Implementação da Eliminação de Gauss

A eliminação gaussiana ou escalonamento, é um método para resolver sistemas lineares. Este método consiste em manipular o sistema através de determinadas operações elementares, transformando a matriz estendida do sistema em uma matriz triangular, também chamada de matriz escalonada do sistema. Transformado na matriz escalonada, a solução pode ser obtida via substituição regressiva. Naturalmente estas operações elementares devem preservar a solução do sistema e consistem em:

1. multiplicação de uma linha por uma constante não nula.
2. substituição de uma linha por ela mesma somada a um múltiplo de outra linha.
3. Permutação de duas linhas.

Em nosso trabalho, executamos uma implementação em linguagem C da eliminação Gaussiana, que está disponível em "https://github.com/gmendonca/gaussian-elimination-pthreads-openmp, e reimplementamos o código nas linguagens de programação Go e Rust com algumas alterações. Abaixo estão os resultados da nossa comparação, obtidas pela análise geral dos códigos, comparação das métricas utilizadas e também análise dos resultados obtidos.

Comparação entre as linguagens C, Rust e Go.

---

1. Tipos de Dados

● **C:** Usa int para inteiros e float para números de ponto flutuante. Como C é tipada estaticamente, é necessário declarar o tipo de cada variável antes de usá-la.
● **Rust:** Utiliza usize para índices e f64 para números de ponto flutuante. O Rust exige que os tipos sejam explícitos, além de delegar ao programador o número de bits para armazenar o dado. Realiza verificação estática rigorosa, também tipada estaticamente.
● **Go:** Usa int para inteiros e float64 para números de ponto flutuante. O Go tem inferência de tipos mais flexível, onde o compilador pode deduzir automaticamente o tipo de uma variável, mas ainda requer definições claras.

Diferença principal: Rust e C tem um sistema de tipos mais rigoroso, garantindo segurança, enquanto Go é mais flexível.

---

2. Acesso às Variáveis

● **Rust:** Tem como padrão o uso de variáveis imutáveis. Para uso de mutabilidade, declara variáveis mutáveis usando let mut (exemplo: let mut n = String::new();).

● **C**: A palavra chave const permite declarar variáveis imutáveis. Não utilizar essa palavra chave implica em a variável ser mutável. Assim, declara variáveis mutáveis de forma padrão (exemplo: int N;).

● **Go:** Utiliza abordagem bastante parecida com a de C. As variáveis são mutáveis por padrão, e imutáveis ao utilizar o const. Exemplo: var n int

Diferença principal: Rust exige especificar mutabilidade explicitamente (mut), enquanto Go e C assumem mutabilidade por padrão. Ou seja, por padrão as variáveis em Rust só podem ser inicializadas uma vez (a não ser que utilize o mut). A imutabilidade como padrão no Rust explicita um dos focos da linguagem, que é o uso de concorrência.

---

3. Organização de Memória

● **Go:** Possui um garbage collector para lidar com posições de memória em desuso. Permite alocar variáveis de forma dinâmica utilizando new e make, assim como em make([][]float64, n).

● **Rust:** Utiliza ownerships para gerenciar alocação e liberação de memória. Borrowing permite fornecer a posse (owner) de um valor para outra variável. Assim, gerencia a memória sem o uso de um Garbage Collector.

● **C**: Com a alocação única, os elementos da matriz são alocados em um único vetor, linearmente (exemplo: volatile float A[MAXN][MAXN]). Além disso, permite alocar espaço na memória utilizando o comando malloc, que deve ser liberado posteriormente com o comando free.

Diferença principal: Rust e C utilizam abordagens similares quanto à alocação de memória, permitindo que o programador tome decisões a respeito de o que deve ou

não se manter em memória. Já o Golang utiliza o GC para poder simplificar o código, aumentando o nível de abstração da implementação.

