UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA
SISTEMAS DE INFORMAÇÃO

Nathália de Oliveira Ferrett

MÚLTIPLAS THREADS EM C

Relatório do Experimento

Limeira – SP
2024

SUMÁRIO

1. DESCRIÇÃO DO PROBLEMA

O programa deve trabalhar com cinco matrizes quadradas de mesmo tamanho, $An{\times}n$, $Bn{\times}n$, $Cn{\times}n$, $Dnxn$ e $Enxn$, de forma que sejam executados procedimentos sobre essas matrizes, considerando uma sequência de passos que podem ser executados em conjunto por mais de um *thread.* O programa deve exibir o tempo gasto com cada operação realizada, assim como o tempo total de processamento.

Considerando como matrizes de entrada as matrizes A, B e C, primeiramente, o programa deve calcular a soma entre A e B e atribuir o resultado à matriz D. Estando a matriz D preenchida, o programa deve calcular a multiplicação entre as matrizes C e D, nesta ordem, e armazenar o resultado na matriz E, a qual, ao final, deve ser reduzida, ou seja, ter todos seus elementos somados, resultando em um único valor.

Ainda, todas as matrizes devem estar gravadas em arquivos e nenhum dado deve ser exigido do usuário durante a execução do programa. Logo, tanto as matrizes de entrada devem já ter valores armazenados em arquivo durante a execução quanto às matrizes D e E também devem ter, após serem calculadas, todos os seus elementos armazenados em arquivo.

Assim, tendo em vista o problema descrito, é possível separar a descrição em sete passos:

1. Leitura das matrizes A e B no arquivo;
2. Soma das matrizes A e B;
3. Gravação da matriz D;
4. Leitura da matriz C;
5. Multiplicação das matrizes C e D;
6. Gravação da matriz E;
7. Redução da matriz E e exibição do valor resultante;

Devido ao fato de alguns passos poderem ser executados simultaneamente por mais de um *thread*, é possível definir três tipos de *threads* a serem usados no programa: de leitura, de escrita e de processamento.

O *thread* de leitura é responsável por efetuar a leitura dos dados armazenados nos arquivos das matrizes de entrada, o *thread* de escrita é responsável por transcrever em arquivo os dados das matrizes resultantes no programa e o *thread* de processamento é responsável por efetuar as operações do programa. Dessa forma, os *threads* de processamento e de leitura podem ser executados simultaneamente com qualquer outro tipo de *thread*, enquanto o *thread* de escrita somente pode ser executado de forma simultânea com outro *thread* de escrita se cada gravação estiver ocorrendo em arquivos diferentes.

Portanto, temos que:

- O Passo 1 deve possuir dois *threads* de leitura e efetuar de forma simultânea a leitura das matrizes A e B.
- O Passo 2 deve possuir uma quantidade T de *threads* de processamento para efetuar a soma entre as matrizes A e B.

- O Passo 3 e o Passo 4 devem ser feitos de forma simultânea, com uma *thread* de escrita para efetuar a gravação e um *thread* de leitura.
- O Passo 5 deve possuir uma quantidade T de *threads* de processamento para efetuar a multiplicação entre as matrizes C e D.
- O Passo 6 e o Passo 7 devem ser feitos de forma simultânea com uma *thread* de escrita para efetuar a gravação e T *threads* de processamento para efetuar a redução.

Ademais, a quantidade de *threads* de processamento deve ser informada na linha de comando para instanciar o programa, a qual deve conter, também, o tamanho das matrizes e os nomes de todos os cinco arquivos que devem ser usados para armazenar as matrizes, nesta ordem.

A estrutura da linha de comando a ser usada é:

```
C/C++
./programa T n arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

Os experimentos a serem analisados devem considerar:

- O programa possuindo um, dois ou quatro *threads* de processamento.
- O programa possuindo matrizes de tamanho 100x100 ou 1000x1000.

Ou seja, um exemplo de linha de comando a ser usada é:

```
C/C++
./programa 1 100 arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

Tendo em vista as alterações possíveis da linha de comando, são esperados seis experimentos diferentes, variando quantidade de *threads* e tamanho de matrizes.

