Sistemas Operacionais 2023/2

ERICK KENZO KOMATI

MARLLON CHRISTIANI DOS SANTOS RIBEIRO

Introdução à Programação Multithread com PThreads

SERRA

2023

SUMÁRIO

1. INTRODUÇÃO

A programação paralela, ao permitir a execução simultânea de múltiplas tarefas, desempenha um papel essencial na otimização de sistemas computacionais. Este trabalho explora esse conceito por meio de testes realizados em uma matriz de números primos, dividida em macroblocos aos quais são designadas as threads, e aborda a importância do gerenciamento de **seções críticas**. A programação concorrente é crucial nesse contexto, permitindo a execução simultânea de partes independentes de um programa, o que pode resultar em significativos ganhos de desempenho.

O foco central desta pesquisa reside na análise dos dados adquiridos de dois respectivos computadores, com ênfase na comparativa do desempenho dos algoritmos em função do número de threads utilizados. Além disso, serão investigados o impacto do tamanho e da quantidade da divisão de processamento da matriz nas métricas de desempenho. Esses testes visam oferecer insights sobre como a escolha do número de threads, bem como a gestão eficiente da concorrência, podem impactar positivamente a eficiência de uma aplicação, contribuindo para a compreensão do comportamento desses fatores no contexto da programação paralela e concorrente

A matriz principal usada nos teste tem o tamanho **12000x12000**, enquanto os macroblocos variam de **1x1** a **12000x12000** de tamanho.

2. Levantamento da configuração completa dos computadores

Como dito na introdução, iremos avaliar como os diferentes componentes de um computador mais antigo porta-se em comparação a um computador mais novo/atual. Abaixo, encontra-se a tabela que mostra os componentes das máquinas, lado a lado.

CPU	PC 1	PC 2
Sistema operacional	Windows 10 Pro	Windows 10 Home
Modelo	AMD Ryzen 5 5600x	Intel Core i3 2100
Frequência Base	3.70 GHz	3.10GHz
Frequência Boost	4.60 GHz	NO
Técnica de Fabricação	7nm	32nm
Quant. Núcleos Físico	6	2
Quant. Núcleos Lógicos	12	4
Instruções especiais suportadas	MMX(+), SSE, SSE2, SSE3, SSE4.1, SSE4.2, SSE4A, X86-64, AMD-V, AES, AVX, AVX2, FMA3, SHA	MMX,SSE,SSE2,SSE3,SSE4.1,SSE4.2,EM64T, VT-x, AVX
Ano de Lançamento	2021	2011

Tabela 1.1: Especificações da CPU dos computadores utilizados

Memória Principal	PC 1	PC 2
Tecnologia	DDR4	DDR3
Frequência Aparente	1333 MHz	665.1 MHz
Capacidade total	16 GB	16 GB
Quantidade Módulos	2	2

Tabela 1.2: Especificações da memória principal dos computadores utilizados

2.1. Arquitetura usada

Em geral, programas compilados para x64 podem ter um desempenho ligeiramente melhor em sistemas operacionais de 64 bits, especialmente ao lidar com grandes conjuntos de dados, devido ao aumento na quantidade de registradores disponíveis.

Optamos por utilizar a arquitetura x64 nos testes realizados neste trabalho, pois possui desempenho superior em comparação com a arquitetura x86. Essa escolha fundamenta-se no fato de que a versão x64 oferece suporte a uma capacidade de memória melhor.

PC 1	x64	x86
Serial	5,649 s	6,852 s
Paralelo	1,575 s	1,864 s

Tabela 2.1: Tempo de execução dos macroblocos de tamanho 500x500 nas arquiteturas, PC 1.

PC 2	x64	x86
Serial	16,221 s	17,388 s
Paralelo	7,039 s	8,202 s

Tabela 2.2: Tempo de execução dos macroblocos de tamanho 500x500 nas arquiteturas, PC 2.

