Implementação da Eliminação de *Gauss* em *Go* e *Rust*

Louise Queiroz da Silva Bezerra, Lui Gill Aquini
Universidade Federal de Pelotas - UFPel, Pelotas, Brazil

I. Introdução

Rust e *Go* são duas linguagens de alto nível multifuncionais desenvolvidas para atuarem no escopo que estava sendo dominada por C e C++. São linguagens que implementam diferentes métodos para resolver problemas que os desenvolvedores possuiam no momento e precisavam de alguma ferramenta diferente para solucioná-los.

A. Rust

No ano de 2015, com o foco no desenvolvimento de um novo motor *web*, a Mozilla Research lança a linguagem Rust em sua primeira versão estável. Uma linguagem *open source* com foco em segurança de memória, performance atrelada diretamente com o uso do *hardware*, concorrência e paralelismo.

B. Go

Go foi desenvolvida pela *Google*, e foi lançada no ano de 2012. Para uso interno da empresa, a principal meta criar uma linguagem que pudesse substituir o C++, logo o objetivo era uma maior facilidade no desenvolvimento de *software*, e utilizar melhor as arquiteturas *multicores* que o mercado ia apresentando. Desde então, *Go* cresce no seu uso e diversas empresas a utilizam para desenvolver seus programas.

II. Eliminação de *Gauss*

É um algoritmo utilizado para se realizar sistemas de equações lineares. O método consiste em tranformar um sistema linear em um sistema tringular superior equivalente a entrada, deixando-o mais simples para resolver. O principal passo da execução, como o nome diz, é a eliminação de certas incógnitas da equação do sistema.

III. Diferenças de implementação

No desenvolvimento do método, nota-se uma maior facilidade para se escrever o código em *Go*, já que ela apresenta uma sintaxe mais próxima de C e *Java*, que são duas linguagens mais estruturadas no contexto de programação e desenvolvimento de *software*. *Go* também apresenta funcionalidades de segurança diferenter de *Rust*, as da linguagem desenvolvida pela Google são bem mais permissívas, trazendo uma facilidade para o desenvolvedor. *Rust*, por outro lado, possui o foco na segurança e gestão de memória, portanto as suas *features* geram uma maior dificuldade na escrita de código, mas um ganho em desempenho e em segurança.

IV. Código em C

A. Tipos de dados

- MAXN (constante): tamanho máximo de N;
- N (int): inteiro que define o tamanho da matriz;
- volatile: indica que as variáveis podem ser modificadas fora do escopo do programa e até por outros subprogramas;
- A (float): matriz de tamanho máximo;
- B (float): vetor;
- X(float): vetor solução;
- seed (int): semente usada na função srand e gerada na timeseed();
- uid(char): nome do user em character;
- argv (char **): passagem de parâmetro de ponteiro para um array de ponteiros que apontam para strings;
- row(int): inteiro que define o número de linhas;
- col(int): inteiro que define o número de colunas;
- timeval, etstart, etstop, timezone, tzdummy, tms, cputstart, cputstop (structs): estruturas de dados manipuladas no programa como informações de tempo;
- nom (int): inteiro utilizado para valor de uma linha de normalização;
- multiplier (float): o multiplicador serve para armazenar o resultado da divisão do elemento atual de A pelo elemento diagonal correspondente e após multiplicar pela norm para encontrar a row correta.

B. Acesso às variáveis

- O modificador volatile indica que o vínculo não é estático e pode ser alterado sem o conhecimento do programa principal;
- Um int e char são acessados como parâmetro da função parameters;
- A variável seed é atualizada caso o int do parâmetro seja 3;
- Caso o int do parâmetro seja maior ou igual a 2 então o tamanho da matriz (variável N) é atualizado;
- col e row servem para contadores de loop no for para percorrer toda a matriz;
- A, B e X também são acessadas ao tratar os inputs;
- etstart2 é utilizada para pegar o tempo inicial quando o clock é iniciado usando o times;
- etstop2 é utilizada para pegar o tempo final quando o clock é parado usando o times;

978-1-7281-7670-3/21/$31.00 ©2021 IEEE

- etstart e etstop fazem o mesmo que as suas segundas versões, porém utilizando o gettimeofday().

C. Organização da memória

- Variáveis A, B, X e N são alocadas na memória estáticamente;
- Os comandos printf() são usados para imprimir informações na tela;
- Ao longo do código são utilizadas diversas funções de biblioteca que armazenam os tempos da CPU e clock.

D. Chamadas de função

- void gauss(): chama a função principal que faz a eliminação de Gauss;
- main(): para inicializar o programa;
- unsigned int timeseed(): utiliza structs e a função retorna a semente para a função srand() com o tempo pego pelas bibliotecas do C;
- void parameters(int argc, char **argv): função que define os parâmetros do programa com base nos argumentos da linha de comando;
- void initializeinputs(): função que inicializa as matrizes A e B e o vetor X com valores aleatórios utilizando a função rand do C;
- void printinputs(): função que imprime a matrizes A e vetor B;
- void printX(): função que imprime o vetor solução X;
- gettimeofday: pega horário do dia pelo sistema.

