
**UNIVERSIDADE FEDERAL DE PELOTAS**
**CENTRO DE DESENVOLVIMENTO TECNOLÓGICO**
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**
**DISCIPLINA DE CONCEITO E LINGUAGENS DE PROGRAMAÇÃO**

**EDUARDO SCHWANTZ**
**PEDRO HENRIQUE LIMA MESQUITA**
**RAFAEL DE MATTIA**
**ULIAN GABRIEL ALFF RAMIRES**

# RELATÓRIO SOBRE A COMPARAÇÃO DA IMPLEMENTAÇÃO EM C DA ELIMINAÇÃO DE GAUSS NAS LINGUAGENS GOLANG E RUST

**PELOTAS, 28 DE ABRIL DE 2023**

- **Introdução**

O desenvolvimento de software é uma área em constante evolução, e frequentemente surgem novas linguagens de programação que prometem melhorias em relação às que já existem. Duas linguagens que ganharam destaque nos últimos anos são Rust e Golang. Ambas foram desenvolvidas pensando em performance, segurança e facilidade de uso, além de suportarem programação concorrente.

Para avaliar essas linguagens, este estudo comparou a implementação em C do algoritmo de Eliminação de Gauss com suas implementações em Rust e Golang. Serão analisadas as diferenças entre as linguagens no que tange tipos de dados, acesso às variáveis, organização de memória, chamadas de função, controle de fluxo, etc. Além disso, serão utilizadas métricas como número de linhas e número de comandos para comparar o código das diferentes implementações.

Outra análise que será realizada é a comparação do desempenho das linguagens, levando em consideração diferentes tamanhos de matrizes trabalhadas.

- **Metodologia**

Para realizar este trabalho, utilizamos a plataforma Repl.it, que nos permitiu criar um repositório com a capacidade de executar códigos em diversas linguagens. Além disso, a plataforma permite desenvolvermos os códigos em grupo em tempo real. A plataforma também é útil para medição de desempenho, já que o ambiente ali é muito mais controlado do que um ambiente de sistema pessoal. O algoritmo de eliminação de Gauss que desenvolvemos utilizou-se como base os seguintes vídeos:
<https://www.youtube.com/watch?v=SdJQeIwSJmY>.
<https://www.youtube.com/watch?v=2dE7mkPYfP0> (Univesp)
<https://www.youtube.com/watch?v=1pYSxyz7n9U&t>

A partir da implementação em C, o código foi reescrito em Rust e em Golang, seguindo as boas práticas e características próprias de cada linguagem. Durante a reescrita, foram realizadas comparações entre as linguagens em termos de tipo de dados, acesso às variáveis, organização de memória, entre outros aspectos relevantes.

Com base nisso, algumas métricas foram definidas para a realização de comparações entre os códigos das linguagens, como números de linhas e o tempo de

execução de cada algoritmo de Gauss para diferentes tamanhos de matrizes.

Ao final, uma análise geral foi apresentada, destacando as principais diferenças e semelhanças encontradas entre as linguagens em relação à implementação do algoritmo de eliminação de Gauss. Também foram apresentadas algumas sugestões de situações em que cada linguagem pode ser mais adequada, de acordo com as características que aprendemos durante o desenvolvimento do trabalho.

- **Golang**

Go, também conhecida como Golang, é uma linguagem de programação de código aberto criada em 2007 pela Google e lançada em 2009. Algumas das características mais notáveis de Go incluem sua sintaxe simples e concisa, que é fácil de ler e escrever; sua velocidade, que faz ela ser adequada para lidar com grandes quantidades de dados em tempo real; e sua segurança, que é garantida por um modelo de memória de segurança estática que ajuda a evitar problemas de segurança comuns, como vazamentos de memória. Go também possui um excelente sistema de gerenciamento de pacotes rico que permite aos desenvolvedores compartilhar e reutilizar código facilmente, bem como uma biblioteca padrão muito interessante que fornece muitas ferramentas úteis para lidar com as tarefas mais corriqueiras de programação.

- **Rust**

Rust é uma linguagem de programação de sistemas desenvolvida pela Mozilla, que enfatiza a segurança e a concorrência. Possui um sistema de tipos de propriedade que previne problemas de segurança e utiliza um compilador que realiza análises de segurança estáticas. Rust também tem um sistema de tipos  que garante que o código seja livre de problemas de memória, como ponteiros nulos ou referências inválidas. Além disso, utilizando um modelo baseado em atores, Rust tem a capacidade de gerenciar concorrência de forma segura, o que facilita a criação de programas concorrentes.

- **Diferenças gerais**

Rust usa “**;**” para separar comandos, enquanto Go não usa, portanto, de forma geral, isso pode reduzir algum tempo a procura de um ponto e vírgula esquecido pelo código em GO.

Quanto aos tipos de dados, as variáveis do código Go são inicializadas com um valor zero, enquanto as variáveis do código Rust não têm um valor padrão. Existem diferenças notáveis nos tipos de dados entre as duas linguagens.

