Universidade Federal de Pelotas
Curso de Engenharia de Computação
Curso de Ciência de Computação
Disciplina: 22000188 - CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO
Turma: 2022/2 – T1
Professor: Gerson Geraldo Homrich Cavalheiro

Implementação da Eliminação de Gauss: Comparação entre RUST, GO e C.

Alunos: Bianca Nunes Coelho - 15102880, Raissa Nunes Coelho - 15102887

Data: 02/05/2023

1. Introdução

Este relatório visa apresentar a comparação entre as linguagens C, GO e RUST. Para a comparação foi utilizado o problema da Eliminação de Gauss em C e reimplementado em GO e RUST.

2. Comparação entre as linguagens

2.1 Códigos feito em GO - Comparação GO e C

GO é uma linguagem compilada, como a linguagem C. O acesso a variáveis em GO é determinado pelo escopo em que se encontram, parecido com a linguagem C, que pode ser: nível de pacote (package), nível de bloco (dentro de if ou outras estruturas de dados) e nível de função. A capitalização da primeira letra, seja no nome de uma função ou variável, determina se é pública ou privada. Maiúscula: publica (até mesmo para outros pacotes) e maiúscula: privada. O tipo dos dados também pode determinar o acesso às variáveis, já que alguns tipos, como slices e maps, possuem funções embutidas.

GO é uma linguagem de programação tipada estaticamente, o que significa que todos os tipos de dados devem ser declarados antes de serem usados. Além disso, todos os dados declarados, sejam variáveis ou constantes, devem ser utilizados no código, senão o código não é compilado. Por essa razão todas as variáveis declaradas nos códigos em GO foram utilizadas

Matrizes e vetores não podem ser declarados com tamanhos não constante, ou seja, vetores e matrizes são estruturas de dados que consistem em uma sequência ordenada de elementos, com sua capacidade definida no momento de sua criação. Para se criar uma matriz ou um vetor com tamanho não definido é necessário declará-lo como uma slice.

Slice é um tipo de dado composto que representa uma sequência de elementos de um determinado tipo. Eles são semelhantes a arrays, mas com tamanho dinâmico, o que significa que o tamanho do slice pode ser alterado durante a execução do programa. Slices em Go são passados por referência, ou seja, quando um slice é passado para uma função, a função recebe um ponteiro para o slice original e pode alterá-lo diretamente. Nos códigos implementados em GOlang, a matriz A e os arrays X e B foram declaradas como slices, já que os tamanhos são apenas definidos em tempo de execução.

A memória é gerenciada automaticamente durante a execução do programa, tanto a alocação de memória quanto a desalocação de memória. O garbage collector é o responsável por isso e também pelo heap.

No heap é alocado as variáveis dinâmicas do código. O garbage collector monitora o heap e liberar memória quando as variáveis não são mais utilizadas. A pilha(stack) é utilizada para guardar variáveis locais e argumentos de função. A alocação e desalocação de memória

nesta região também é feita automaticamente. Quando uma função retorna, as variáveis são automaticamente desalocadas.

A chamada de função de GO é igual a do C, sendo necessário apenas chamar pelo nome da função e colocar os argumentos que serão passados.

Os comandos de fluxos de C e GO são iguais, como pode ser visto no código, as estruturas for e if. As duas linguagens possuem bibliotecas de conversão de strings, de tempo e matemática (math/rand).

Diferente de C, não há tratamento de exceções em GO. Por isso, em todos os códigos foi necessário o uso de um if com a variável err1 e err2 para verificar se houve alguma falha na conversão de string para inteiro.

Diferente de C, em que o programador tem total controle sobre as threads sendo executadas em um programa concorrente, podendo até determinar quantas tarefas são executadas em uma thread, em GO este controle é retirado do programador. O número de threads sendo utilizado e o número de tarefas por threads não pode ser determinado, apenas o número máximo(GOMAXPROCS).

A concorrência em GO é feita com o uso de goroutines (vale mencionar que a função main é considerada uma goroutine). As goroutine são assíncronas(a goroutine main e as outras goroutines continuam executando sem esperar as outras terminarem) e compartilham o mesmo espaço de endereço de memória do programa principal, sendo necessária sincronização delas por waitgroups ou mutexes. Goroutines só podem se comunicar entre si com o uso de canais (channel). Como a Eliminação de Gauss é um código simples, seu uso não se fez necessário.

