# Análise do impacto do uso de múltiplas linhas de execução no processo de busca pelo menor valor de uma matriz

Willian Cavaller Faino
Graduação em Ciência da Computação
UNIOESTE - Universidade Estadual
do Oeste do Paraná
Cascavel, Brasil
willianfaino@hotmail.com

## I. Introdução

Uma das características mais importantes em grande parte dos softwares é o desempenho, um software cuja execução seja muito demorada não é agradável, principalmente no patamar atual das máquinas computacionais, onde o poder de processamento é dezenas ou até centenas de vezes maior do que de uma máquina de uma década atrás. Incluir o uso de múltiplas linhas de execução (*Multithreading*) é uma forma de, em certos casos, solucionar o problema da falta de desempenho, já que o uso dessa técnica consiste em dividir o trabalho realizado pelo algoritmo em *n* linhas de execução que são executadas simultaneamente, reduzindo consideravelmente o tempo de execução do software.

Neste trabalho será aplicada a técnica de *Multithreading* em um algoritmo que realiza a busca pelo menor valor em uma matriz dinamicamente alocada e verificar o impacto dela no desempenho do software.

## II. Descrição do Problema

A busca pelo menor valor em uma matriz dinamicamente alocada não é o tipo de atividade que apresente problemas de desempenho, pois percorrer uma matriz e atualizar a variável que armazena o menor valor não demandam uma grande capacidade de processamento e, portanto, tem execuções costumeiramente rápidas. Porém este processo simples é interessante por ser de fácil compreensão e por também ser fácil de ser implementado com a técnica de múltiplas linhas de execução, o que além de resultar em um código curto, permite também identificar, na sua codificação e especificação, sua seção crítica.

## III. Descrição das Implementações

A implementação do algoritmo foi feita na linguagem de programação C++. O software não possui uma interface gráfica, portanto as informações que serão requeridas deverão ser inseridas diretamente no terminal durante a execução. Inicialmente, serão requisitadas as dimensões da matriz a ser alocada (número de linhas e colunas), em seguida deverá ser inserida a quantidade de threads que realizarão a varredura na matriz para buscar o menor valor e então será requisitado se será ou não feito o uso de semáforo para bloquear a seção critica durante a execução das threads.

Após serem inseridas as informações, se for sinalizado que será utilizado semáforo, este será inicializado, caso contrário este passo não acontecerá e o programa irá para o próximo: Criar um vetor dinâmico de threads baseado na quantidade informada pelo usuário. Em seguida, um vetor de índices será inicializado, neste vetor serão armazenados os índices onde cada thread irá iniciar e finalizar sua varredura na matriz. Então, a matriz será finalmente alocada e preenchida com diversos valores aleatórios utilizando a função *rand()*. Ao fim da alocação e preenchimento, será informado no terminal a seguinte mensagem: "*Alocacao e preenchimento da matriz finalizados*".

Ao fim destes passos iniciais, a variável que armazena o menor valor da matriz será inicializada com o valor da posição $M_{0,0}$ da matriz para que a variável não fique com algum lixo de memória em seu valor. Após isto, uma função que realiza o *split* da matriz será executada: Baseado na quantidade de threads, a função irá determinar os índices de inicio e fim de varredura nas posições do vetor *v_index[]*. Agora serão criadas as threads dentro de um laço *for*.

Com as threads criadas, uma variável *clock* será inicializada com o valor da quantidade de clocks que foram necessários para executar até esta parte do algoritmo, e então serão iniciadas as execuções das *n* threads. Após a execução das mesmas, a variável *clock* será atualizada, contendo agora o valor de clocks durante a execução das threads, esta variável será convertida para representar quantos milissegundos (ms) foram necessários para executar as *n* threads. Por fim, serão exibidos no terminal: O menor valor da matriz (encontrado durante a execução das threads) e o tempo de execução das threads, para que seja analisado o impacto do uso de múltiplas threads. Caso não seja usado um semáforo para realizar o bloqueio da seção crítica, ao fim dos passos anteriores, uma informação adicional será apresentada: Uma validação do menor resultado, comparando o valor encontrado pelas threads sem bloqueio da seção crítica com o valor encontrado pelo método de busca por força bruta, que é busca feita por uma única linha de execução (garante o resultado certo por impossibilitar o acesso simultâneo da variável global que armazena o menor valor da matriz), se a comparação apontar que ambos os valores são iguais, será informado que a busca apresentou um resultado válido, caso contrário apresentará que o resultado da busca das threads sem bloqueio foi inválido.