3. Comparação PC 1 e PC 2

Neste tópico, apresentam-se os resultados dos testes realizados nos computadores correspondentes. Ao analisar os dados de tempo de processamento durante a execução paralela no PC 1 e PC 2 em relação à quantidade de macroblocos (variando de 1 a 12 mil), observa-se um comportamento peculiar. Notavelmente, há uma fase em que o tempo de execução permanece constante, mesmo com o aumento na quantidade de macroblocos. No entanto, à medida que o tamanho do macrobloco se aproxima do tamanho da matriz principal, nota-se um aumento significativo no tempo de execução. Essa tendência sugere que existe um ponto crítico em que o desempenho começa a ser impactado à medida que os macroblocos se aproximam das dimensões da matriz principal.

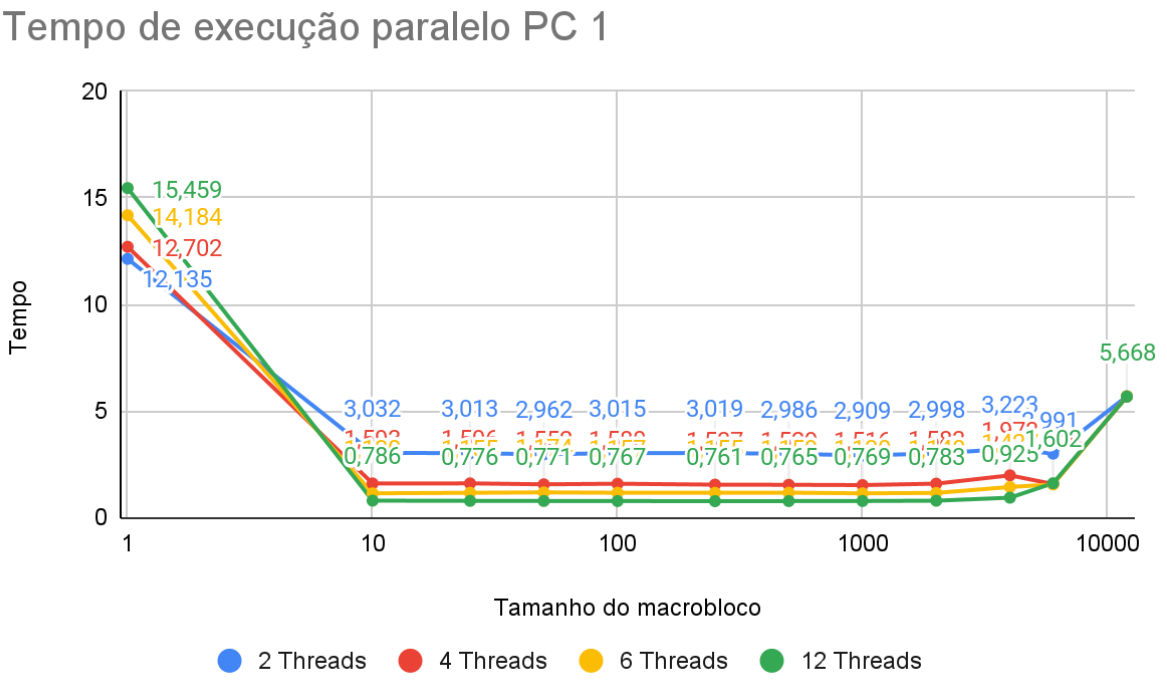

Imagem 1.1: Relação de tempo de execução e tamanho de macrobloco do PC 1.