Ao final, o programa deve exibir para o usuário: o resultado da redução da matriz E, o tempo gasto com a soma, o tempo gasto com a multiplicação, o tempo gasto com a multiplicação e o tempo total de processamento.

2. INSTRUÇÕES PARA COMPILAR O PROGRAMA

O programa é feito em linguagem C pura. Para ser compilado em ambiente Linux, deve ser feito uso do compilador GCC, o qual deve estar previamente instalado no sistema operacional antes da execução do programa.

Para compilar o programa, é possível fazer o uso do comando `make` por meio do Makefile fornecido ou compilar o programa manualmente utilizando a linha de comando abaixo:

```
C/C++
gcc -pthread projetoSO.c -o prog
```

Após compilado, execute o arquivo com qualquer uma das linhas abaixo de maneira a testar o experimento descrito.

1. Para testar a execução com um *thread* e matrizes quadradas de tamanho 100:

```
Unset
./prog 1 100 arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

2. Para testar a execução com dois *threads* e matrizes quadradas de tamanho 100:

```
Unset
./prog 2 100 arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

3. Para testar a execução com quatro *threads* e matrizes quadradas de tamanho 100:

```
Unset
./prog 4 100 arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

4. Para testar a execução com um *thread* e matrizes quadradas de tamanho 1000:

```
Unset
./prog 1 1000 arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

5. Para testar a execução com dois *threads* e matrizes quadradas de tamanho 1000:

```
Unset
./prog 2 1000 arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

6. Para testar a execução com quatro *threads* e matrizes quadradas de tamanho 1000:

