Discentes:
Bianca Andrade Galvão
Pedro Lima Almeida Santos
Rafael Athaliba Bomfim Fraga

Docente: Edson Mota

SISTEMAS DISTRIBUÍDOS

RELATÓRIO DE ANÁLISE DE DESEMPENHO:

Programação paralela para a verificação de números primos em um arquivo

Salvador
2025

SUMÁRIO

1. INTRODUÇÃO

O objetivo deste trabalho foi desenvolver um programa para identificar números primos a partir de um arquivo de entrada, utilizando diferentes abordagens de processamento:

- Implementação sequencial (1 thread);
- Implementação paralela com 5 threads;
- Implementação paralela com 10 threads.

Neste relatório será demonstrada a análise de desempenho das versões implementadas, com detalhes sobre a estratégia de implementação, tempos de execução observados e um gráfico comparativo do desempenho das diferentes versões.

2. ESTRATÉGIA DE IMPLEMENTAÇÃO

A versão sequencial processa os números um por um, verificando se cada número é primo antes de gravá-lo no arquivo de saída. Como não há paralelismo, a execução ocorre de forma linear em relação ao número de elementos no arquivo de entrada.

Já na versão com 5 threads, o trabalho foi dividido entre 5 unidades de execução independentes. Cada thread processou uma parte dos números do arquivo de entrada. O objetivo era melhorar a performance aproveitando o paralelismo e a capacidade de múltiplos núcleos de processamento.

Por fim, na versão com 10 threads, o número de threads foi aumentado para maximizar o uso de múltiplos núcleos do processador. Assim como na versão com 5 threads, o trabalho foi dividido entre as threads, e a expectativa era de um desempenho consideravelmente mais rápido em comparação com a versão sequencial.

3. ANÁLISE DOS RESULTADOS

A execução sequencial foi a mais lenta das três versões, com média de 90 milissegundos, em dez execuções, já que não há divisão do trabalho entre múltiplas threads. A versão com 5 threads teve um desempenho melhor que a versão sequencial, apresentando média de 61 milissegundos. Já a versão com 10 threads, teve o melhor desempenho entre todas as versões, com um tempo de 55,5 milissegundos, apresentando a menor quantidade de tempo de execução. No entanto, a diferença de tempo entre as versões de 10 threads e 5 threads foi mínima devido a alguns fatores que podem ter influenciado diretamente nessa observação.

O primeiro deles é a Sobrecarga de Gerenciamento de Threads, à medida que o número de threads aumenta, também aumenta a sobrecarga associada ao gerenciamento dessas threads. Criar e gerenciar múltiplas threads requer recursos adicionais do sistema operacional, como agendamento e comunicação entre threads. Se a quantidade de trabalho que pode ser paralelizado não for suficientemente grande, a sobrecarga pode superar os ganhos (benefícios) do paralelismo.