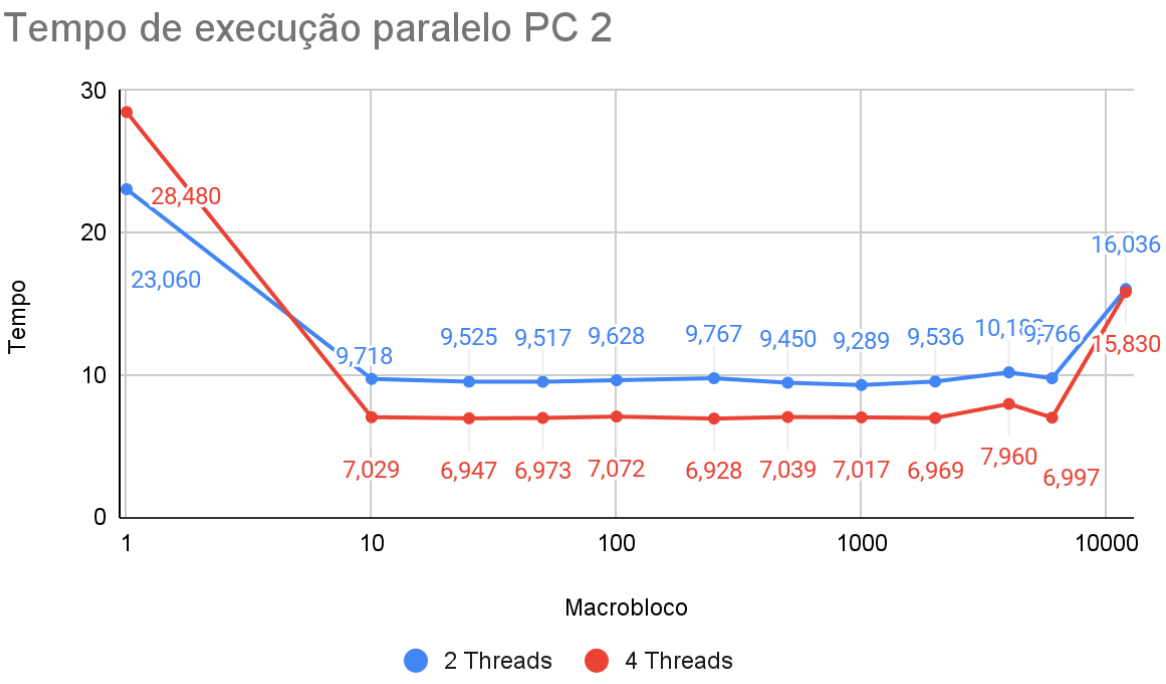

Imagem 1.2: Relação de tempo de execução e tamanho de macrobloco do PC 2.

Quando dividimos a matriz em macroblocos para processamento paralelo, a eficiência do paralelismo depende de como esses blocos são distribuídos entre as threads. Se o tamanho do bloco for muito pequeno, o overhead de sincronização das threads pode superar os benefícios do processamento paralelo. Por outro lado, se o tamanho do bloco for muito grande, algumas threads podem ficar ociosas enquanto outras ainda estão processando, levando a subutilização do paralelismo.

A eficiência do paralelismo pode ser afetada pela granularidade da tarefa (tamanho dos blocos de trabalho), e encontrar o equilíbrio certo é crucial. Nas imagens 1.1 e 1.2, a região que esse equilíbrio acontece fica mais perceptível.