E. Comandos de controle de fluxo

- É utilizado vários if e um else para a função parameters onde dependendo do resultado do parâmetro as variáveis são alocadas;
- São utilizados vários for para percorrer as matrizes e vetores nas funções de inputs e print;
- É utilizado for na função gauss para percorrer toda a matriz e vetor.

F. Número de linhas

Ao retirar linhas desnecessárias (comentários) o total de linhas do programa em c resultou em 171 linhas de código.

G. Comandos

Existem 17 comandos de fluxo neste código, sendo eles if, else e for, que servem para definição de parâmetros e percorrer matrizes e vetores.

H. Bibliotecas

Utiliza um total de 8 bibliotecas próprias do C para sua execução, sendo de aleatório, tempo, matemática e configurações do sistema.

V. Código em Golang

A. Tipos de dados

- MAXN (constante): tamanho máximo de N;
- N (int): inteiro que define o tamanho da matriz;
- A (float): matriz;
- B (float): vetor;
- X(float): vetor solução;
- norm (float): armazena um valor de normalização de linha;
- linha (float): contador para linhas;
- col (float): contador para colunas;

B. Acesso às variáveis

- A matriz A e vetores B e X são alocadas e inicializadas com a memória necessária para armazenar os elementos do valor, com a função make do Go, nesse caso o parâmetro é o tamanho de N;
- Para as variáveis locais (norm, linha e col) elas são utilizadas nas funções.

C. Organização da memória

- As matrizes são inicializadas usando make() para alocar memória dinamicamente.

D. Chamadas de função

- main(): inicializa o programa;
- time.Now(): função da biblioteca time para pegar o horário do sistema;
- rand.Seed(): gera uma semente aleatória;
- initialize.in(): inicializa a matriz e os vetores;
- initialize.in.test(): roda o teste do programa com valores previamente decididos;
- print.in(): imprime a entrada, a matriz A e vetor B;
- print.out(): imprime a saída, o vetor X;
- gaussElimination(): faz a eliminação de gauss com o apoio da matriz e vetores globais.

E. Comandos de controle de fluxo

- São utilizados diversos for para iterar através dos elementos da matriz;
- É utilizado um if para conferir o tamanho da matriz N, e dois ifs para configuração dos argumentos da linha de comando.

F. Número de linhas

Tem um total de 117 linhas, sendo menor que o código em C.

G. Comandos

Existem 15 comandos de fluxo neste código, sendo três if para verificação de tamanho e argumentos, e todos os outros são for para manipular a matriz e os vetores.

H. Bibliotecas

Utiliza um total de 5 bibliotecas próprias para sua execução. Sendo de matemática, random, time e conversão de strings.

VI. Código em Rust

A. Tipos de dados

- Vec<String>(vetor): vetor de strings;
- usize(int): tipo de dado inteiro;
- Vec<f64>(vetor): vetor de números em ponto flutuante de dupla precisão;
- Vec<Vec<f64»(matriz): matriz de números em ponto flutuante de dupla precisão;
- n(int): tamanho da matriz;
- num(usize): valor retornado após a função de argumentos.

B. Acesso às variáveis

- let start = Instant::now();: cria uma variável start do tipo Instant e atribui o valor retornado pela função Instant::now();
- let args: Vec<String> = env::args().collect();: cria uma variável args do tipo Vec<String> e atribui o valor retornado pela função env::args().collect();
- let num: usize = match args[1].parse::<usize>() ...: cria uma variável num do tipo usize e atribui o valor retornado pela expressão args[1].parse::<usize>();
- let mut a/b/x = vec![vec![0.0; num]; num];: cria uma matriz (se for a) ou vetor (caso seja b ou x) a com dimensões num x num inicializada com zeros;
- print.in();: chama a função print.in e passa como argumentos as variáveis a e b;
- let end = Instant::now();: cria uma variável end do tipo Instant e atribui o valor retornado pela função Instant::now();
- let duration = end.duration.since(start);: cria uma variável duration do tipo Duration e atribui a diferença entre os valores de tempo start e end;
- Acessa as matrizes e vetores utilizando o operador de acesso a elementos de vetor [], onde a[linha][col] acessa o elemento na linha e coluna especificadas;
- Os vetores a, b e x são inicializados na heap, que é a região de memória de acesso aleatório (RAM) onde os dados são armazenados durante a execução do programa.