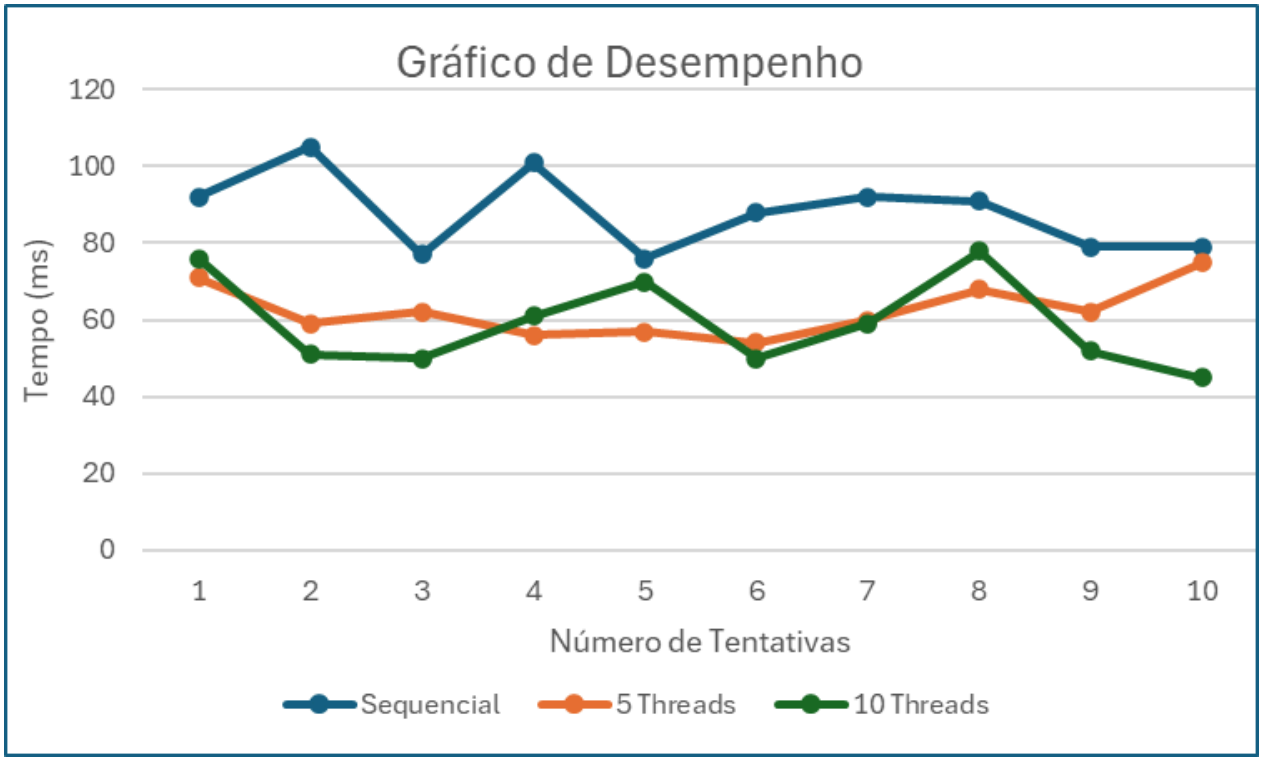
Outro fator importante é a Limitação do Sistema de Hardware, se o número de núcleos disponíveis no processador for limitado, adicionar mais threads além do número de núcleos disponíveis pode não resultar em ganhos significativos. Em muitos sistemas, o aumento de threads acima do número de núcleos físicos pode levar a uma competição mais intensa pelos recursos de CPU, o que, em alguns casos, pode até causar uma leve degradação do desempenho devido ao custo de alternância de contexto.

A Granularidade do Trabalho também deve ser considerada, no caso específico da verificação de primalidade, o trabalho a ser distribuído entre as threads pode ser relativamente pequeno. Ou seja, mesmo que tenhamos mais threads, a quantidade de trabalho para cada thread pode não ser suficiente para justificar uma melhora significativa no tempo de execução.

Por fim, a verificação de números primos pode não ser tão intensiva em termos de CPU para justificar uma divisão em um grande número de threads. Em tarefas com maior complexidade computacional ou onde o tempo de processamento por unidade de trabalho é maior, o ganho de performance com mais threads seria mais evidente.