---

4. Chamadas de Função e Subprograma

Todos esses códigos usam main() como função principal, e possuem três funções auxiliares que inicializam as entradas, imprimem a matriz e realizam o cálculo da eliminação de gauss. As funções declaradas são as seguintes:

● **Rust:** fn initialize_inputs(n: usize, a: &mut Vec<Vec<f32>>, b: &mut Vec<f32>, x: &mut Vec<f32>) {}, fn print_matrix(n: usize, a: &Vec<Vec<f32>>, b: &Vec<f32>) {}, fn gauss(n: usize, a: &mut Vec<Vec<f32>>, b: &mut Vec<f32>, x: &mut Vec<f32>) {}.

● **Go:** func initializeInputs(){}, func printMatrix(){}, func gauss() {}, func printSolution() {}.

● **C:** void parameters(int argc, char **argv) {}, void initialize_inputs() {}, void print_inputs() {}, void gauss() {}.

Diferença principal: Todos os códigos apresentam subprogramas em sua estrutura. O código em Go possui uma sintaxe mais simples com funções sem parâmetros e retorno por valor, enquanto Rust tem verificações mais rígidas e passa parâmetros por valor (ou referência), enquanto C é menos rígido, passando parâmetros por valor (ou com ponteiro). O retorno da função nas três linguagens usadas na implementação do trabalho é por valor e também por referência.

---

5. Comandos de Controle de Fluxo e Iteração

● **Rust:** No código utilizado, usa por exemplo for row in (0..n).rev() {} e for col in row + 1...n {} para iterar. A linguagem possui suporte a switch-case, for, while e loop.

● **Go:** No código utilizado, usa for i := 0; i < n; i++ {}. A linguagem possui suporte a switch-case, for e go-to.

● **C:** No código utilizado, usa por exemplo for (row = N - 1; row >= 0; row–) {} para iterar. A linguagem possui suporte a switch-case, for, while, do-while e go-to.

Diferença principal: O for implementado em Go e C permite manipular a variável de iteração de forma manual (por exemplo, atribuir um novo valor à variável) em

meio ao código. Já em Rust, o for utiliza range para garantir que a variável percorrerá todo o intervalo definido na declaração do loop.

C possui suporte à maior quantidade de comandos de controle de fluxo. Golang permite simular o comportamento do do-while e do while utilizando a estrutura do for.

---

6. Tratamento de Saída

- **Rust**: Usa **`println!()`**, que é uma macro para imprimir texto no console, formatando automaticamente os valores.
- **Go**: Utiliza o pacote **`fmt`** para saída formatada, com **`fmt.Println()`** para impressão direta e **`fmt.Printf()`** para formatação específica.
- **C**: Usa a função **`printf()`**, que permite formatar e exibir textos e valores numéricos no console.

Comentário: O tratamento de saída nas três linguagens é semelhante, permitindo a impressão formatada de matrizes, vetores e tempos de execução, variando apenas na sintaxe e no método de formatação.