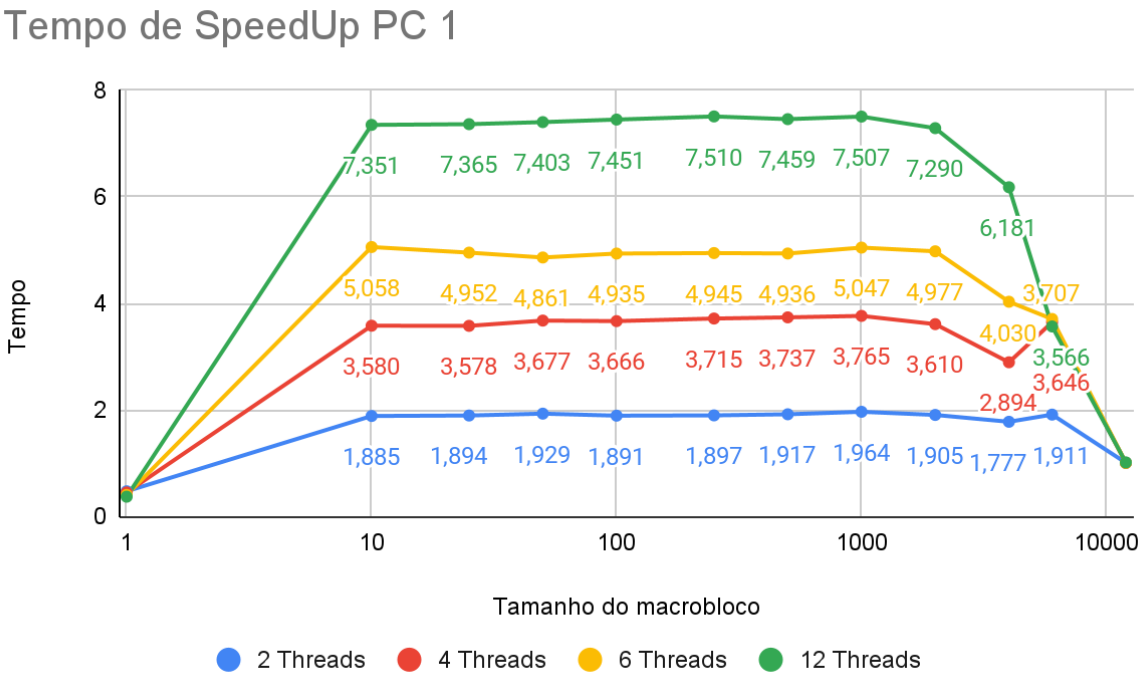

Imagem 2.1: Speedup, tempo serial e tempo multithread do PC 1.

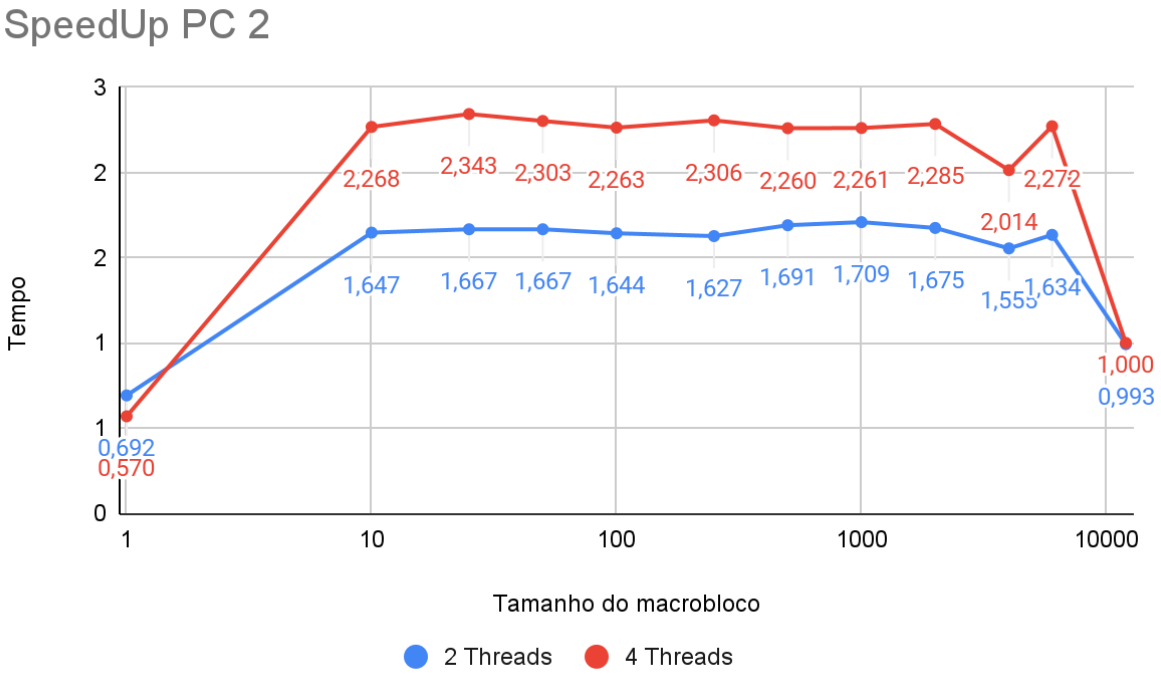

Imagem 2.2: Speedup, tempo serial e tempo multithread do PC 2.

No gráfico do **SpeedUp** foi comparado o tempo serial em relação ao tempo de execução paralelo. quando passamos de 2 threads para 4, temos um speedup de aproximadamente 2x, mas conforme avançamos as quantidades de threads, esse ganho vai diminuindo. A **Lei de Amdahl** estabelece que o speedup máximo que pode ser alcançado por um programa paralelo é limitado pela fração sequencial do programa, que não pode ser paralelizada. Isso significa que, mesmo que você adicione mais threads para paralelizar parte do programa, a presença de partes sequenciais limita a eficácia do paralelismo.

