriented programming. [1] A huge ecosystem of libraries has been built for C++ developers over the years and it is regularly updated with new useful features, the latest stable version was released in 2017. However, this language still suffers from the weaknesses of C and its old design.

On the other hand, high-level programming languages such as Java or Python are great for programmers because they automatically manage memory and they come with a lot of libraries implementing useful data structures and algorithms. However, using high-level languages, that is, languages with strong abstraction from the computer architecture, often result in lower performances than an optimized C or C++ code.

Recently, new languages such as Go or Rust were created to resolve issues related to C and C++ without giving up on low-level programming and performances. They also claim to ease the creation of highly concurrent programs by providing proper mechanisms. [2] [3] However, many programmers are timorous to use these new languages because they are familiar with C++ and are not convinced to obtain the same performances.

The goal of this project is to perform a benchmark analysis of the three languages C++, Go and Rust to compare their performances and features. By comparing them on the same aspects, we hope to provide an answer regarding the performances of Go and Rust against C++ and therefore evaluate to what extent they can be a serious replacement to the latter.

II. Project contributions

This project was proposed by SCS Lab to CS550 students as a final project. It was supervised by Neeraj Rajesh with whom we held weekly meetings to review the work achieved and plan the next steps together. Some of the work was done collaboratively during these meetings, such as the establishment of the benchmark list. The implementation of the benchmarks in C++ and Go was carried out by Florentin Bekier. The implementation of the benchmarks in Rust, as well as the script to run and evaluate them, was carried out by Rémi Blaise.

III. Project description

The languages versions evaluated are C++11 (with GCC 8.4), Go 1.14, and Rust 1.41.1. The measurements were performed on a computer with an Intel Core i5-9300H (9th gen, 4 cores, 2.40 GHz) processor and 8GB of RAM.

This project was conducted in five stages. The first step was to establish a set of benchmarks that were going to be evaluated. The purpose of this study is to evaluate the languages on their overall performance, and especially for high-performance programming so we based the benchmarks on the most common features used by programmers. The set of benchmarks shall reflect a basic usage of the programming languages (no specific tasks), only contain tasks that are comparable between the three languages, and be representative of the current programming methods. Therefore, we have also taken into account any specificity of the languages to rely on standard libraries as much as possible. The final list was discussed and validated collaboratively. In total, we decided to benchmark 12 categories of tasks, each divided in 1 to 6 test case consisting of an individual program. In each program, we repeat the operation N times and measure the metrics for the overall run. This allows us to compute an average per repetition. This value N is determined for each program to have an execution time of about one second. The value can be specified as an argument of the program. For some programs,

we can also pass as an argument the size of the data structure that will be used (array, vector, etc.). Here is the benchmark list:

- Hello World (basic program)
- Memory management (allocation, write, read and delete)
- Iteration / recursion
- Data structures operations (array, vector and b-tree)
- Matrix operations (multiplication)
- Concurrency mechanisms (thread, process and mutual exclusion)
- Strings
- Data serialization (binary, XML and JSON)
- Hashing (hash maps and cryptographic hash functions)
- Sockets (TCP)
- Timers precision and time read
- Computation: complex numbers

The second step of the project was to select the metrics that are going to be measured and analyzed to compare the languages. We agreed on four basic metrics: binary size, execution time, memory usage, and CPU usage. Those metrics are the most interesting for the majority of programmers because they indicate the overall performance of a program. The binary size is interesting when programming for devices with limited storage such as mobile or embedded devices. The execution time is the main performance indicator of a program and often has to be minimized. The memory usage and CPU usage are other indicators of performance that always need to be minimized, for limited resources devices but also computers, servers, and high-performance machines. We also measured the compilation time.

After establishing those prerequisites, we proceeded with the implementation of the test cases for each language. This step required extra care to make sure that every test case was almost identical in each language and that every language had the same test cases. As a result, we obtained 40 tests (programs) for C++, 38 tests for Go, and 39 tests for Rust. The difference between the number of tests can be explained because C++ dynamic arrays and vectors are different whereas they are the same data structure in Go and Rust. Moreover, we added some tests in Go and Rust when multiple options were available to do the same thing (e.g. hash maps in Rust with different hashing functions).

