

UNIVERSIDADE DO VALE DO ITAJAÍ

CIÊNCIA DA COMPUTAÇÃO – ESTRUTURA DE DADOS

PROF. MSc. EDUARDO ALVES DA SILVA

GUSTAVO HENRIQUE STAHL MÜLLER

**PARALELIZAÇÃO DE TRANSFORMAÇÕES LINEARES E  
SOMATÓRIOS DE VETOR COM OPENMP E PTHREADS**

ITAJAÍ

2021

**SUMÁRIO**

[**1.** **INTRODUÇÃO** 3](#_Toc69294012)

[**2.** **DESENVOLVIMENTO** 4](#_Toc69294013)

[**2.1.** **TRANSFORMAÇÕES LINEARES** 4](#_Toc69294014)

[2.1.1 EM CÓDIGO 5](#_Toc69294015)

[2.1.1 COMPARAÇÕES 8](#_Toc69294016)

[**2.2.** **SOMATÓRIO DE VETOR** 9](#_Toc69294017)

[2.2.1 EM CÓDIGO 9](#_Toc69294018)

[2.2.2 COMPARAÇÕES 10](#_Toc69294019)

[**3** **CONCLUSÃO** 11](#_Toc69294020)

[3.1 OPINIÃO 11](#_Toc69294021)

1. **INTRODUÇÃO**

Esse trabalho demonstra dois algoritmos: transformações lineares e somatórios de vetor, sendo paralelizados de duas maneiras diferentes, em C++.

Uma das maneiras é usando OpenMP, uma interface de código aberto para paralelização de algoritmos em alto nível, e a segunda é atraves de Pthreads, mais especificamente, sua implementação para Windows (WIN\_PTHREADS\_H).  
 O repositório de código está disponível em: https://github.com/GustavoHenriqueMuller/parallel-algorithms

1. **DESENVOLVIMENTO**

Essa seção contém a explicação dos dois algoritmos e suas resoluções em código.

* 1. **TRANSFORMAÇÕES LINEARES**

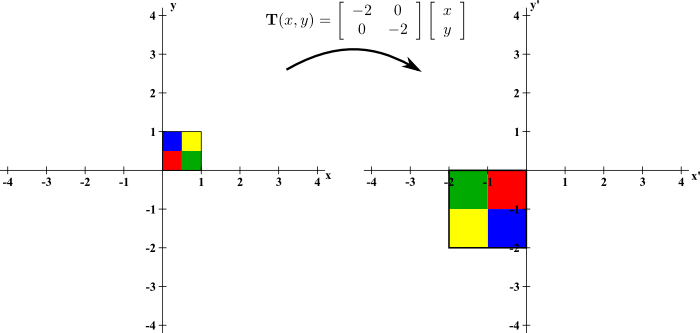
A aplicação de uma transformação linear, também conhecida como multiplicação de matrizes, é um algoritmo de alta importância para a ciência da computação. Através dele, é possível determinar posições de vetores após serem transformados por uma matriz que possui vetores que representam a base de um espaço vetorial.

Isso é extremamente útil para a computação gráfica, já que transformações lineares podem representar uma multitude de processamentos gráficos, incluindo, mas não limitado à, rotação e mudança de escala de imagens ou objetos.

Através de uma matriz que representa os vetores que compõem o objeto, e através de outra matriz que transforma esses vetores de uma certa maneira, o resultado da mulitplicação dessas duas matrizes será os vetores originais do objeto depois da transformação. É importante notar que multiplicação de matrizes é originalmente lida da direita para a esquerda, logo, a primeira matriz é aquela mais à direita.  
 No exemplo abaixo, a primeira matriz é composta por somente um vetor que indica a extremidade do bloco (vetor [1,1]). A segunda matriz, chamada de T e definida na figura, indica que o eixo *x* da primeira matriz [1,0] será multiplicado pelo vetor [-2,0], e que o eixo *y* da primeira matriz [0,1] será multiplicado pelo vetor [0, -2].

Isso resulta no vetor [-2, -2], que representa a extremidade do bloco após sofrer a transformação.

**Figura 1** – Transformação linear de inversão e ampliação sendo aplicada em um bloco. Fonte: Math Insight.

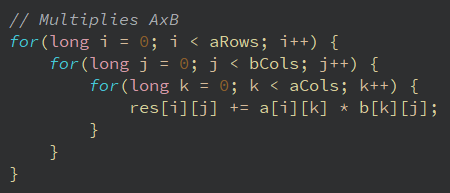


* + 1. EM CÓDIGO

A aplicação de transformações lineares de maneira sequencial é dada por:

**Figura 2** – Transformação linear de maneira sequencial.

Fonte: O autor.

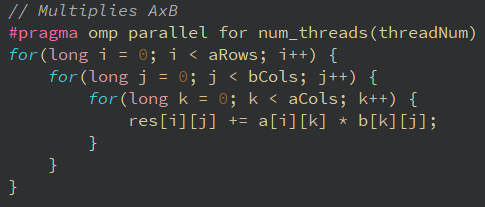


O código passa por cada valor de cada linha de A e multiplica por cada valor de cada coluna de B, e soma todas essas multiplicações para formar um dos resultados finais da matriz de resposta.

