CS 131: HW 6 Report

Fulati Aizihaer
University of California, Los Angeles

1. Introduction

Companies like Dudelsack often deal with large sets of large log files. Although GNU grep works fine for these tasks, it is single-threaded, which means that it cannot utilize the multithreaded functionalities for parallel file search to make searches faster. This report evaluates rewriting the GNU grep to a parallelized version of grep in Rust (1.87.0) to get results faster and improve search performance.

2. Overview

Rust is an open-source systems programming language that was built to be fast, safe and great at handling multiple tasks at once. Rust was developed by Mozilla and it is used by hundreds of companies around the world to build everything from web servers to system tools. Rust is memory efficient with no runtime or garbage collectors, making it perform better and more reliably in many situations. It also avoids many common programming bugs, making it a good choice for reliable and secure development.

3. Performance

Rust is a very fast programming language that has a performance that is comparable with C/C++ in terms of overall speed and performance, because it compiles directly into machine code. In some cases, Rust performs better because it doesn't utilize garbage collection and developers can manage memory very efficiently. This makes it perfect for searching through large sets of log files quickly.

4. Reliability

Rust helps developers write safe code that doesn't crash. Unlike other languages that are prone to memory safety issues, Rust is designed with memory safety in mind. Through its ownership models, it prevents bugs like accessing memories that have already been freed. These features help catch mistakes early and make it more stable.

5. Concurrency

Rust provides safer concurrent primitives, making concurrent programming more reliable compared to most other languages. It allows you to write code that runs in parallel on multiple threads which is much quicker and more efficient, while making sure that you don't run into bugs. Through libraries like *rayon*, developers can easily write code that uses all CPU cores effectively, speeding up tasks like searching through files.

6. Security

Rust is designed to help write more secure code, as it protects against many issues that cause security bugs in other languages such as buffer overflow or memory leaks. Rust doesn't need a garbage collector and can catch bugs early by avoiding null pointers, dangling pointers, or data races all together with safe code. Therefore, it is great for writing secure and high performing tools.

7. Ease of use

Rust can be tricky to learn at first to figure out different built in systems, but once you understand the basics, it is very developer friendly. Compared to C or C++, Rust is easier to use and has a lower learning curve. Rust also has a strong community support, libraries, documentations, online resources where new users can take advantage of when they are first learning how to use the language.

8. Flexibility and Generality

Rust can be used for all sorts of projects. It is increasingly used for system programming, web assembly and embedded systems. If you already have some code written in C, Rust can work with it directly. This means that you

can switch to Rust slowly instead of rewriting everything at once. Additionally, Rust has tons of open-source libraries that cover most tasks that developers might need, such as *rayon* library for parallelism.

9. Rust Weakness

Rust is still a relatively new language compared to older languages like C and C++, so it doesn't have quite the same size of community or number of mature libraries. This means developers might need to build more tools from scratch, or spend more time finding the right libraries for their use case. C++ also has decades of tutorials, solutions, and example code readily available, which Rust is still working on.

10. GNU grep vs grep with Rust

GNU grep is a tool that searches one or more input files for lines that match a given pattern, then it outputs those matching lines. One major limitation of GNU grep is that it only runs on a single thread. Even if a computer has multiple cores, GNU grep searches through files one at a time, which slows things down significantly when dealing with a large number of files or really large files like those used at Dudelsack.

Compared to GNU grep which was written in C, there are many advantages in rebuilding grep in Rust. First, Rust's high performance and memory safety make it an ideal platform for processing large volumes of data quickly and securely. Libraries like *rayon* allow developers to efficiently search through many files at once in parallel using multithreading. Unlike many other languages including C, developers in Rust don't have to manually manage memory or worry about bugs like buffer overflow, because Rust automatically implements safe memory handling.

However, there are also some challenges when it comes to developing grep with Rust. One challenge is working with extremely large files, where some lines are up to a gigabyte in size. To handle this well, the program needs to be able to break the files into smaller pieces and manage memory carefully. A library like *tokio* can help by reading files in the background without blocking other tasks. It uses asynchronous I/O, which is more efficient when dealing with large amounts of data.

Another challenge is to keep the search results in the correct order when using multiple threads. Since each thread finishes at a different time, the program can utilize channels to correctly organize the outputs. GNU grep doesn't have this problem as it runs in order.

Finally, like many other languages, Rust has a learning curve. It may take more development time to become familiar with the language and to learn how to effectively use the tools and libraries it provides to solve specific problems.

Conclusion

While C gives developers full control, it doesn't come with any built-in safety guarantees, which increases the risk of crashes and security vulnerabilities. On other hand, Rust is a great option for rewriting grep with parallelize feature as it offers various performance, reliability, safety, and concurrency features, which makes it suitable to handle Dudelsack's large sets of large log files. Rust's ability to manage memory efficiently and thread safety allows it to run tasks in parallel with fewer bugs and more efficiently. Although there are some minor challenges, overall Rust is a viable long term solution for Dudelsack.

References

1. Erolin, J. (2021). Rust vs C++ performance: When speed matters. BairesDev. https://www.bairesdev.com/blog/when-speed-matters-comparing-rust-and-c/

2. Klabnik, S., & Nichols, C. (2018). The Rust programming language. No Starch Press. https://doc.rust-lang.org/book/

3. The Rust Project Developers. (2025). The Rust standard library. https://doc.rust-lang.org/std/

4. Tokio Project Developers. (2025). Tokio: An asynchronous Rust runtime. https://docs.rs/tokio/latest/tokio/

5. GNU Project. (2025). GNU grep: Print lines that match patterns. https://www.gnu.org/software/grep/