With all the tests ready, we compiled them with the standard level of optimization for production of each language (O2 or equivalent) and executed them all in the same environment while measuring the metrics via an execution script. Lastly, we gathered the results together to provide a relevant analysis.

IV. Results

This section presents the results we found by executing our benchmarks, summarized in tables and charts, and provide an analysis of those results. The complete data can be found with the source code.

A. Compilation time

By measuring the total compilation time of the project for each language, we can compute the average compilation time of the scripts. The compilation time is not the most important metric to consider when it comes to choosing a programming language, but it is your first contact with the language and it impacts the swiftness of the development of large projects.

TABLE I
TOTAL AND AVERAGE COMPILATION TIME

Language	Total time (s)	Number of scripts	Average time (s)
C++	13.75	40	0.34
Go	7.43	38	0.19
Rust (first time)	53.12	39	1.36
Rust (next times)	1.11		0.03

In this domain, both Go and Rust compilers are performing better than the GCC compiler. Go shows a reduction of 45% in compilation time compared to C++ while Rust is the only language to cache intermediate compilation results, making it compile 3 times slower on the first time but 10 times faster than C++ on subsequent times.

B. Binary size

Binary size can be a crucial metric for software on embedded systems with limited storage available. This is why we are reporting the average binary size obtained on the project.

TABLE II
AVERAGE BINARY SIZE

Language	Binary size
C++	17.95
Go	
Rust	2682.44

We can see here a significant difference between the languages. While Rust binaries are on average slightly bigger than Go, both Go and Rust show considerably heavier binary sizes by more than 100 times on average. C++ remains unbeaten in terms of binary sizes.

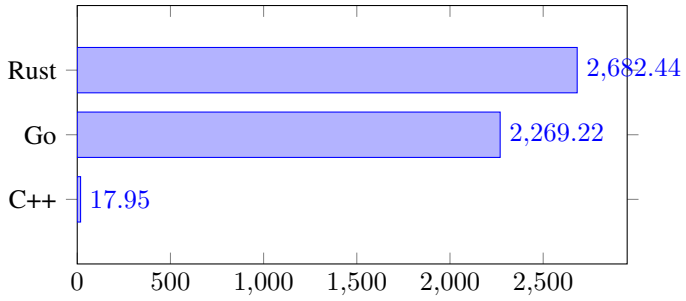

Fig. 1. Average binary size

C. Execution time

For each test, we measured the total execution time of the program as well as the time of the critical section of the code that runs the test, excluding the initialization time. We then calculated the time difference with C++ for Go and Rust.

TABLE III
EXECUTION TIME

Language	Average	Median	Best	Worst	SD
Go	356.77%	117.82%	−81.23%	4702.51%	8.63
Rust	−18.16%	−20.83%	−100.00%	246.55%	0.74

This is maybe the most interesting result of this study. Most Go scripts appear to be several times slower than C++. On the contrary, most Rust scripts appear to be faster than C++. On average, Go is 4.5 times slower while Rust is 18% faster than C++.