O paralelismo em GO pode ser alcançado utilizando goroutines e sincronização juntamente com o for, que itera sobre o número máximo de cores de um computador. Isso difere muito de C. Em C foi utilizado a API OpenMP para paralelizar a eliminação de gauss. Porém, está API não existe para GO.

2.2 Códigos feito em Rust - Comparação RUST e C

Rust é uma linguagem compilada que foi projetada para ser “segura, prática e concorrente”. Diferente de outras linguagens consideradas seguras, Rust não possui um garbage collector e nem runtime.

As variáveis de Rust são imutáveis, ou seja, após compiladas, os valores das variáveis não podem ser modificadas. Se você precisar modificar uma variável, precisará declará-la como mutável usando a palavra-chave mut. Além disso, o acesso a variáveis é feito por meio de bindings, que são declarações que associam um nome a um valor. Para criar uma variável em Rust, você precisa primeiro declarar seu nome e, em seguida, associá-lo a um valor. (Ex.: let x: i32;) Let é o operador utilizado para declarar que o identificador é uma variável.

Em Rust, os vetores e matrizes são tipos de coleções que permitem armazenar vários valores do mesmo tipo. A principal diferença entre eles é que os vetores têm apenas uma dimensão, enquanto as matrizes têm duas ou mais. Porém, como havia mencionado anteriormente, seus tamanhos e valores são imutáveis uma vez declarados. Para que se possa declarar os tamanhos dos arrays X e B e da matriz A na linha de comando (mudar os tamanhos e os valores em tempo de execução) é necessário utilizar a função vec!. Matrizes em Rust são vetores aninhados.

Os identificadores em Rust devem ser sempre em letra minúscula e se necessário utilizar sublinhado para identificadores maiores.

Em C é utilizado OpenMP para a paralelização das funções e o autor do código implementa ThreadPool. Em Rust utilizamos Rayon para paralelizar e criar ThreadPool. Rayon é uma biblioteca de paralelismo de dados leve que converte facilmente um código sequencial em paralelo em apenas algumas linhas.

Como C, Rust possui uma biblioteca chamada thread que implementa funções ou métodos para a execução e criação de threads.

A organização da memória em RUST é gerenciada pelo sistema de propriedade (ownership system), evitando o uso de garbage collector. Ownership é feita em tempo de compilação.

Vinculações de variáveis têm uma propriedade no Rust: elas "têm propriedade" daquilo a que estão vinculadas. Isso significa que, quando uma ligação sai do escopo, o Rust libera os recursos vinculados do stack e até mesmo do heap. Por isso ownership da vinculação de variáveis e escopo são extremamente importantes e sempre devem ser levadas em conta. A organização de memória é igual a linguagem C, já que Rust também é uma linguagem de baixo nível.

Os comandos de fluxo de Rust são parecidos com C, exceto pelo comando for que possui uma sintaxe diferente, não admitindo o formato usado em C. O comando for pode ser visto sendo usado nos códigos em Rust no github.

3. Comparação por métricas

3.1 GO

A tabela a seguir mostra a quantidade de linhas para cada código de Go. É importante notar que em GO não há a possibilidade de fazer chunks ou pools.

	Linhas
GOsequential	113
GOconcurrent	122
GOparallel	132

3.2 RUST

A tabela abaixo mostra o número de linhas para cada código em Rust. Usamos cargo para fazer o projeto em Rust. Considerando que o arquivo main.rs possui 127 linhas temos os seguintes números de linhas.

	Linhas
sequential	127+21=148
thread	127+38=165

chunk	127+40=167
parallel/rayon	127+27=154
pool	127+27=154
chunk_pool	127+42=169

3.3 C

A tabela a seguir mostra o número de linhas para cada código em C. Para pool_threads e chunk_pool_threads foi considerada a soma do número de linhas do código principal somado ao thpool.c que é necessário para a sua compilação.

	Linhas
gauss - serial	212
threads_gauss	237
chunk_threads_gauss	259
openmp	215
pool_threads	242+557 = 799
chunk_pool_threads	260+557 = 817

3.4 Comparação

A linguagem C possui o maior número de linhas utilizadas enquanto GO tem a menor. GO é uma linguagem em que a concorrência e o paralelismo são simples de serem implementados através das goroutines, diferentemente das outras duas linguagens em que threads precisam ser manipuladas pelo programador.

