Trabalho de implementação - Encontrar subsequências em um vetor de caracteres

Rafaella Busch
Nivea Martins

Computação Concorrente (ICP-117) - 2022.1

1. Descrição do Problema:

O problema consiste em encontrar a quantidade de vezes que uma subsequência escolhida pelo usuário aparece em um vetor de caracteres aleatórios, de dimensão também escolhida pelo usuário. Para a entrada, é pedido que o usuário dê a dimensão do vetor e da subsequência, além da quantidade de threads que serão criadas. A partir dessas informações, o programa gera um vetor de tamanho n preenchido com caracteres aleatórios de A a Z (apenas letras maiúsculas), e pede para que o usuário entre com a subsequência desejada, que é armazenada em outro vetor de dimensão igual à escolhida pelo usuário.

A partir desses dois vetores, uma solução sequencial para esse problema é descrita na função *qntSubsequências,* que percorre o vetor principal, e ao encontrar um caractere semelhante ao caractere na posição 0 do vetor de subsequência chama a função *ehSubsequencia,* que confere se a seção do vetor é de fato uma repetição da subsequência escolhida.

Como as principais estruturas de dados manipuladas pelo problema são vetores, é possível se beneficiar de uma solução concorrente para melhora de desempenho do mesmo, pois em um caso de concorrência, o vetor poderia ser dividido em partes menores onde as threads responsáveis procurariam simultaneamente pelas repetições da subsequência, aumentando a velocidade em casos de vetores com dimensão muito grande.

2. Projeto e Implementação da Solução Concorrente:

Para desenvolver uma solução concorrente para o problema proposto, é necessário dividir as tarefas principais da solução sequencial. Dessa forma, é

necessário seccionar o vetor em partes iguais para que as threads dividam o trabalho de forma igualitária. Há algumas formas de fazer isso, como divisão em blocos ou pulando posições de acordo com o número de threads, mas como o propósito desse problema é encontrar uma cadeia contínua de caracteres, a primeira solução se torna a ideal para o caso.

A função *ehSubsequencia* ainda será utilizada, pois seu papel é apenas verificar a existência de uma subsequência a partir de determinada posição, e como poderia haver casos de subsequências divididas em mais de uma thread, uma verificação na própria função principal da thread poderia ignorar casos desse tipo.

Sendo assim, a solução concorrente escolhida consiste em dividir o vetor principal em blocos de acordo com a quantidade de threads, e percorrer esses pedaços de vetor procurando algum caractere semelhante ao caractere da posição 0 do vetor da subsequência. Ao encontrar tal caractere, a thread chama a função *ehSubsequencia* que realiza sua verificação. A thread então incrementa uma variável compartilhada de acordo com a saída da função *ehSubsequencia* (realizando exclusão mútua para impedir condição de corrida).

3. Casos de Teste

Como o vetor é preenchido com valores aleatórios, apenas sua dimensão e a subsequência podem ser de escolha do usuário. Dessa forma, foram utilizados vetores de 2 tamanhos diferentes, e 3 subsequências de 3 tamanhos para cada vetor, além de 3 quantidades de threads. Os resultados dos testes foram:

- Para vetores de 100000000 posições:

  - Com 2 threads:

  subsequência de 1 caractere (3846184 repetições):

  Tempo Sequencial: 0.207031

  Tempo Concorrente: 0.113281

  Aceleração: 1.822758

  subsequência de 2 caracteres (148117 repetições):

  Tempo Sequencial: 0.218750

  Tempo Concorrente: 0.152344

Aceleração: 1.435895

subsequência de 4 caracteres (225 repetições):

Tempo Sequencial: 0.230469

Tempo Concorrente: 0.121094

Aceleração: 1.903223

- Com 4 threads:

subsequência de 1 caractere (3846184 repetições):

Tempo Sequencial: 0.210938

Tempo Concorrente: 0.074219

Aceleração: 2.842102

subsequência de 2 caracteres (148117 repetições):

Tempo Sequencial: 0.214844

Tempo Concorrente: 0.082031

Aceleração: 2.619058

subsequência de 4 caracteres (222 repetições):

Tempo Sequencial: 0.234375

Tempo Concorrente: 0.078125

Aceleração: 3.0000

- Com 8 threads:

subsequência de 1 caractere (3846184 repetições):

Tempo Sequencial: 0.207031

Tempo Concorrente: 0.046875

Aceleração: 4.416661

subsequência de 2 caracteres (148117 repetições):

Tempo Sequencial: 0.214844

Tempo Concorrente: 0.046875

Aceleração: 4.53338

subsequência de 4 caracteres (222 repetições):

Tempo Sequencial: 0.234375

Tempo Concorrente: 0.050781

Aceleração: 4.615407

- Para vetores de 10000000000 posições:

- Com 2 threads:

subsequência de 1 caractere (54222207 repetições):

Tempo Sequencial: 2.851562

Tempo Concorrente: 2.042969

Aceleração: 1.395793

subsequência de 2 caracteres (2081922 repetições):

Tempo Sequencial: 2.960938

Tempo Concorrente: 1.597656

Aceleração: 1.853301

subsequência de 4 caracteres (3173 repetições):

Tempo Sequencial: 3.265625

Tempo Concorrente: 1.882883

Aceleração: 1.734374

- Com 4 threads:

subsequência de 1 caractere (54222207 repetições):

Tempo Sequencial: 2.937500

Tempo Concorrente: 1.00000

Aceleração: 2.937500

subsequência de 2 caracteres (2081922 repetições):