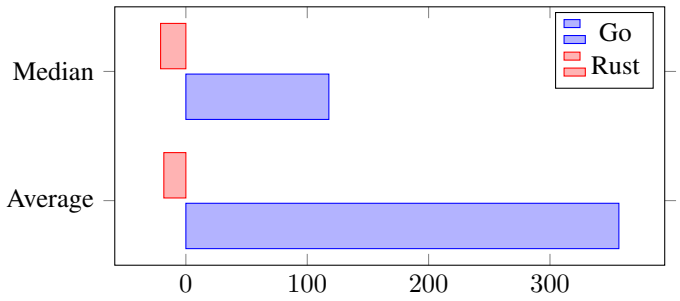

Fig. 2. Critical execution time difference with C++

D. When is Go good?

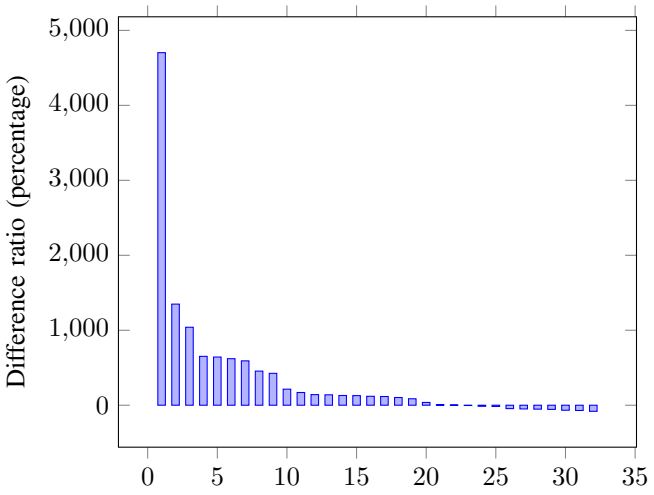

Fig. 3. Execution time difference of Go compared to C++

Go performed better than C++ on the following tasks:

- Hashing
- Iterative looping
- String splitting
- JSON serialization
- Concurrency (discussed further in a later section)

However, Go performed worst on:

- Complex and matrix operations
- Vector and B-Tree sorting
- Recursive looping
- Binary and XML serialization
- Socket ping-pong
- Timers

E. When is Rust good?

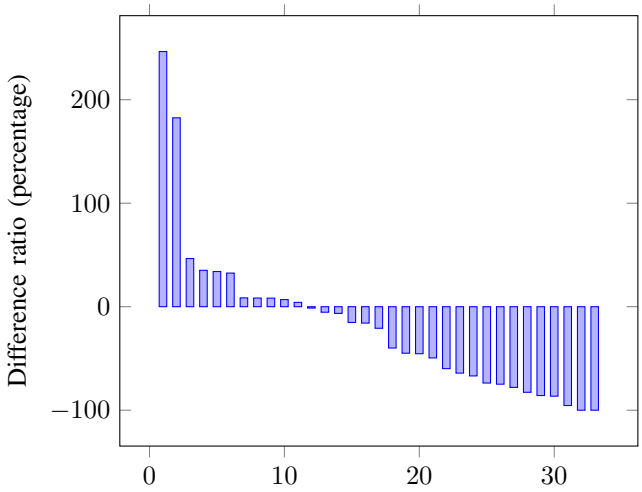

Fig. 4. Execution time difference of Rust compared to C++

Rust performed better than C++ on the following tasks:

- Complex number operations
- Vector sorting
- Hashing
- Iterative and recursive loops
- Binary and JSON serialization
- String serialization
- Timing

However, Rust performed worst on:

- B-Tree sorting
- Matrix operations
- XML serialization

F. Memory usage

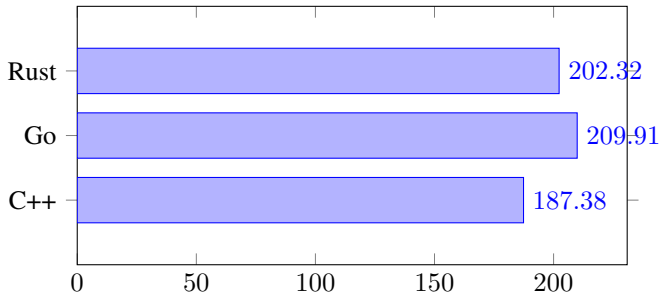

Fig. 5. Average memory usage

Our benchmarks don't focus on memory-intensive task so the results are merely indicative. However, we can notice that the best language in terms of average memory consumption is C++, while the most memory-consuming is Go. Again, C++ proves to be better for programming on limited resources devices.

G. About concurrency

Rust offers the same concurrency mechanism as C++: it promotes the usage of threads through the standard library and process forking is doable thanks to a library wrapper C code.