C. Organização da memória

- A função env::args() retorna uma estrutura de dados que contém os argumentos passados para o programa pela linha de comando. Ao chamar o método collect() dessa estrutura, os argumentos são coletados em um vetor de strings (Vec<String>).
- let mut a = vec![vec![0.0; num]; num];: cria uma matriz a com dimensões num x num e aloca espaço na memória suficiente para armazenar todos os seus elementos;
- let mut b/c = vec![0.0; num];: cria um vetor b com num elementos e aloca espaço na memória suficiente para armazenar todos os seus elementos;
- Os vetores a, b e x são inicializados na heap, que é a região de memória de acesso aleatório (RAM) onde os dados são armazenados durante a execução do programa.

D. Chamadas de função

- Instant::now(): retorna uma instância da estrutura Instant, que representa um ponto no tempo;
- env::args(): retorna uma estrutura de dados que contém os argumentos passados para o programa pela linha de comando;
- args[1].parse::<usize>(): converte a segunda string do vetor args em um número inteiro sem sinal;
- initialize.in(): inicializa as variáveis a, b e x com valores aleatórios;
- gauss.elimination(): executa a eliminação gaussiana para resolver o sistema de equações lineares representado pelas matrizes a e b, e armazena o resultado no vetor x;
- initialize.in.test(): inicializa os vetores a e b com valores pré-definidos para um caso de teste específico;
- print.in(): imprime as entradas da matriz A e do vetor B;
- print.out(): imprime a saída que é o vetor solução X.

E. Comandos de controle de fluxo

- É utilizado um if para para verificar os argumentos;
- match: verifica se a conversão do segundo argumento passado para um número inteiro sem sinal foi bem sucedida. Se sim, atribui o valor a num. Se não, imprime uma mensagem de erro e retorna;
- Há quatro for dentro da função da eliminação de Gauss, para iterar através dos elementos da matriz e dos vetores;
- Um último if é usado para analisar o tamanho do vetor n e ver se é menor que 100.

F. Número de linhas

Tem um total de 109 linhas, sendo menor de todos os códigos apresentados.

G. Comandos

Existem 14 comandos de fluxo neste código, com o match se destacando por ser um novo comando presente em rust, ele permite comparar um valor com vários padrões (patterns) e executar o bloco de código correspondente ao primeiro padrão que casar com o valor.

H. Bibliotecas

Utiliza um total de 3 bibliotecas próprias para sua execução. Sendo de tempo e random.

VII. Comparação de desempenho

Ao comparar o desempenho dos três códigos é possível perceber que ao reproduzir o mesmo projeto em três linguagens de programação diferentes, o desempenho muda bastante. Na tabela abaixo é representado o tempo necessário para a CPU rodar o código e realizar a criação da matriz, vetores e eliminação de Gauss.

VIII. Conclusão

Deste modo, concluímos com esse trabalho que cada linguagem de programação visa uma área específica, e caso seja usada para outra modalidade irá funcionar porém terá um pior desempenho do que outra linguagem que otimiza esse processo melhor. No caso deste experimento, foi a linguagem C, ao demonstrar os menores tempos de execução da CPU mesmo com tamanhos altos de sua matriz. A linguagem Go foi muito próxima do C, porém possuiu dificuldades em matrizes muito grandes mas atingiu uma certa proximidade com o C, entretanto a linguagem Rust foi a com pior desempenho chegando a sair da casa dos milissegundos e entrando em segundos. Esse resultado é compatível com o visto em teoria visando que Rust é rápido, porém em alguns casos ao exigir uma manipulação extensiva de strings ou vetores seus concorrentes como C acabam por vencer. Enquanto o Go é focado em facilitar o uso e a produtividade do programador, e não necessariamente o desempenho extremo. Também possui verificação de limites de array (array bounds checking) que afetou este experimento. Logo, concluímos que neste caso a melhor solução apresentada foi a da linguagem C, e que as outras linguagens, principalmente Rust, visam outras aplicações.

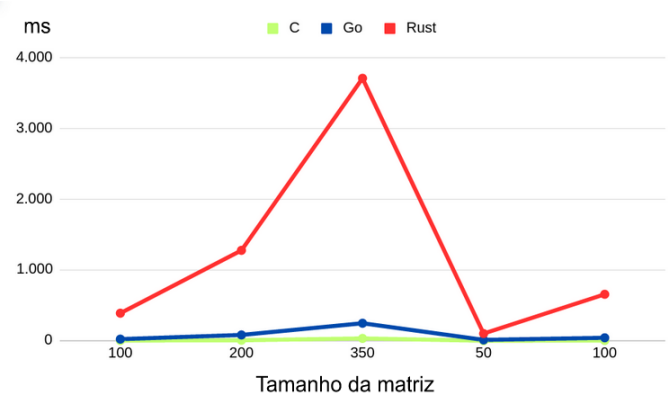

Fig. 1. Tabela de comparação do tempo da CPU.