3.1. Macrobloco de tamanho 1 x 1

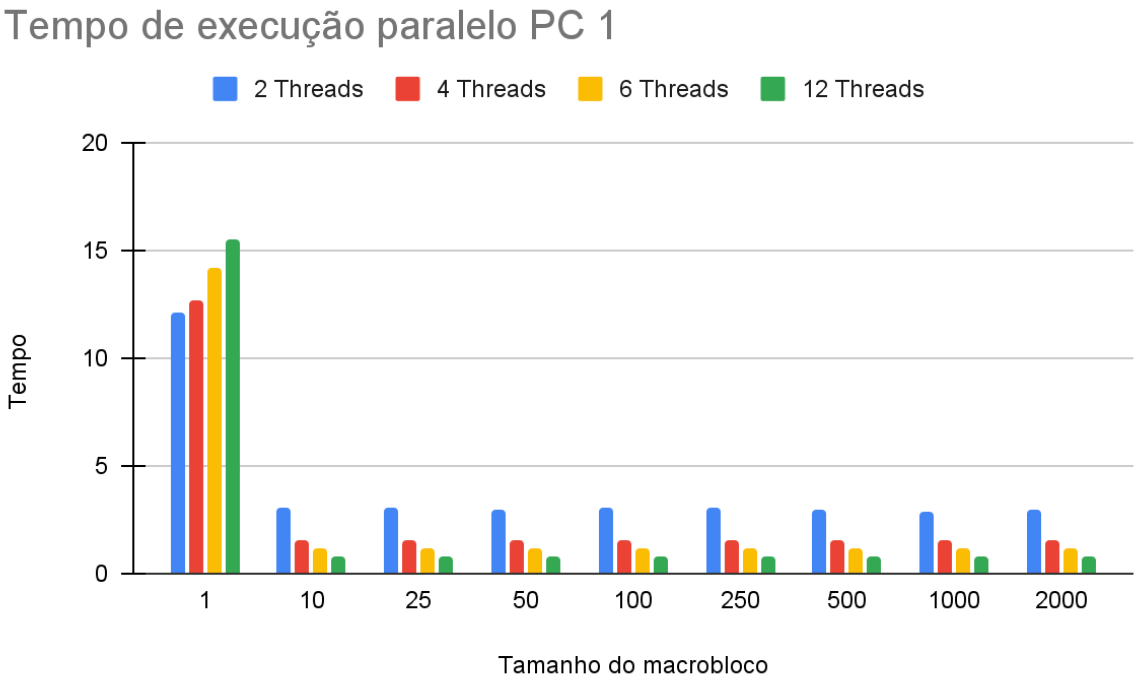

Imagem 3.1: Gráfico com enfoque no tamanho 1 x 1 de macroblocos PC 1.