```
Unset
./prog 4 1000 arqA.dat arqB.dat arqC.dat arqD.dat arqE.dat
```

3. GRÁFICOS COM OS TEMPOS DE EXECUÇÃO

Tendo em vista os seis experimentos definidos acima, após a compilação do programa e execução dos testes, foram obtidos os seguintes resultados:

- Para matriz de tamanho 100 e uma thread de processamento:
  Tempo soma: 0.000031 segundos.
  Tempo multiplicação: 0.002142 segundos.
  Tempo redução: 0.000016 segundos.
  Tempo total: 0.002189 segundos.

- Para matriz de tamanho 100 e duas *threads* de processamento:
  Tempo soma: 0.000095 segundos.
  Tempo multiplicação: 0.000137 segundos.
  Tempo redução: 0.000155 segundos.
  Tempo total: 0.000387 segundos.

- Para matriz de tamanho 100 e quatro *threads* de processamento:
  Tempo soma: 0.000748 segundos.
  Tempo multiplicação: 0.000219 segundos.
  Tempo redução: 0.000228 segundos.
  Tempo total: 0.001195 segundos.

- Para matriz de tamanho 1000 e uma thread de processamento:
  Tempo soma: 0.002916 segundos.
  Tempo multiplicação: 2.459677 segundos.
  Tempo redução: 0.001404 segundos.
  Tempo total: 2.463997 segundos.

- Para matriz de tamanho 1000 e duas *threads* de processamento:
  Tempo soma: 0.000166 segundos.
  Tempo multiplicação: 3.091697 segundos.
  Tempo redução: 0.000251 segundos.
  Tempo total: 3.092114 segundos.

- Para matriz de tamanho 1000 e quatro *threads* de processamento:

Tempo soma: 0.000244 segundos.

Tempo multiplicação: 4.262671 segundos.

Tempo redução: 0.000448 segundos.

Tempo total: 4.263363 segundos.

Dessa forma, foi possível definir quatro gráficos de comparação entre os tempos de soma, multiplicação, redução e processamento.

Ainda, de maneira a facilitar a visualização dos dados de cada experimento, cada gráfico foi separado em dois: um contendo exclusivamente experimentos com matrizes de tamanho 100 e outro, contendo matrizes de tamanho 1000.

Assim, neste relatório, estão sendo considerados oito gráficos para a análise dos dados, com quatro tipos de tempos e dois tamanhos de matrizes para cada um desses tempos.

Em cada gráfico, o eixo Y representa o tempo em segundos.

3.1 Tempo de soma

Para matrizes de tamanho 100:

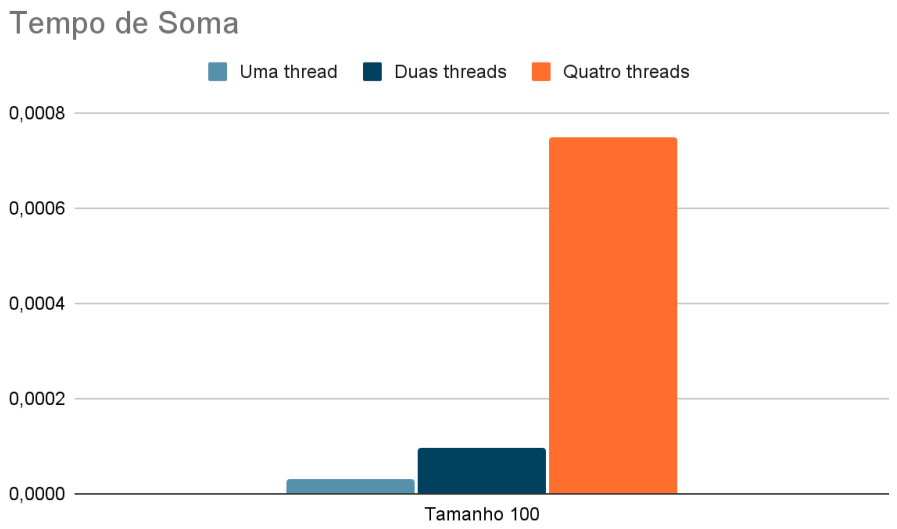

Gráfico 1 - Tempo de soma para matrizes (nxn), em que $n = 100$