Em Rust, o sistema de tipos é mais rigoroso e utiliza o conceito de propriedade de variáveis (quando a variável é passada para uma função, a “propriedade” é transferida para essa função. Isso evita problemas comuns de segurança como vazamentos de memória e acessos inválidos). Já em GO, o sistema de tipos é mais flexível, permitindo mais liberdade ao lidar com tipos de dados, mas a verificação de erros só ocorre em tempo de execução.

No que tange escopo e acesso a variáveis, Go e Rust permitem a definição de variáveis locais e globais, bem como estruturas de dados personalizadas (structs) e arrays. No entanto, Rust é mais rígido com relação ao escopo de variáveis, exigindo a especificação de propriedades de mutabilidade e posse para que variáveis sejam usadas em diferentes partes do código.

Quanto à organização de memória, Rust é mais rigoroso com relação à organização da memória, e precisamos especificar as propriedades de mutabilidade e posse das variáveis. Isso acaba tornando o código Rust mais seguro contra erros de memória, como vazamentos ou corrupções. Go é menos rigoroso, o que o torna mais fácil de programar, mas potencialmente menos seguro.

Quanto a implementação e chamadas de funções, as funções em Go são declaradas usando "func" e os argumentos são passados por valor ou referência, dependendo do tipo. Em Rust, as funções são declaradas usando "fn" e os argumentos são passados por valor ou referência, dependendo da propriedade de mutabilidade da variável.

- **A implementação em Go**

A linguagem GO surpreende primeiramente com a simplicidade na sintaxe. Ao desenvolver em GO, a experiência de programação foi confortável e de uma curva de aprendizado macia. A linguagem é muito simples e limpa. É uma linguagem compilada, o que possivelmente implica em uma velocidade maior de execução, que vamos checar nos testes a seguir, comparando com outras linguagens também compiladas (C e Rust)

Uma característica que chamou muita atenção e é muito interessante do GO é a atribuição múltipla, o que significa que é possível atribuir vários valores a várias variáveis em uma única linha. Isso é muito útil quando trabalhamos com funções que retornam vários valores. Abaixo está um exemplo usado na própria implementação da eliminação de Gauss. Em outra linguagem, seria preciso utilizar um temporário para essa operação.

```
func trocarLinhas(matriz [][]float64, linha_1 int, linha_2 int) {
    matriz[linha_1], matriz[linha_2] = matriz[linha_2], matriz[linha_1]
}
```

No final da implementação, em um primeiro momento confirmamos para nós mesmos que a linguagem GO realmente é pouquíssimo verbosa, são necessários comandos simples e intuitivos para programar. Não houve espanto ao “bater o olho” no código. Linguagens mais clássicas como C e Java costumam  falhar nesse aspecto.

Ficamos com vontade de experimentar a concorrência em GO, mas devido a falta de tempo em detrimento do semestre com poucas semanas, mantemos o código serializado, o que é uma pena dada ao suporte interessantíssimo de GO à paralelismo.

A comparação da experiência de programação com C nem faz muito sentido dada as grandes camadas de abstração e recursos úteis que a linguagem apresenta, diferente de C, uma linguagem de programação de sistemas muito mais enxuta e antiga.

- **A implementação em Rust**

Como não havíamos programado em Rust ou GO, não sabíamos o que esperar em nenhuma das linguagens. Entretanto, assim como GO nos surpreendeu positivamente, Rust nos trouxe uma tarefa intrigante. O sentimento de poder de expressão de C junto com um ar de altíssimo nível foi enriquecedor e quebrou expectativas.

Uma das principais diferenças que notamos ao começar a trabalhar com Rust foi o tratamento de memória. Enquanto em C, o programador é responsável por gerenciar a

memória manualmente, em Rust, isso é feito de forma mais automatizada através de um sistema de propriedade e referência. Essa abordagem permite que o compilador de Rust faça verificações de segurança em tempo de compilação, evitando muitos dos erros comuns de gerenciamento de memória que podem ocorrer em C.

Além disso, o sistema de propriedade e referência de Rust é altamente eficiente em termos de desempenho. Em C, a alocação e desalocação de memória podem ser caras e levar a vazamentos de memória. O sistema de propriedade e referência do Rust é projetado para minimizar a sobrecarga e garantir que a memória seja liberada assim que não for mais necessária.

Outra diferença perceptível que notamos entre Rust e C foi em termos de segurança. Enquanto em C, é possível ter acesso a memória inválida, como ler ou escrever em endereços de memória que não foram alocados ou já foram deslocados em algum momento, em Rust, isso é praticamente impossível. O compilador de Rust verifica a segurança do código em tempo de compilação e não permite que o programa acesse a memória inválida, evitando erros e vulnerabilidades de segurança comuns em C.