Tempo Sequencial: 3.000000

Tempo Concorrente: 1.250000

Aceleração: 2.400000

subsequência de 4 caracteres (3173 repetições):

Tempo Sequencial: 3.460938

Tempo Concorrente: 1.164062

Aceleração: 2.973156

- Com 8 threads:

subsequência de 1 caractere (54222207 repetições):

Tempo Sequencial: 3.05081

Tempo Concorrente: 0.703125

Aceleração: 4.338929

subsequência de 2 caracteres (2081922 repetições):

Tempo Sequencial: 3.04685

Tempo Concorrente: 0.730469

Aceleração: 4.171087

subsequência de 4 caracteres (3173 repetições):

Tempo Sequencial: 3.308594

Tempo Concorrente: 0.746094

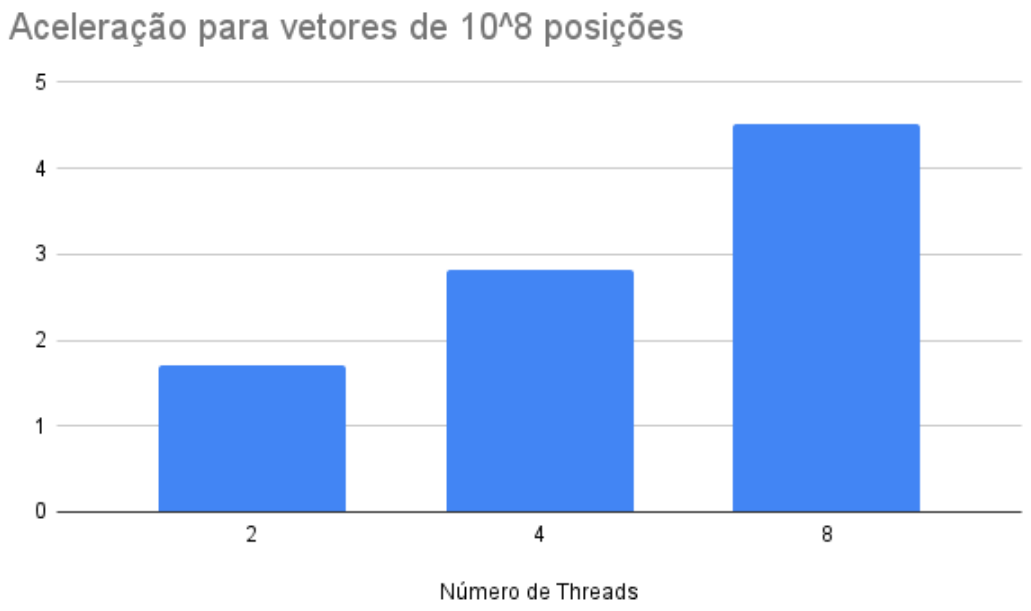
Aceleração: 4.434553

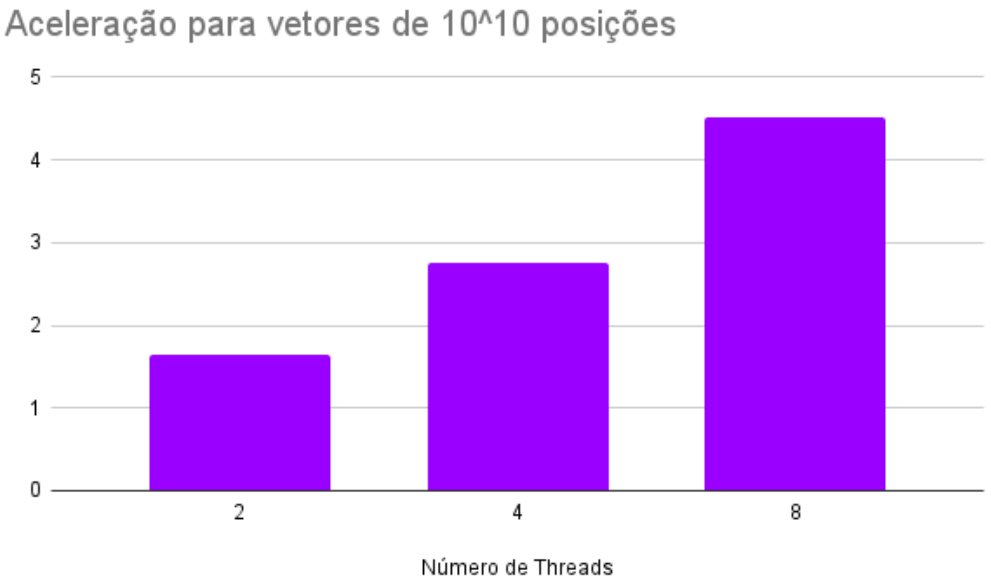
4. Avaliação e Desempenho

Os testes acima foram realizados em um processador com as configurações a seguir:

Sistema Operacional: Windows 11
Processador: Intel(R) Core(™) i7-8550 CPU @ 1.80GHz 1.99GHz;
Processadores Lógicos: 8;
Núcleos: 4.

Cada fase de teste foi executada 3 vezes, sendo a mais rápida explicitada no tópico anterior.





5. Discussão

Como visto a partir dos gráficos, o ganho de desempenho do programa é diretamente proporcional ao número de threads (se considerado um número de threads de até 8 unidades), o que já era esperado do programa. Tal comportamento de desempenho só é visível em vetores de tamanho acima de 10^8 posições, visto que em escopos menores os tempos de processamento concorrente e sequencial é praticamente zero.