No Gráfico 1, é possível observar que, para duas matrizes quadradas de tamanho 100, a soma utilizando apenas um *thread* principal é significativamente mais rápida se comparada aos outros dois experimentos, devido ao tempo gasto para a criação das *threads* nos outros dois casos. Ainda, entre a utilização de dois ou quatro *threads*, a criação de mais *threads* demanda mais tempo de processamento, mesmo que a operação de soma seja feita em menos tempo, o que

faz com que o tempo total para quatro *threads* seja maior do que o tempo total gasto por duas *threads*.

Para matrizes de tamanho 1000:

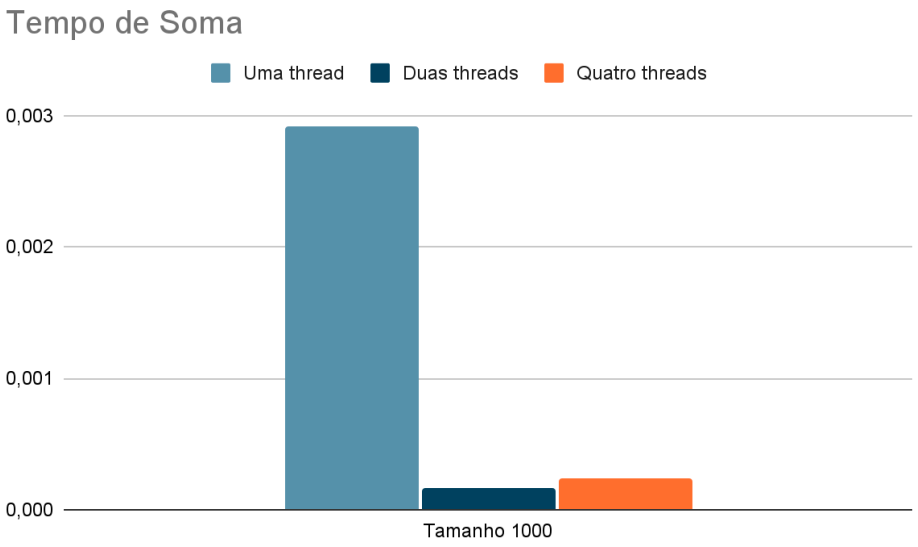

Gráfico 2 - Tempo de soma para matrizes (nxn), em que $n = 1000$

Ao contrário do que foi observado no Gráfico 1, o Gráfico 2 evidencia que, para matrizes quadradas de tamanho 1000, a utilização de apenas um *thread* principal se mostrou significativamente mais lenta; mesmo que a criação dos *threads* nos outros experimentos tenha exigido parte do tempo de processamento, a operação foi realizada em um tempo mais curto. O tempo gasto para a criação de *threads* é mais relevante se comparado ao experimento entre dois e quatro *threads*, visto que os dois *threads* demandaram menos tempo de criação e, por isso, o tempo gasto no processamento total se mostrou mais rápido do que para os quatro *threads*.

3.2 Tempo de multiplicação

Para matrizes de tamanho 100:

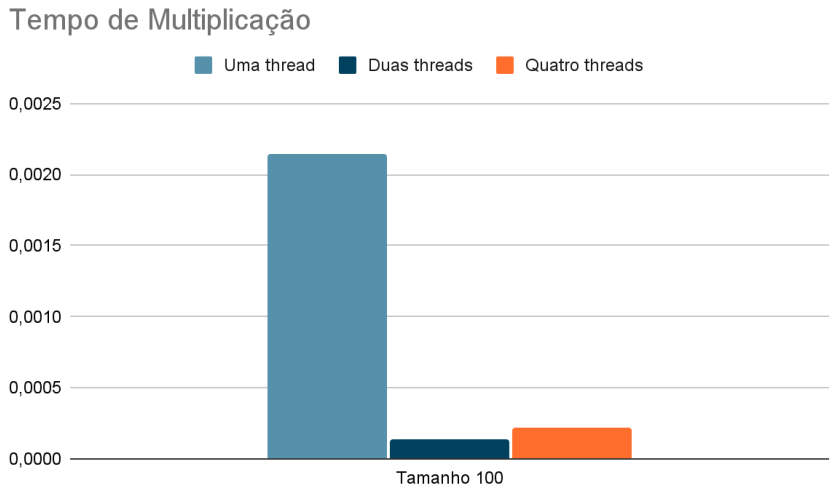

Gráfico 3 - Tempo de multiplicação para matrizes (nxn), em que $n = 100$

De forma similar ao que foi observado no Gráfico 2, o Gráfico 3 mostra que a execução da multiplicação com apenas um *thread* se revelou mais lenta, enquanto que, para dois e quatro *threads*, o tempo de processamento total, mesmo considerando o tempo para criação de *threads*, foi mais rápido. Ainda, de maneira similar, dois *threads* se mostraram mais eficientes para a operação do que quatro *threads*, devido a menor quantidade de tempo gasto para criação total dos *threads* e a influência desse fator no tempo total.

Para matrizes de tamanho 1000:

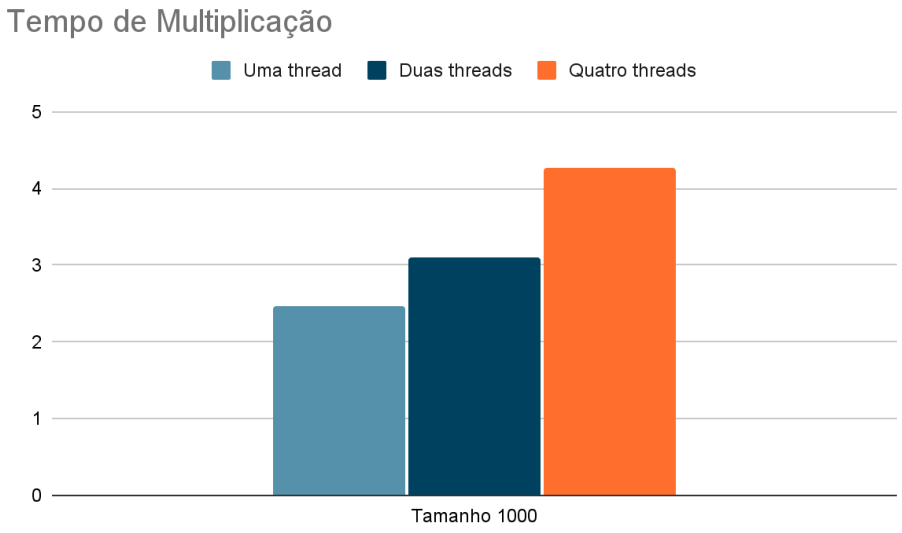

Gráfico 4 - Tempo de multiplicação para matrizes (nxn), em que $n = 1000$

O Gráfico 4 evidencia que, para matrizes quadradas de tamanho 1000, o uso de múltiplas *threads* se mostra menos eficiente. O tempo de processamento é maior para uma maior quantidade de *threads*, mostrando a relevância, neste caso, do tempo gasto para a criação de *threads* em comparação a execução que usa apenas um *thread* principal. Assim, para a multiplicação, o tempo gasto para a criação das *threads* não compensa a rapidez da execução da operação por múltiplas *threads*.