4. GRÁFICO COMPARATIVO

Gráfico 1

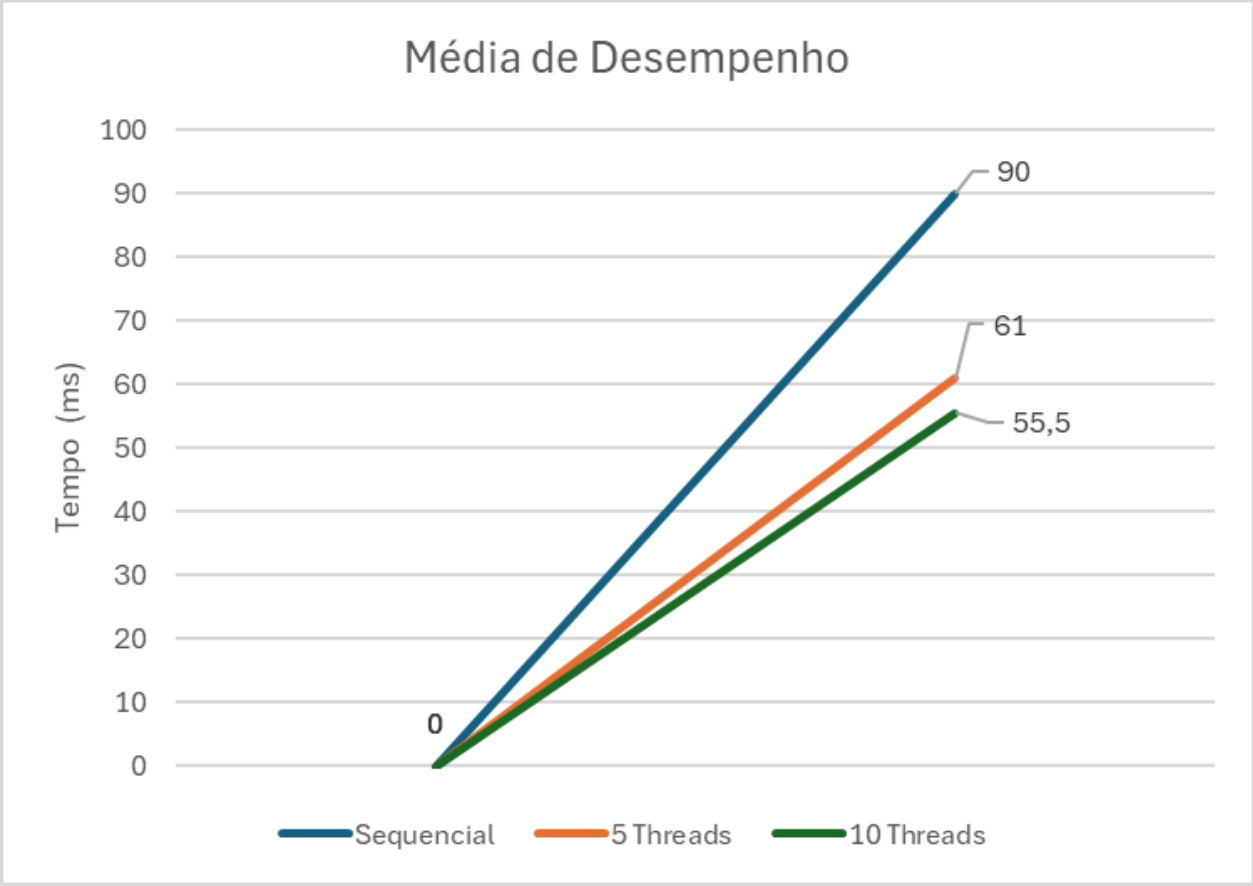


Fonte: Autoral

O gráfico acima (Gráfico 1) apresenta a análise de três testes de desempenho considerando a execução sequencial, com 5 threads e com 10 threads, ao longo de 10 amostras. A métrica avaliada foi o tempo de execução.

Os resultados mostram que a execução sequencial obteve um tempo médio de 90ms, enquanto a execução com 5 threads reduziu esse tempo para 61ms. Já com 10 threads, a média foi significativamente menor, atingindo 55,5ms. A imagem abaixo (Gráfico 2) ilustra esse comparativo, destacando a variação do tempo desde o marco inicial até a conclusão dos testes.

Gráfico 2



Fonte: Autoral

5. CONCLUSÃO

A análise de desempenho das versões sequencial, com 5 threads e com 10 threads revelou que a paralelização traz benefícios, mas especialmente quando o número de threads ultrapassa o número de núcleos disponíveis ou quando a tarefa não é suficientemente intensiva, os ganhos diminuem à medida que o número de threads aumenta.

Embora a versão com 10 threads tenha sido a mais rápida, a diferença em relação à versão com 5 threads foi bem pouca. Esse fenômeno pode ser explicado pela sobrecarga de gerenciamento de threads e pela limitação de recursos do sistema, que impede que o ganho de paralelismo seja tão expressivo.

Portanto, a escolha do número ideal de threads depende da natureza da tarefa e das especificações do hardware, e em alguns casos, uma versão com menos threads pode ser mais eficiente devido a uma sobrecarga reduzida no gerenciamento de threads.