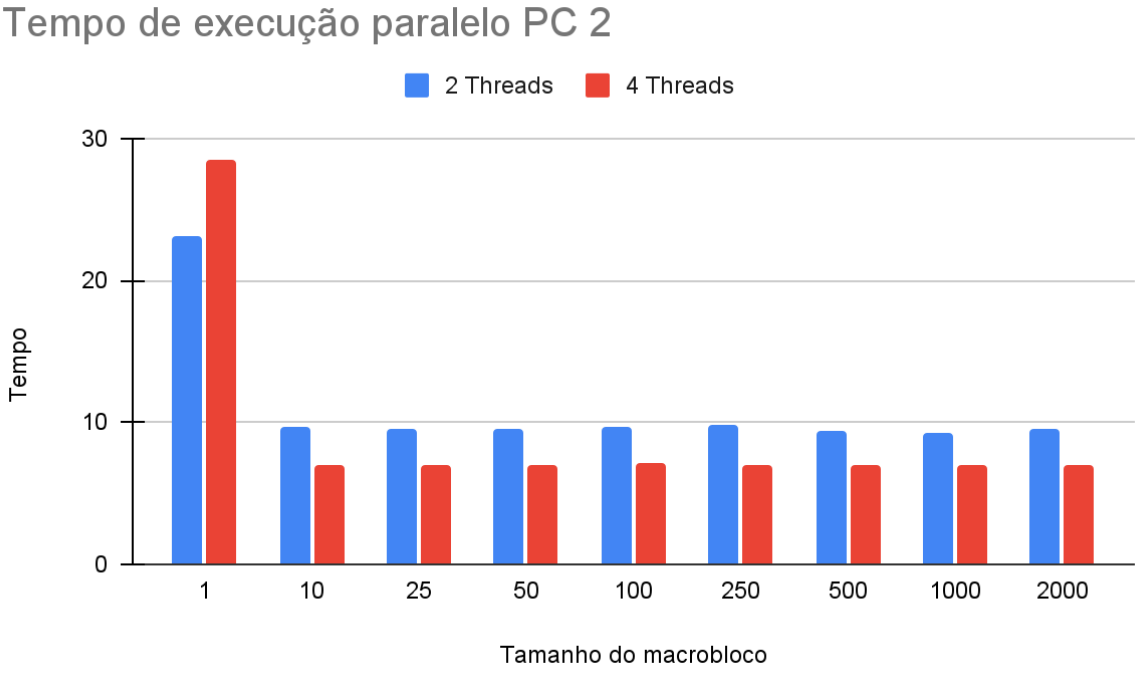

Imagem 3.2: Gráfico com enfoque no tamanho 1 x 1 de macroblocos PC 2.

Ao escolher um tamanho de macrobloco de 1 x 1, o tempo de execução pode ser impactado negativamente devido ao overhead associado ao gerenciamento de threads. Nesse cenário, cada thread é responsável por processar apenas um único elemento da matriz por vez. O tempo de trabalho das threads pode se tornar significativo quando elas dependem do acesso a um mesmo local na memória para manipulação.

A gestão de threads vem com um custo inerente, que pode prevalecer sobre os benefícios da paralelização, especialmente em situações em que o trabalho em cada thread é mínimo. Para alcançar uma paralelização eficiente, muitas vezes é necessário dividir o problema em partes maiores. Dessa forma, cada thread pode realizar uma quantidade mais substancial de trabalho antes de necessitar de sincronização com outras threads.

Nesses casos, a execução sequencial do código pode ser mais eficiente, uma vez que o custo de coordenação entre as threads supera os ganhos obtidos pela execução concorrente.

3.2. Macrobloco de tamanho 12000 x 12000

PC 1

Informações

SERIAL	5,709000 s

Macrobloco	2	4	6	12
1	12,135 s	12,702 s	14,184 s	15,459 s
10	3,032 s	1,593 s	1,130 s	0,786 s
25	3,013 s	1,596 s	1,155 s	0,776 s
50	2,962 s	1,552 s	1,174 s	0,771 s
100	3,015 s	1,580 s	1,157 s	0,767 s
250	3,019 s	1,537 s	1,155 s	0,761 s
500	2,986 s	1,529 s	1,159 s	0,765 s
1000	2,909 s	1,516 s	1,130 s	0,769 s
2000	2,998 s	1,583 s	1,148 s	0,783 s
4000	3,223 s	1,973 s	1,427 s	0,925 s
6000	2,991 s	1,566 s	1,547 s	1,602 s
12000	5,683 s	5,678 s	5,682 s	5,668 s