3.3 Tempo de redução

Para matrizes de tamanho 100:

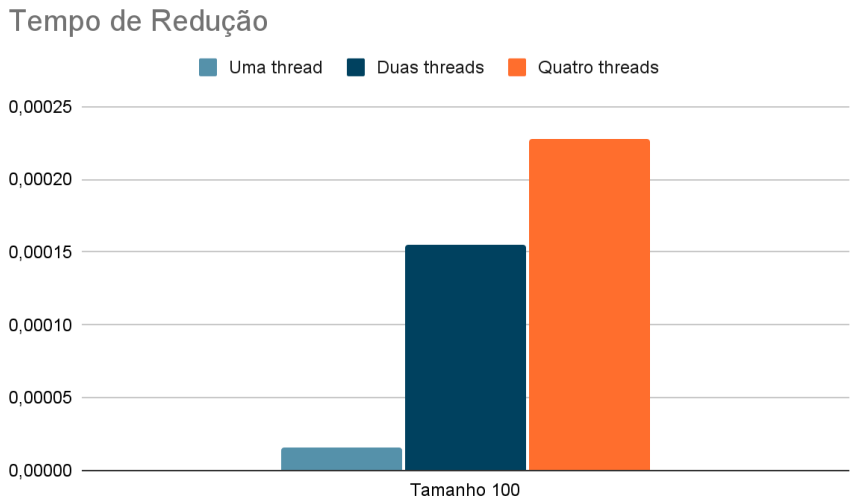

Gráfico 5 - Tempo de redução para matrizes (nxn), em que $n = 100$

Conforme demonstrado pelo Gráfico 5, a execução da redução com apenas uma *thread* é significativamente mais rápida que o tempo necessário para a redução com dois ou quatro *threads*, mostrando-se mais eficiente. O tempo despendido para a criação de *threads* aumenta o tempo de processamento, reduzindo, assim, a eficiência do experimento conforme o número de *threads* aumenta.

Para matrizes de tamanho 1000:

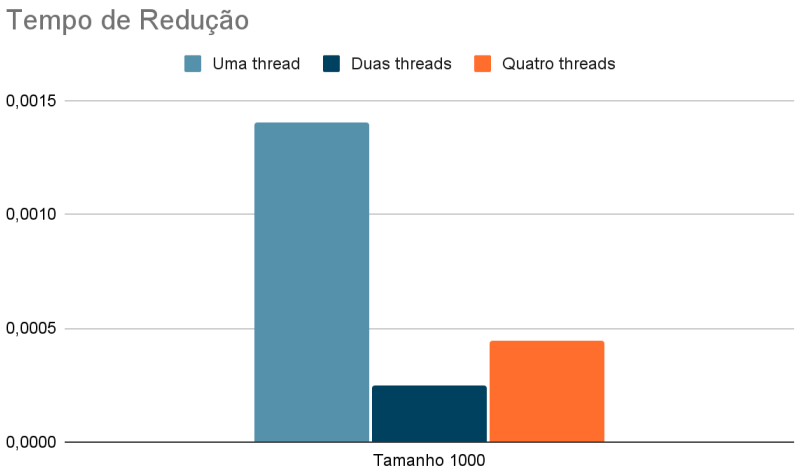

Gráfico 6 - Tempo de redução para matrizes (nxn), em que $n = 1000$

Um comportamento oposto ao Gráfico 5 é observado no Gráfico 6: o tempo para a operação com apenas um *thread* principal é significativamente maior. O tempo despendido para a criação das *threads* é compensado pela performance no uso de *threads* na operação. Ainda, entre dois e quatro *threads*, o processamento com dois *threads* se mostra mais rápido, evidenciando o gasto de tempo para a criação do dobro de *threads*.

3.4 Tempo total de processamento

Para matrizes de tamanho 100:

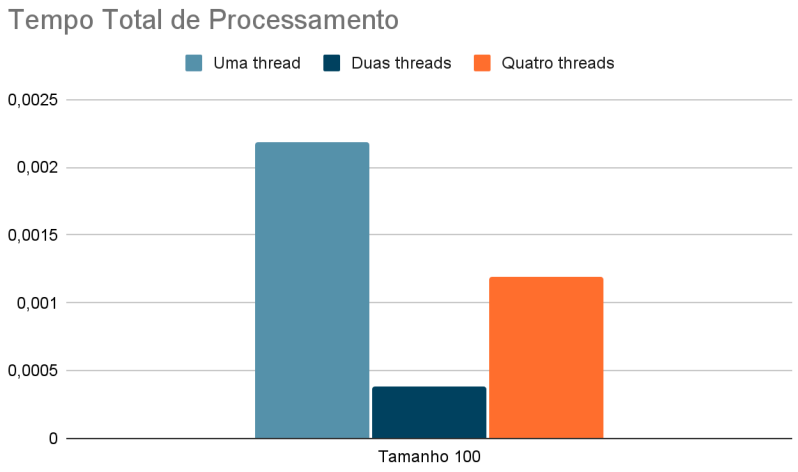

Gráfico 7 - Tempo total de processamento para matrizes (nxn), em que $n = 100$

O Gráfico 7 evidencia que, embora o tempo gasto com a soma e redução para múltiplas *threads* seja superior, a diferença do tempo de processamento para a

multiplicação entre um único *thread* e múltiplas *threads* é maior do que as diferenças para esses mesmos experimentos para as operações de soma e redução, fazendo com que o tempo total de processamento das operações seja maior para um único *thread*. No entanto, se comparado o tempo total entre dois e quatro *threads*, é possível observar que o ganho de performance não neutraliza o tempo gasto com a criação de *threads,* fazendo com que, para uma menor quantidade de *threads*, o tempo total seja menor.

Para matrizes de tamanho 1000:

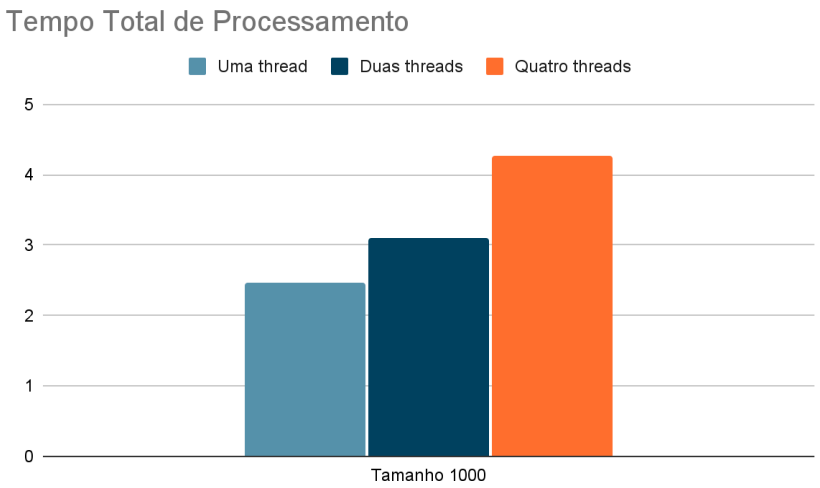

Gráfico 8 - Tempo total de processamento para matrizes (nxn), em que $n = 1000$