## IV. Materiais Utilizados

Para a realização deste trabalho foram utilizados diversos materiais, os quais estão listados a seguir:

- Notebook Acer Aspire F5 573G;
  - Processador Intel Core i7-7500U;
    - Dual core;
    - 4 threads;
    - Velocidade da CPU (medida/base): 3,51 GHz/2,70GHz;
  - RAM instalada: 16GB;
- Geany (para implementação do código);
- Prompt de comando (para visualização das informações e inserção dos argumentos requisitados pela aplicação);

## V. Metodologia das Análisees

As análises foram feitas em cima de uma variedade de caso de testes, que variam desde a quantidade de threads utilizadas até aa definição do uso ou não de semáforo. A primeira variação é feita na quantidade de threads, onde as mesmas foram variadas entre 1 (execução single-thread), 2, 4 e 8 threads. Como o processador da máquina utilizada possui apenas 4 threads, foi realizado um teste com 8 para verificar o comportamento do software ao executar mais threads simultâneas do que o disponibilizado pelo hardware.

Em seguida, foi feita uma variação no tamanho da matriz, inicialmente foram feitos testes em matrizes quadradas, com dimensões 100x100, 1000x1000 e 10000x10000. Como as threads "dividem" a matriz em linhas, foram feitos testes também onde a quantidade de linhas e colunas fossem diferentes para verificar se isso impactaria algo no desempenho do algoritmo, para esta parte dos testes foram usadas matrizes com dimensões 100x1000, 100x10000, 1000x100 e 10000x100.

Por fim, a última análise foi feita referindo-se ao uso ou não de semáforo para realizar o bloqueio da seção crítica, evitando o acesso simultâneo da variável global que armazena o menor valor da matriz. O objetivo desta análise é verificar o impacto que o uso de semáforo tem no desempenho e se não o usar pode invalidar o resultado da busca pela modificação simultânea da variável do menor valor.

## VI. Análise dos Resultados

As seguintes imagens apresentam a execução de um caso de teste do algoritmo.

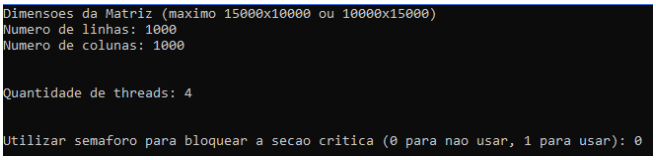


*Figura 1 - Inserção de argumentos requisitados pelo algoritmo*

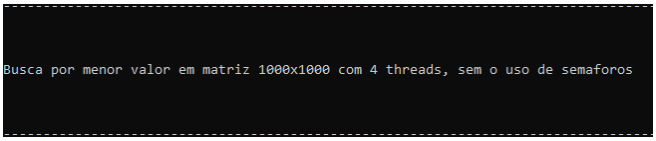


*Figura 2 - Após a inserção, o algoritmo informa que tipo de execução será realizada*

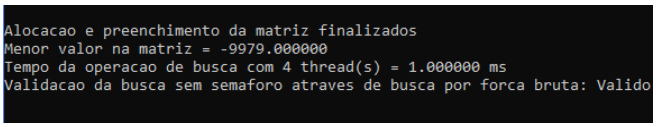


*Figura 3 - Apresentação dos resultados no prompt de comando após o fim da execução da(s) thread(s)*

Na figura 1, vemos que foi escolhida uma matriz de dimensões 1000x1000, que será processada por 4 threads sem o bloqueio de um semáforo. Em seguida a figura 2 nos mostra que o software nos informa as dimensões da matriz e como ela será processada, isso marca o início da preparação do algoritmo e da execução dos threads.

Por fim, a figura 3 ilustra os resultados do processamento da matriz, neste caso vemos o menor valor encontrado na matriz pelas 4 threads sem semáforo (-9979), o tempo da operação de busca com as 4 threads (1 milissegundo) e a validação da busca através de uma busca por força bruta, que indica se a busca multithread sem bloqueio obteve o resultado correto (Válido).

Após executados todos os casos de teste, foram geradas tabelas com os resultados e, combinando os resultados das tabelas, foram gerados gráficos (um para execução sem semáforo e outro para execução com semáforo) que apresentam de forma mais compreensível o impacto do numero de threads, tamanho da matriz e uso do semáforo. Estas tabelas e gráfico foram deslocados para o fim deste relatório para que pudessem ter sua visualização melhorada, por terem um tamanho maior.

Os resultados, tanto da primeira parte da execução, realizada sem uso de semáforo, quanto da segunda parte, com uso de semáforo, ficaram divididos em 4 tabelas, cada uma apresentando todas as variações de tamanho da matriz, porém com um valor único de threads (e.g.: A tabela 1 apresenta os resultados das execuções com 1 thread, a Tabela 2 exibe resultados da execução com 2 threads, a Tabela 3 os resultados da execução com 4 threads, etc).