Tabela 3.1: tempo de processamento x tamanho macrobloco PC 1

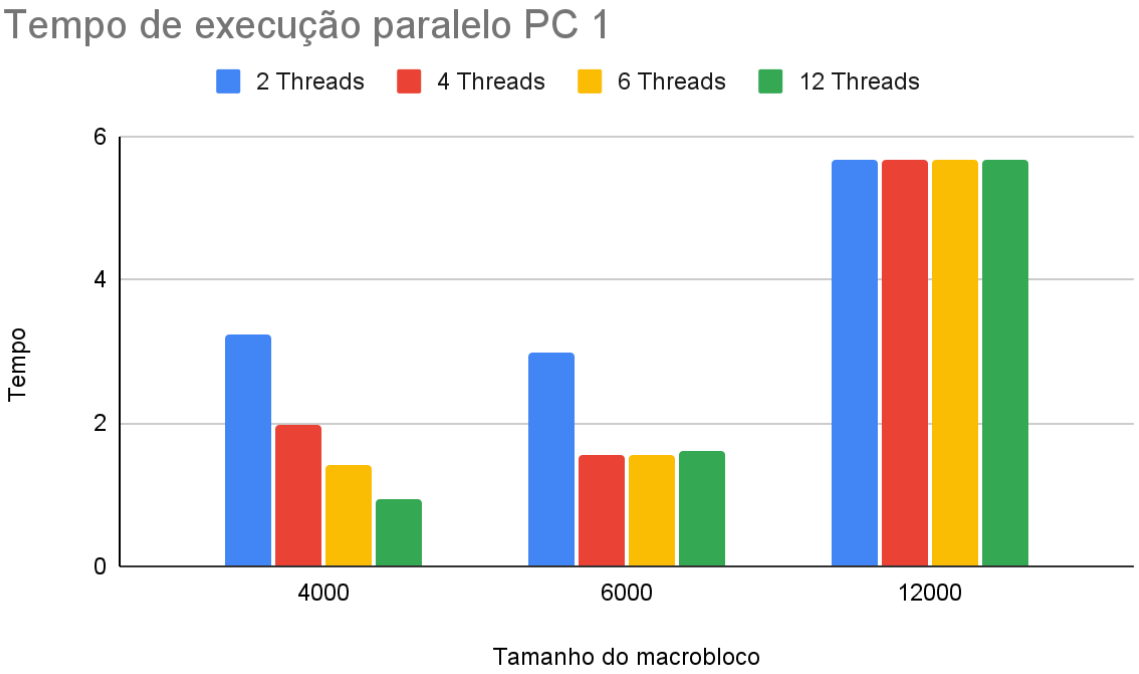

Imagem 4.1: Gráfico do tamanho macrobloco 12000x12000 em relação ao tempo PC 1.

PC 2

SERIAL	16,221 S

	Thread	
Macrobloco	2	4
1	23,060 s	28,480 s
10	9,718 s	7,029 s
25	9,525 s	6,947 s
50	9,517 s	6,973 s
100	9,628 s	7,072 s
250	9,767 s	6,928 s
500	9,450 s	7,039 s
1000	9,289 s	7,017 s
2000	9,536 s	6,969 s
4000	10,182 s	7,960 s
6000	9,766 s	6,997 s
12000	16,036 s	15,830 s