O Gráfico 8 mostra que, apesar da soma e redução apenas com o *thread* principal ter despendido mais tempo, o processamento para a operação de multiplicação efetuada unicamente por esse mesmo *thread* são mais rápidas, devido ao fato de o tempo gasto para a criação de *threads* nos outros dois experimentos ser significativo o suficiente para atrasar o processamento total, fazendo com que o tempo total de processamento para um único *thread* seja menor. Ademais, a diferença do tempo de processamento para a multiplicação entre múltiplas *threads* e o único *thread* é maior do que as diferenças para esses mesmos experimentos para as operações de soma e redução, fazendo com que o tempo total tenha menos influência do tempo gasto para essas duas operações.

Ainda, para duas *threads*, o tempo de processamento total é menor do que para quatro *threads*, o que demonstra que o tempo despendido para a criação do dobro de *threads* ocasiona em um tempo total de processamento maior, mesmo que a operação seja feita em menos tempo.

4. CONCLUSÃO

Tendo em vista os resultados obtidos a partir dos seis testes e da análise dos gráficos, é possível concluir que a velocidade de processamento depende não somente da quantidade de threads, mas também do tamanho da matriz na qual as operações serão efetuadas. Para matrizes (nxn), em que $n = 100$, usar um único *thread* principal é menos eficiente em termos de tempo total de processamento. No entanto, o contrário é observado para matrizes (nxn), em que $n = 1000$.

A criação de threads demanda uma quantidade maior de recursos computacionais e, portanto, consome um tempo significativo, podendo elevar o tempo de processamento total, mesmo que, para a realização de operações, a performance seja melhor. No caso, por exemplo, da multiplicação para matrizes de tamanho 1000, essa condição impacta a execução de maneira que a execução com apenas um *thread* se mostre mais rápida, apesar da realização da operação em si ser mais lenta.

Dessa forma, o uso de *threads* não é sempre eficiente. A criação e gerenciamento de *threads* pelo sistema operacional adiciona uma carga de tempo considerável, podendo tornar o uso de um único principal *thread* mais conveniente, a depender da circunstância.