Go, however, introduces a new way of doing concurrency through goroutines. Goroutines are much more efficient than threads as they spawn quicker and have less memory overhead. In our benchmarks, we can spawn 2 million goroutines in the same time it takes to spawn 25,000 threads in C++ and Rust while also using much less memory. Moreover, the language smartly manages goroutines: it parallelizes goroutines if one is blocking the others, but otherwise groups goroutines in the same threads. This mechanism is meant to optimize the distribution of concurrent operations. [4]

Goroutines are particularly useful in the use case of web services: indeed, web services have this particular need of having to answer the most possible concurrent requests while requiring an insignificant processing time compared to waiting for IO operations (with filesystems, databases, other services, etc.)

Go doesn't provide out-of-the-box threading and forking mechanisms. Both languages offer efficient mutex mechanisms.

H. About programming ease and learning curve

Apart from performances, the main goal of these new languages is the pledge to bring programming benefits:

- Go is quicker and easier to handle than C++. It brings the ease of development of modern higher-level languages into a close-to-the-system programming language. However, as the benchmark results showed, it may not be as competitive as Rust in terms of performances.
- Rust has a steep learning curve and one main pledge: addressing the undefined behavior issue of C++, source of so many bugs in operating systems, and software, but keeping zero-overhead in the runtime. To do that, it designs a subtle and innovative memory-management mechanism. It also brings a much better developer experience than C++ with friendly erroring, a standard package manager, ... [5]

V. Discussion

This project was really interesting to realize. We learned a new way of working as it was for both of us the most rigorous research project we did. Our supervisor guided us thoroughly to perform the most useful, accurate, and scientific analysis. Having weekly meetings was a good way for us to work regularly and have constant feedback. Therefore, we think that we learned a lot of things for this project and carried it out well. There are, of course, a lot of things that can be improved, especially on the precision of our measures and the analysis. We measured the benchmarks on our personal computer whereas it might have been better to use a specific server but we didn't have access to any. Moreover, some of the values in our data seem to be wrong which may be due to our execution script. Lastly, we think that the analysis of the data may be pushed a bit further but we didn't have time to do so as the main part of our project was to write the benchmark tests.

VI. Conclusion

To conclude, based on our results we think that Rust is the best option to replace C++ as a high-performance language since the majority of the benchmarks were faster in Rust than in C++. However, C++ is faster for some tasks and it is also better on memory usage and much better on binary size. Therefore, C++ is still the best language for programming on resource-limited devices, such as embedded chips. Go, however, is definitively slower than C++ and Rust and should be avoided for programs requiring high performances. Nevertheless, Go and Rust have some major advantages compared to C++ that are memory safety and programming efficiency. It is indeed faster to code in Rust and Go than in C++, and especially Go. The programs also contains less bugs and they are easier to identify.

References

[1] B. Stroustrup, *A History of C++: 1979–1991*. New York, NY, USA: Association for Computing Machinery, 1996, p. 699–769. [Online]. Available: https://doi.org/10.1145/234286.1057836

[2] Go language FAQ. [Online]. Available: https://golang.org/doc/faq

[3] J. Kincaid, "Google's Go: A new programming language that's Python meets C++," *Techcrunch*, 2009. [Online]. Available: https://techcrunch.com/2009/11/10/google-go-language/

[4] Goroutines explained. [Online]. Available: https://yourbasic.org/golang/goroutines-explained/

[5] A. Lozovsky, "Rust vs C++ comparison," *Apriorit*, 2018. [Online]. Available: https://www.apriorit.com/dev-blog/520-rust-vs-c-comparison

[6] The computer language benchmarks game. [Online]. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/

[7] N. D. Matsakis and F. S. Klock, "The Rust language," in *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, ser. HILT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 103–104. [Online]. Available: https://doi.org/10.1145/2663171.2663188

[8] Y. Lin, S. M. Blackburn, A. L. Hosking, and M. Norrish, "Rust as a language for high performance GC implementation," in *Proceedings of the 2016 ACM SIGPLAN International Symposium on Memory Management*, ser. ISMM 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 89–98. [Online]. Available: https://doi.org/10.1145/2926697.2926707