Tabela 3.2: tempo de processamento x tamanho macrobloco PC 2

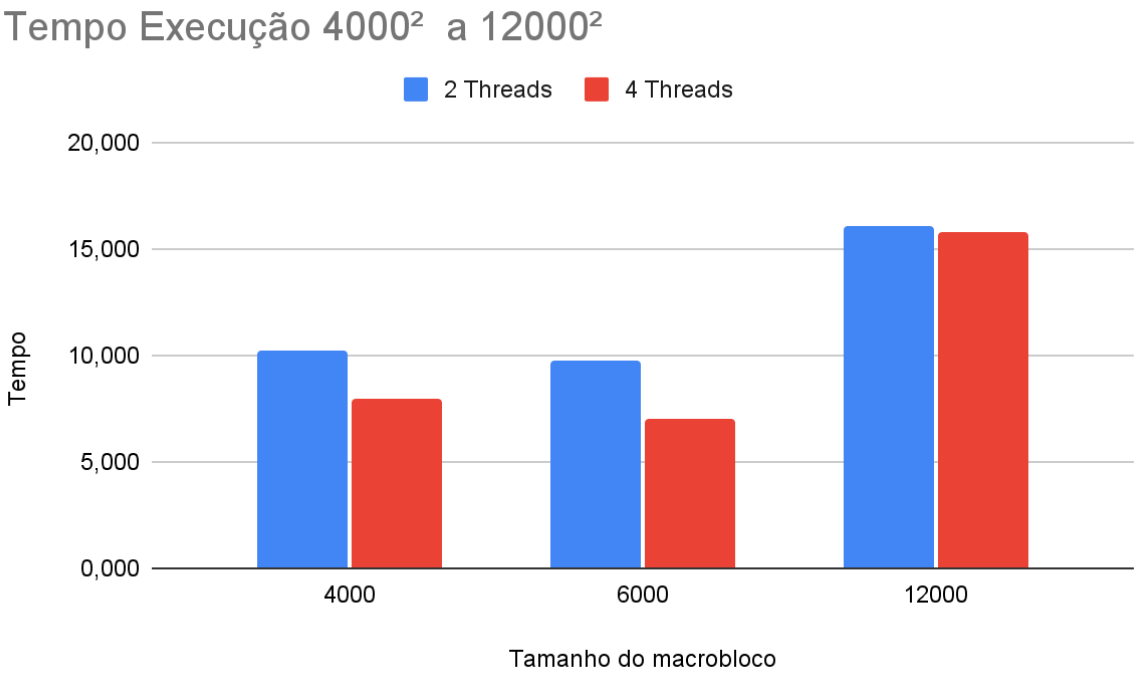

Imagem 4.2: Gráfico do tamanho macrobloco 12000x12000 em relação ao tempo PC 2.

Reforçando as análises anteriores, nas tabelas 3.1 e 3.2, e nas imagens 4.1 e 4.2, quando o tamanho do macrobloco é o mesmo que o tamanho da matriz, ele se torna um caso serial, Nessa configuração apenas uma thread está sendo usada em todo macrobloco.

3.3. Overhead de threads

O aumento de threads em um programa multithread pode resultar em um overhead crítico, devido a diversos fatores. A intensificação da competição por recursos pode ocasionar uma redução na eficiência, à medida que as threads aguardam a disponibilidade de núcleos e o custo do escalonamento aumenta. Em cenários com um grande número de threads disputando recursos compartilhados, como memória, é possível que resulte em bloqueios e diminuição de eficiência do paralelismo. A crescente necessidade de sincronização entre as threads acarreta um alto overhead, enquanto o acesso concorrente à memória pode impactar negativamente o cache hit rate.

Portanto, embora o uso de threads ofereça a possibilidade de explorar o paralelismo e melhorar o desempenho, é crucial encontrar um equilíbrio. A escolha do número ideal de threads depende da natureza específica da aplicação, do hardware subjacente e de estratégias eficientes de programação multithreaded. O aumento indiscriminado do número de threads pode resultar em um overhead crítico que supera os benefícios do paralelismo.

3.4. Remoção dos mutexes

A utilização de mutexes desempenha um papel crucial na programação multithread, impedindo condições de corrida potenciais, nas quais duas ou mais threads procuram acessar ou modificar dados em uma região crítica simultaneamente. Ao remover os mutexes do código-fonte, assumindo um tamanho de macrobloco de 1x1, e permitindo um acesso significativo à memória por parte das threads, observou-se um aumento substancial no tempo de execução do processamento multithread. Isso se deve à ocorrência de condições de corrida, onde a competição pelo acesso a um processo é ignorada, eliminando a espera pela conclusão de uma thread antes de acessá-la.

No entanto, a exclusão dos mutexes impacta negativamente a contagem de números primos. Apesar da ausência de erros no código em si, a busca e o armazenamento dos dados na memória ocorrem em momentos distintos, permitindo potencialmente a alteração do conteúdo da memória durante o intervalo entre essas duas atividades, assim acarretando em uma disrupção da contagem de primos.

4. CONCLUSÃO

Concluindo este trabalho sobre programação paralela, fica claro como a execução simultânea de múltiplas tarefas desempenha um papel crucial na otimização de sistemas computacionais. Ao explorarmos a aplicação prática desse conceito em diversos testes realizados em uma matriz para achar números primos, dividindo-as em macroblocos e designado-as para threads específicas, compreendemos a importância vital do gerenciamento de seções críticas. A programação concorrente emerge como peça-chave, possibilitando a execução simultânea de partes independentes de um programa, proporcionando ganhos no desempenho.

A análise da diferença entre os dois computadores distintos destaca a influência do número de threads no desempenho dos algoritmos. Além disso, exploramos o impacto do tamanho e da quantidade da divisão de processamento da matriz nas métricas de desempenho. Esses testes fornecem valiosos insights sobre como a escolha cuidadosa do número de threads, juntamente com uma gestão eficiente da concorrência, pode positivamente impactar a eficiência de uma aplicação.