4. Comparação de desempenho

Para fins de comparação, consideramos quatro tamanhos de matrizes: 10 e 100. No código em C há um limite no tamanho da matriz, esse limite é de 2000. Para todos os testes o seed para a geração do número aleatório foi fixado em 8 e o número de threads em 4.

Para fins de comparação, abaixo se encontram as especificações das máquinas utilizadas como teste.

Máquina 1:

- SO: KALI LINUX
- Intel(R) Core(TM) i5-7200H CPU @2.50 GHz
    - 2 Núcleos
    - 4 processadores lógicos
- 8 GB RAM
- 1 TB - HD

Máquina 2:

- SO: WINDOWS 11
- Intel(R) Core(TM) i5-9300H CPU @2.40 GHz
  - 4 Núcleos
  - 8 processadores lógicos
- 8 GB RAM
- 512GB - SSD

Os resultados abaixo estão todos em segundos e a última linha de cada é a média do tempo de cada coluna.

4.1 N = 10

LINUX

LINUX	seed = 8	matriz = 10	n_Threads = 4		
C					
gauss	threads	chunk	openmp	pool	chunk pool
0,000038	0,00199	0,00463	0,000635	0,001147	0,001917
0,000034	0,003	0,00313	0,000686	0,001185	0,000491
0,000035	0,00261	0,00311	0,000635	0,001223	0,00152
0,000038	0,00444	0,00384	0,000261	0	0,00237
0,000038	0,00308	0,00392	0,000652	0,001217	0,00298
0,0000366	0,003024	0,003726	0,0005738	0,0011578	0,0018556

GO		
sequential	concurrent	parallel
0,00000593	0,00001254	0,00002083
0,00000293	0,00004851	0,00002587
0,00000243	0,00001208	0,00000699
0,00000203	0,00005719	0,00006663
0,00000192	0,00001485	0,00007529
0,000003048	0,000029034	0,000039122

rust					
serial	threads	chunk	parallel	pool	chunk pool
0,000616	0,000582	0,002	0,00166	0,000535	0,000257
0,000058	0,00052	0,00241	0,00188	0,001002	0,000289
0,000098	0,000606	0,00217	0,00151	0,000654	0,000213
0,000051	0,000498	0,00288	0,00151	0,000502	0,000273

0,000102	0,00102	0,00234	0,00149	0,000543	0,000262
0,000185	0,0006452	0,00236	0,00161	0,0006472	0,0002588

WINDOWS

WINDOWS	seed = 8	matriz = 10	nThreads = 4		
C					
gauss	threads	chunk	openmp	pool	chunk pool
0,000035	0,001375	0,010542	0,002989	0,001444	0,00292
0,00001	0,001859	0,00972	0,002425	0,001768	0,002425
0,0000009	0,00162	0,009015	0,002588	0,001244	0,003367
0,000015	0,001774	0,009776	0,002783	0,001541	0,00269
0,000014	0,001769	0,011188	0,00178	0,001488	0,002671
0,00001498	0,0016794	0,0100482	0,002513	0,001497	0,0028146

Go		
sequential	concurrent	parallel
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Na máquina 2 o tempo de GO com n = 10 foi muito pequeno.

rust					
serial	threads	chunk	parallel	pool	chunk pool
0,0000539	0,0010181	0,0052085	0,00242	0,000571	0,0040049
0,0000554	0,0010332	0,004088	0,0014276	0,000585	0,0029419
0,0000748	0,0009993	0,0046313	0,0013451	0,0008523	0,00303
0,00000861	0,0014151	0,004533	0,0017114	0,0006462	0,0041108
0,00000542	0,0016617	0,004109	0,0014803	0,0005717	0,003641
0,000039626	0,00122548	0,00451396	0,00167688	0,00064524	0,00354572

4.2 N = 100

LINUX

LINUX	seed = 8	matriz = 100	nThreads = 4		

C					
gauss	threads	chunk	openmp	pool	chunk pool
0,00167	0,01566	0,0099	0,00313	0,001936	0,01347
0,003541	0,01418	0,0349	0,00218	0,00232	0,01365
0,003985	0,01597	0,02637	0,00912	0,002436	0,0134
0,001711	0,01539	0,01414	0,00297	0,00653	0,01648
0,002394	0,00706	0,02514	0,00345	0,006002	0,03947
0,0026602	0,013652	0,02209	0,00417	0,0038448	0,019294