Nas tabelas 1, 2, 3 e 4, vemos os resultados das execuções sem semáforos, com 1, 2, 4 e 8 threads. Analisando as quatro tabelas, nota-se que os únicos casos de teste que apresentam um tempo de execução com maior capacidade de demonstrar os impactos que buscamos são os casos que envolvem matrizes de 10000x10000, já que os outros casos não tiveram mais de 3 ms de duração em suas execuções, portanto será dado o devido enfoque nos casos de maior tempo com as matrizes de 10000x10000. Nas tabelas até agora analisadas, notamos que o uso de mais threads acarretou num melhor desempenho da busca, diminuindo um pouco o tempo necessário para realizar a mesma na matriz. É necessário informar que em todos os testes realizados sem semáforo durante a implementação do trabalho os resultados foram validados ao fim da execução, tendo todos os resultados das

buscas retornado o valor correto para suas determinadas matrizes. Isso, porém não implica que o método nunca apresentará erros, tendo em vista que o acesso simultâneo à variável global de menor valor para alterá-la sempre terá uma chance, seja ela mínima ou não, de uma alteração simultânea executada por múltiplas threads que resultarão num valor equivocado em relação ao que se esperava.

Voltando à análise dos resultados obtidos, em resumo, todas as execuções sem semáforo apresentaram resultados corretos nas buscas e tiveram um desempenho que melhorava a cada aumento na quantidade de threads, até mesmo ao alocar 8 threads para a execução, o que poderia resultar num tempo um pouco maior devido ao fato do processador da máquina dispor de apenas 4 threads simultâneas, então o fator escalonamento poderia gerar um pequeno declínio no desempenho.

Agora, analisando os resultados obtidos com uso de semáforo, inicialmente comparando os valores com os das quatro primeiras tabelas, notamos que não há nenhuma mudança grande no tempo de execução, porém destacaremos estas mudanças da mesma forma, pois como já mencionado anteriormente, a busca por menor valor em uma matriz não é uma atividade que demande muito tempo de processamento, portanto qualquer variação nesse tempo é interessante ser mencionada.

Novamente, serão usadas como base principal para a análise as execuções sobre matrizes 10000x10000. No caso de execução com 1 thread, obtivemos um valor muito semelhante ao das execuções sem semáforo (211ms sem semáforo, 212ms com semáforo). A execução com 2 threads foi a que apresentou a maior variação entre as execuções com e sem semáforo: Enquanto a execução sem semáforo levou 140ms, o uso do semáforo elevou esse tempo para 194ms, estes 54ms de diferença, porém, não apresentam nenhuma perda significante já que se trata de milissegundos, portanto a diferença é praticamente imperceptível ao se observar a execução do algoritmo. Agora partindo para a execução com 4 threads, esse foi o único caso onde a execução com semáforos teve tempo menor do que a execução livre de bloqueio, onde a primeira apresentou um tempo de 99ms enquanto a sem bloqueio demandou 126ms para ser executada. Por fim as execuções com uso de 8 threads tiveram a mesma semelhança entre si do que as execuções de linha única de execução: Com semáforo levou 107ms para executar e sem semáforo 106ms.

## VII. Conclusões

Após a organização dos resultados dos testes e feitas as análises, é perceptível que, apesar de ser um processo simples e rápido, a busca pelo menor valor em uma matriz dinâmica pode ser ainda mais rápida com o uso de múltiplas threads. Também conclui-se que o uso de semáforo não apresentou nenhum grande impacto negativo no desempenho do algoritmo, porém esta técnica garante que teremos sempre o resultado correto no fim da execução e apesar de todas as execuções sem semáforo terem apresentado o mesmo, é de se esperar que em alguns casos, mesmo que específicos, o acesso simultâneo à variável que armazena o resultado interfira negativamente na atividade de obter o resultado correto.

O software implementado apresentou melhora no desempenho ao utilizar múltiplas threads, tanto com quanto sem semáforos, portanto fica claro que o uso de multithreading é interessante para uma busca em matrizes, preferencialmente com uso de semáforo, que não apresentou impactos no desempenho e ainda nos garante a atomicidade do acesso à variável que determinará o resultado final, ou seja, sempre teremos o resultado esperado.