Em OpenMP, a única alteração à ser feita é o uso de uma diretiva que separa o trabalho de passar por cada linha de A entre várias threads diferentes, para alcançar melhor desempenho:

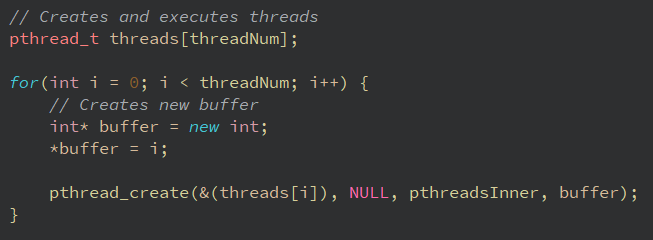
**Figura 3** – Transformação linear em OpenMP.

Fonte: O autor.



Por último, em Pthreads, o código é mais complexo devido ao menor nível de abstração oferecido. Primeiro, é preciso criar um conjunto de threads e definir qual é a sua função alvo. Como argumento dessa função, é passado um *buffer* alocado dinâmicamente que representa o índice da iteração atual. Isso será usado depois para calcular o intervalo de linhas que cada thread deve operar sobre.

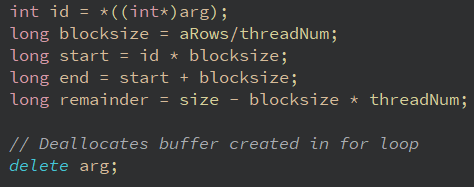
**Figura 4** – Criação de *threads*.

Fonte: O autor.  


Após a criação e início das *threads*, é calculado o tamanho do bloco, que é a quantidade de linhas da primeira matriz dividido pela quantidade de *threads*. O intervalo de linhas também é calculado à partir desse tamanho de bloco. O *buffer* mencionado anteriormente também é desalocado após ser utilizado:

**Figura 5** – Criação de variáveis locais à *thread*.

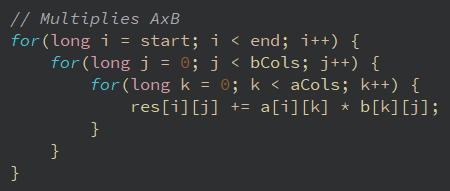
Fonte: O autor.



A multiplicação acontece no intervalo calculado anteriormente (*start* até *end*):

**Figura 6** – Multiplicação de matrizes feita por cada *thread* em seu intervalo respectivo.

Fonte: O autor.

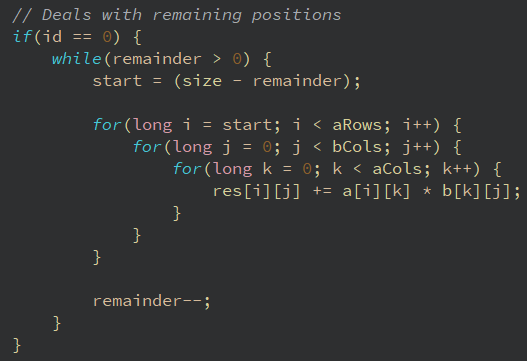


Por último, uma variável de resto (*remainder*) é calculada. Ela representa a quantidade de linhas da matriz que não foram pegas por nenhuma *thread*. Isso acontece quando o tamanho do bloco não é um número inteiro. Essa era uma situação que o OpenMP tratava automaticamente, mas com Pthreads, precisa ser lidada manualmente.

Para lidar com essa situação, utilizei um algoritmo muito simples: As linhas que não foram pegas por nenhuma thread são sequencialmente multiplicadas pela primeira thread criada. Esse algoritmo não é o mais efetivo, pois caso muitas casas não forem pegas, grande parte do algoritmo será sequencial e não paralelo, prejudicando o desempenho.

As linhas não pegas sempre serão as últimas, então é possível deduzir a posição da primeira linha através da quantidade de linhas não pegas e do tamanho de linhas. Esse valor representa o índice da linha de início (*start*):

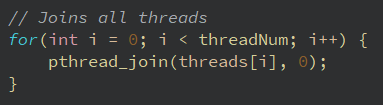
**Figura 7** – Lidando com linhas não pegas por threads.

Fonte: O autor.  


Finalmente, basta juntar todas as *threads*, assim garantindo a integral multiplicação das matrizes, consequentemente finalizando o algoritmo:

**Figura 8** – Juntando todas as *threads*.

Fonte: O autor.



* + 1. COMPARAÇÕES

A tabela abaixo mostra uma comparação dos tempos de execução das três implementações diferentes de aplicação de transformações lineares. As seguintes propriedadaes se aplicam aos testes:

* A multiplicação foi feita entre duas matrizes quadradas de mesmo tamanho.
* 4 *threads* foram usadas para OpenMP/Pthreads.
* O código executou em um processador i5-10400F, de 6 núcleos e uma frequência de 2.90 GHz até 4.30 GHz.