GO		
sequential	concurrent	parallel
0,000993	0,000596	0,00178
0,00103	0,000571	0,00143
0,00054	0,000563	0,00132
0,000991	0,000598	0,00177
0,000993	0,000644	0,00137
0,0009094	0,0005944	0,001534

rust					
serial	threads	chunk	parallel	pool	chunk pool
0,039	0,0312	0,102	4,3	0,0517	0,112
0,043	0,028	0,106	3,9	0,051	0,108
0,039	0,028	0,106	3,9	0,047	0,118
0,043	0,027	0,113	4,5	0,0518	0,113
0,058	0,024	0,118	3,89	0,059	0,107
0,0444	0,02764	0,109	4,098	0,0521	0,1116

WINDOWS

WINDOWS	seed = 8	matriz = 100	nThreads = 4		
C					
gauss	threads	chunk	openmp	pool	chunk pool
0,00082	0,017404	0,099942	0,009847	0,003753	0,015042
0,000833	0,013044	0,105878	0,009488	0,004868	0,013387
0,001384	0,014723	0,105555	0,00899	0,002724	0,014109
0,000852	0,013902	0,114077	0,009398	0,003415	0,018053
0,001349	0,01532	0,103195	0,009513	0,005417	0,012825
0,0010476	0,0148786	0,1057294	0,0094472	0,0040354	0,0146832

Go		
sequential	concurrent	parallel
0,0006054	0,0006708	0,0018957
0,0009969	0,0005476	0,0019982
0,0006396	0,0009981	0,0019931
0,0016703	0,0005832	0,0023179
0,0005129	0,0009962	0,002992
0,00088502	0,00075918	0,00223938

rust					
serial	threads	chunk	parallel	pool	chunk pool
0,0261388	0,0180359	0,0682401	2,2435127	0,0435308	0,1164426
0,0299205	0,0212577	0,0675138	2,2537143	0,04237	0,1038488
0,0267316	0,0177272	0,067861	2,3055806	0,0425032	0,1193028
0,0268156	0,1870005	0,0730786	2,4041959	0,0442965	0,10475
0,0322433	0,020046	0,0718381	2,7864393	0,0445189	0,1180852
0,02836996	0,05281346	0,06970632	2,39868856	0,04344388	0,11248588

4.3 Comprando resultados

Os resultados acima mostram que houve grandes diferenças entre as máquinas na execução dos códigos, especialmente na linguagem GO.

Os desempenhos foram parecidos, tanto quando N = 10 e N= 100, contendo apenas uma divergência: nos códigos paralelos. O openMP em C continua tendo um melhor desempenho do que as outras linguagens, tirando de consideração a anomalia de GO na Máquina 2(Windows) que teve tempo 0s.

Como foi usado cargo em rust, primeiro é necessário o comando cargo build antes de cargo run, o que pode aumentar o tempo de rodar a eliminação de gauss. C também precisa ser compilado com make antes de executar os códigos. Apenas GO tem o comando go run para Windows, que faz ambos ao mesmo tempo.

5. Conclusão - Análise Geral

Rust e GO são linguagens novas que têm como principal objetivo a serem usadas para se desenvolver programas concorrentes, focando também em desempenho e segurança. Além disso, tentam ter mais legibilidade e maior facilidade em seu uso. Porém, quando comparadas com C, seus desempenhos deixam a desejar. Como são linguagens recentes, Rust e GO ainda precisam ser revisadas e atualizadas.

6. Bibliografia

MCNAMARA, Timothy Samuel. **Rust in Action**: Systems programming concepts and techniques. Nova York: Manning, 2021. ISBN 9781617294556.

COX-BUDAY, Katherine. **Concurrency in Go**: Tools & Techniques for developers. California: O'Reilly, 2017. ISBN 9781491941195.

Go. **The Go Programming Language.** Disponível em: https://go.dev/. Acesso em: 20 abr. 2023.

Rust. **Rust Programming Language**. Disponível em: https://www.rust-lang.org/. Acesso em: 20 abr. 2023.

DOXSEY, Caleb. **Introducing Go**: Build reliable, scalable programs. 1. ed. California: O'Reilly, 2016. ISBN 9781491941959.

STONE, Josh. Rayon: A data parallelism library for Rust. **Rayon**. Disponível em: https://github.com/rayon-rs/rayon. Acesso em: 20 abr. 2023.