## VIII. Tabelas e Gráfico com resultados das Execuções Sem Uso de Semáforo

| Threads | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 0 | 2 | 211 | 0 | 2 | 1 | 3 |

*Tabela 1 - Tempos de execução com 1 thread.*

| Threads | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 1 | 1 | 140 | 0 | 1 | 0 | 2 |

*Tabela 2 - Tempos de execução com 2 threads*

| Threads | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 0 | 1 | 126 | 0 | 1 | 0 | 1 |

*Tabela 3 - Tempos de execução com 4 threads.*

| Threads | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 0 | 1 | 106 | 0 | 1 | 0 | 1 |

*Tabela 4 - Tempos de execução com 8 threads.*

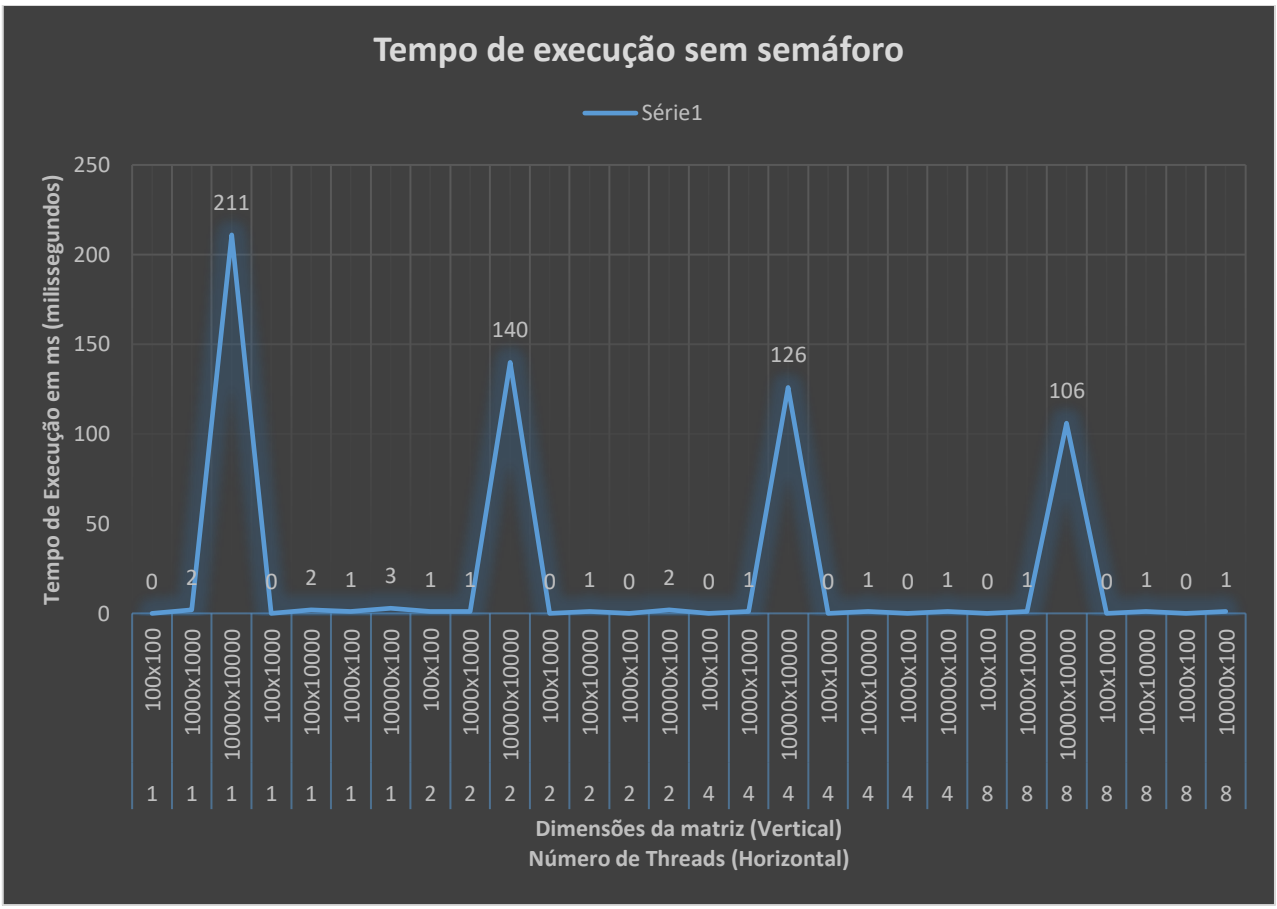


*Gráfico 1 - Gráfico combinado dos valores de tempos de execução das 4 tabelas referentes à execução sem uso de semáforo.*

## IX. Tabelas e Gráfico com resultados das Execuções Com Uso de Semáforo

| Threads | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 0 | 2 | 212 | 1 | 2 | 1 | 2 |