**Tabela 1** – Tempos de execução para Transformações Lineares.

Fonte: O autor.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **100 x 100 elementos** | **1000 x 1000**  **elementos** | **2000 x 2000**  **elementos** |
| **Sequencial** | 0.0039s | 4.227s | 18.33s |
| **OpenMP** | 0.0009s (~23%) | 1.008s (~23%) | 4.511s (~24%) |
| **Pthreads** | 0.002s (~51%) | 1.135s (~26%) | 4.754s (~25%) |

A porcentagem em parênteses das implementações paralelas representa a razão do tempo em relação ao tempo sequencial.

* 1. **SOMATÓRIO DE VETOR**

O somatório de todos os elementos de um vetor é um algoritmo simples, porém muito utilizado. Através do somatório, e então da divisão entre ele e a quantidade de elementos, a média aritmética simples do vetor é calculada.

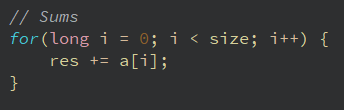
Com uma leve alteração do algoritmo, é possível que ele também compute médias aritméticas ponderadas, sendo que essas duas médias possuem amplo uso em estatística e servem de base para muitos outros conceitos, como variância, desvio médio e desvio padrão.

* + 1. EM CÓDIGO

O somatório de um vetor de maneira sequencial é dado por:

**Figura 9** – Somatório de vetor de maneira sequencial.

Fonte: O autor.

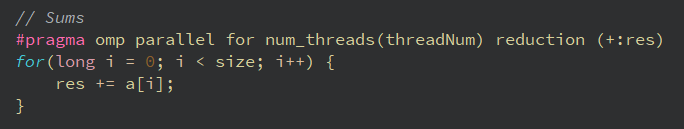


O código passa por cada valor do vetor e soma ele à variável de resultado, chamada de *res*.

Em OpenMP, o algoritmo permanece igual, porém é necessário usar uma diretiva que separa o trabalho do *for* igualmente para as *threads*, e também a aplicação de *reduction*, que evita a situação onde várias *threads* tentam escrever o seu resultado para a variável *res* ao mesmo tempo, causando somas incorretas.

**Figura 10** – Soma de vetor em OpenMP.

Fonte: O autor.



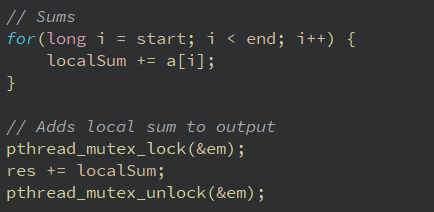
Em Pthreads, os algoritmos de cálculo de tamanho de bloco e o tratamento de espaços não pegos por threads são os mesmos da transformação linear. A única diferença entre essa implementação e a implementação sequencial é que cada *thread* possui uma variável de soma privada. Quando o algoritmo termina, essa variável de soma privada é somada à variável compartilhada de resultado (*res*).

Isso evita que a variável de soma compartilhada seja travada constantemente para evitar a situação de que duas *threads* escrevem nela ao mesmo tempo.

Essa situação era corrigida automaticamente com a cláusula *reduction* em OpenMP, mas deve ser corrigida manualmente em Pthreads pelas funções *pthread\_mutex\_lock()* *e pthread\_mutex\_unlock()*, que devem ser chamadas antes e depois que uma *thread* escreva à uma variável compartilhada, respectivamente.

**Figura 11** – Soma realizada à variável privada à *thread* (localSum), que é posteriormente somada à variável de resultado compartilhada.

Fonte: O autor.



* + 1. COMPARAÇÕES

A tabela abaixo mostra uma comparação dos tempos de execução das três implementações diferentes de somatório de vetor. As seguintes propriedadaes se aplicam aos testes:

* 4 *threads* foram usadas para OpenMP/Pthreads.
* O código executou em um processador i5-10400F, de 6 núcleos e uma frequência de 2.90 GHz até 4.30 GHz.

**Tabela 1** – Tempos de execução para Transformações Lineares.

Fonte: O autor.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **10.000.000 elementos** | **100.000.000**  **elementos** | **160.000.000**  **elementos** |
| **Sequencial** | 0.056s | 0.571s | 0.876s |
| **OpenMP** | 0.023s (~41%) | 0.219s (~38%) | 0.348s (~39%) |
| **Pthreads** | 0.021s (~37%) | 0.218s (~38%) | 0.348s (~39%) |

A porcentagem em parênteses das implementações paralelas representa a razão do tempo em relação ao tempo sequencial.

1. **CONCLUSÃO**

3.1 OPINIÃO

Com certeza é possível perceber a diferença de desempenho entre os algoritmos sequenciais e paralelos. O maior ganho ocorreu na transformação linear de duas matrizes 1000x1000 usando OpenMP, que obteve um tempo de somente 23% do tempo de execução original (ganho de ~4.34x).

No geral, OpenMP e Pthreads tiveram um ganho de desempenho muito próximo, mas considero OpenMP melhor para uso real pois ele fornece uma interface abstrata, relativamente simples e de fácil entendimento, em contraste à Pthreads, que oferece funções de mais baixo nível.