Rust nos pareceu possuir uma sintaxe um pouco mais complexa (ou talvez sofisticada) que C, porém demonstrou ser altamente expressiva e permite escrever código de forma mais concisa e legível. Além disso, a documentação e o suporte da comunidade de Rust são excelentes, tornando mais fácil para os desenvolvedores aprender a usar a linguagem.

No geral, ficamos satisfeitos com a experiência no decorrer do desenvolvimento do algoritmo. A linguagem é altamente segura e eficiente no que se trata de memória e desempenho, ao mesmo tempo que oferece um poder de expressividade muito parecido com a linguagem C, mas com um nível de clareza acima. Com toda a certeza, Rust se trata de uma linguagem muito moderna e devemos utilizar ela com mais frequência, dada sua capacidade.

- **Resultados**

Realizando uma análise de desempenho comparando C, Go e Rust, temos os seguintes dados:

| Ordem da Matriz | C | Rust | Golang |
|---|---|---|---|
| 4x5 | 0.000041 (TCPU) | 6.65µs | 1.04µs |
| 20x21 | 0.000165 (TCPU) | 467.69µs | 9.551µs |
| 100x101 | 0.001300 (TCPU) | 143.482729ms | 1.47453ms |
| 200x201 | 0.006169 (TCPU) | 1.133618554s | 9.431729ms |
| 300x301 | 0.019114 (TCPU) | 3.783182313s | 68.402269ms |

Apesar das diversas tentativas de medir um tempo considerável em C, a execução foi absurdamente rápida inclusive nas matrizes maiores, e utilizando diferentes métodos de medição de tempo. <https://stackoverflow.com/questions/13156031/measuring-time-in-c>

Quando medimos Walltime em C, o tempo foi sempre 0. Em Tempo de CPU, conseguimos alguns dados que comprovaram que o tempo de execução em matrizes maiores aumenta, mas esses dados não podem ser comparados com os medidos em Go e Rust, já que nesses casos eles medem Walltime e não tempo de CPU. Entretanto, preferimos deixar a referência do C como a mais performática e inserir os dados em Tempo de CPU assim mesmo na tabela.

Também é interessante analisarmos a robustez da ferramenta de medição de tempo do GO e RUST. Em algumas linguagens, a medição de desempenho e tempo pode ser uma grande dor de cabeça. As unidades de tempo de relógio na tabela foram convertidas automaticamente pela função de medição de tempo do GO, não nos dando praticamente trabalho nenhum no tratamento desses dados de tempo.

Não sabemos ao certo por que o tempo de execução foi tão expressivamente pior em RUST que em GO em todas as matrizes, pois segundo algumas pesquisas, Rust teria que ser mais performático que GO. Talvez isso se deva a características do ambiente Repl.it, dos compiladores disponibilizados no ambiente, ou até mesmo de alguma falha na nossa implementação de forma a melhor utilizar os recursos de Rust.

- **Conclusão**

Após realizarmos a comparação entre a implementação do algoritmo de eliminação de Gauss nas linguagens C, Rust e Golang, podemos concluir que cada uma delas possui suas particularidades e vantagens.

Em termos de segurança, Rust destaca-se por oferecer um compilador que impede muitos dos erros de programação mais comuns, como acesso a memória não alocada ou não inicializada. Já em termos de sintaxe, Golang apresenta uma sintaxe extremamente simples e de fácil aprendizado, o que acreditamos ser uma vantagem para programadores iniciantes.

Quando falamos de desempenho, tivemos algumas surpresas nos nossos testes. C, como esperado, foi incrivelmente rápida e serviu de referência, mas Rust teve um desempenho abaixo de GO. Não sabemos ao certo o que causou essa anomalia no desempenho de Rust, mas temos algumas ideias:

1) Ambiente de desenvolvimento repl.it, talvez o compilador presente no ambiente repl.it, ou a forma como a memória é disponibilizada para uso afeta de alguma forma a execução de código RUST.
2) Incompetência de nós programadores, que provavelmente deixamos alguma ferramenta de alto valor para desempenho para trás na hora de fazer a implementação em RUST, e tivemos maior facilidade em GO devido ao mais alto nível da linguagem.
3) Diferença na precisão da métrica do tempo das linguagens.

No entanto, quando comparamos só as duas linguagens de programação de sistemas (C e RUST), vale ressaltar que a linguagem Rust, mesmo um pouco mais lenta, a falta da necessidade de se importar com diversos detalhes no desenvolvimento foi uma experiência encantadora e bem chamativa quando comparada com C.

Concluímos mais uma vez que não existe linguagem ideal para todos os casos. A escolha da linguagem ideal dependerá das necessidades e características de cada projeto. Para aplicações que requerem alto desempenho e controle de memória, C ainda se destaca, mas Rust pode vir a se tornar uma opção muito viável e adequada. Já para projetos como aplicações de mais alto nível que demandam um balanço entre simplicidade, rápida prototipação, velocidade e curva de aprendizado mais amigável, Golang mostra-se uma linguagem muito interessante.