Contudo, ainda é notável o ganho de tempo de processamento em relação a realização das operações por *threads* múltiplas. Quando a operação é custosa em termos de duração de execução para um único *thread,* a divisão da tarefa faz com que o tempo total de processamento seja consideravelmente mais curto, mesmo com o tempo que é despendido para a criação do recurso, como é possível observar para a multiplicação de matrizes de tamanho 100. Assim, o uso de *threads* ainda pode se mostrar benéfico em situações em que a operação a ser executada é suficientemente complexa para que a duração mais rápida da execução se sobreponha ao custo de tempo de criação das *threads*.

Além disso, é possível encontrar um equilíbrio entre o tempo de execução das operações e o tempo necessário para a criação das *threads*. Embora a criação de *threads* seja custosa, ainda existe um ganho de desempenho significativo, situação essa que é evidenciada pelo tempo total de processamento mais curto usando dois threads para matrizes de tamanho 100, onde se observa uma compensação do tempo gasto na criação de *threads* pela rapidez da execução das operações, o que demonstra uma grande vantagem para o experimento.

Portanto, sob as condições do experimento proposto, é necessário considerar o tempo de criação de *threads* em relação ao ganho de performance esperado. Para algumas operações, o uso de um único *thread* pode ser mais vantajoso, tendo em vista que não há consumo de tempo para alocar recursos e gerar novos *threads*, enquanto que, para outras, mais de um *thread* em execução são mais benéficos.

5. ACESSO AO CÓDIGO FONTE E AO VÍDEO DEMONSTRATIVO

Para acessar o código fonte, assim como os arquivos utilizados para os experimentos, acesse o repositório do GitHub no link a seguir: https://github.com/naferrett/projeto-SO

Para assistir ao vídeo mostrando o código fonte, a compilação, trechos dos arquivos de entrada e as execuções do programa para os seis experimentos, acesse o vídeo no link a seguir: https://youtu.be/AeVC0i1OEAw?si=gXJHcnfn9Um38H6g

Observação: Os comentários para a documentação do código foram adicionados em momento posterior à gravação do vídeo. Visto que o vídeo tem como objetivo explicar cada uma das funções implementadas, o uso dos comentários durante sua exibição não é necessária.

Para a adição dos comentários, a implementação das funções **não** foi alterada, permanecendo a mesma que é vista no vídeo.

Assim, o código documentado pode ser encontrado no repositório do GitHub no link acima.