*Tabela 5 - Tempos de execução com 1 thread.*

| Threads | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 0 | 1 | 194 | 0 | 1 | 0 | 1 |

*Tabela 6 - Tempos de execução com 2 threads.*

| Threads | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 0 | 1 | 99 | 0 | 1 | 1 | 1 |

*Tabela 7 - Tempos de execução com 4 threads.*

| Threads | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
|---|---|---|---|---|---|---|---|
| Dimensões da matriz | 100x100 | 1000x1000 | 10000x10000 | 100x1000 | 100x10000 | 1000x100 | 10000x100 |
| Tempo de execução (ms) | 0 | 2 | 107 | 0 | 1 | 0 | 0 |

*Tabela 8 - Tempos de execução com 8 threads.*

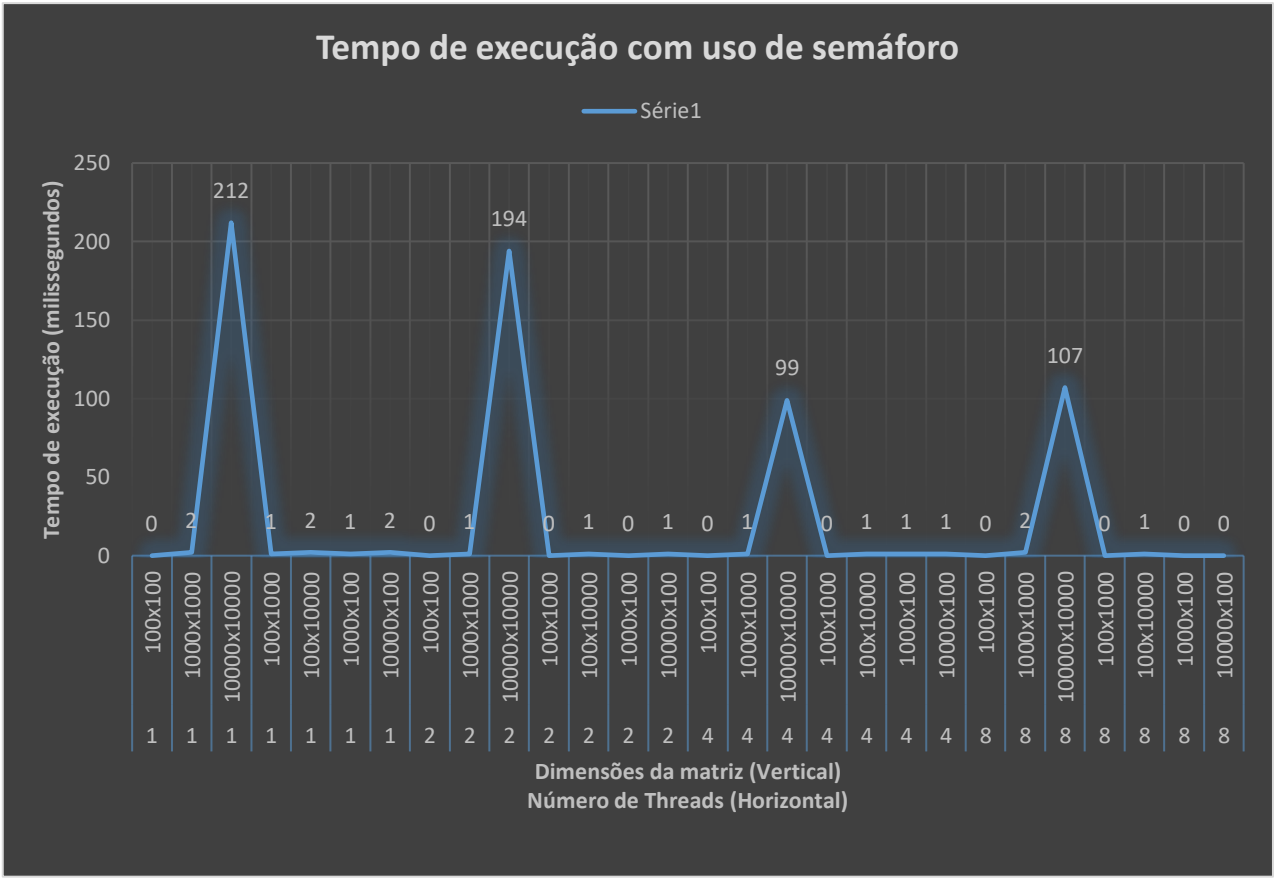


*Gráfico 2- Gráfico combinado dos valores de tempos de execução das 4 tabelas referentes à execução com uso de semáforo.*