Métricas Comparativas

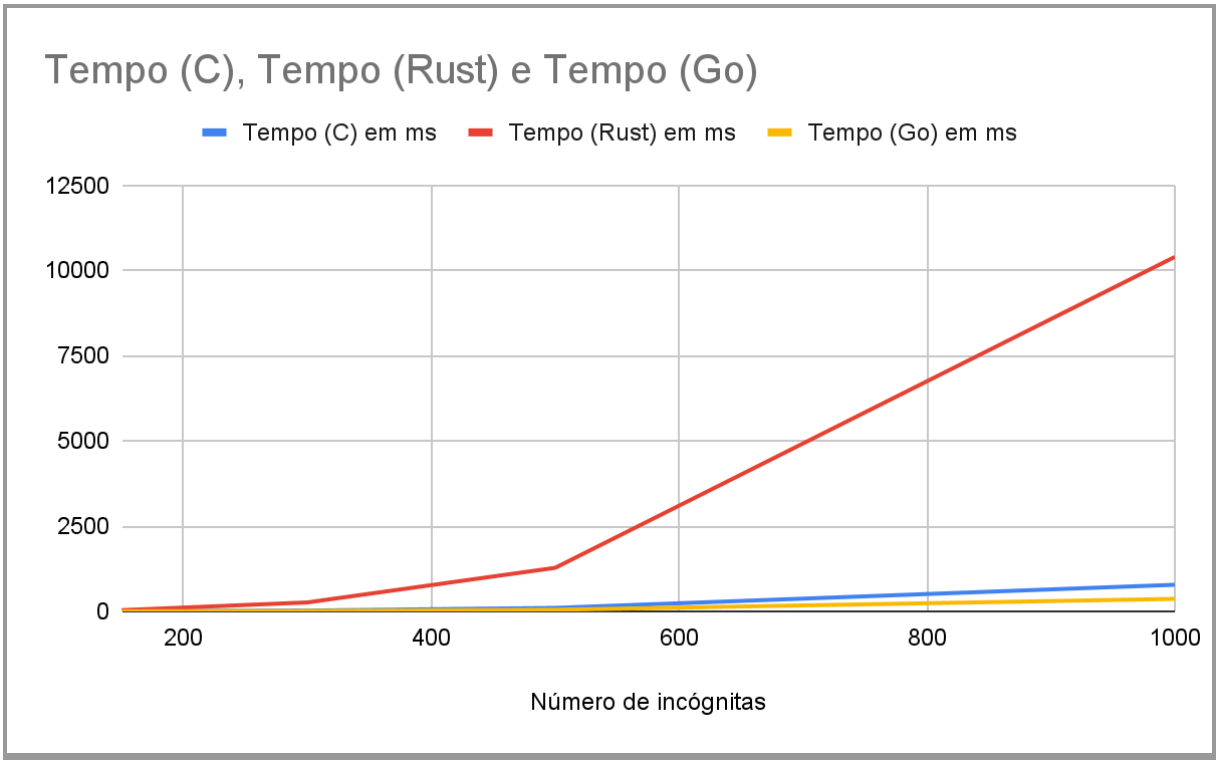
Métrica	C	Rust	Go
Número de linhas	168	92	96
Número de comandos	~75	~49	~44
Uso de if	5	4	4
Uso de for	14	11	8
Número de variáveis	19	12	13

declaradas

Número de caracteres no código	5114	3051	2539

Executou-se o código 5 vezes para cada tamanho de matriz. Registrou-se o tempo de execução em cada linguagem e depois foi feita uma média dos 5 valores.

Número de incógnitas	Tempo (C) em ms	Tempo (Rust) em ms	Tempo (Go) em ms
150	2,95	34,4	1,5
300	22,67	261,6	10,4
500	101,31	1276,6	45,4
1000	778,84	10397,2	365,2



Conclusão

A comparação entre C, Rust e Go na implementação da eliminação de Gauss mostra que cada linguagem tem vantagens distintas. Go apresentou o melhor

desempenho, sendo significativamente mais rápido do que C e Rust, o que pode ser atribuído à sua eficiente alocação de memória e otimizações internas, além do gerenciamento automático via garbage collector.

C, apesar de ser tradicionalmente eficiente, teve um desempenho inferior ao Go, possivelmente devido ao maior custo de manipulação manual de memória. Já Rust, embora garanta maior segurança e controle sobre a alocação, teve o pior tempo de execução, provavelmente devido às verificações rigorosas e ao modelo de propriedade, que pode introduzir overhead.

Características da máquina utilizada

Especificações do dispositivo

Nome do dispositivo	APNA
Processador	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
RAM instalada	8,00 GB (utilizável: 7,73 GB)
ID do dispositivo	9201C89F-146A-4374-9D09-4E0A10607653
ID do Produto	00342-42314-78995-AAOEM
Tipo de sistema	Sistema operacional de 64 bits, processador baseado em x64
Caneta e toque	Nenhuma entrada à caneta ou por toque disponível para este vídeo