Relatório de Análise de Desempenho: Algoritmo Paralelo para Contagem de Números Primos

Hélio Peres Martins Neto

22 de setembro de 2025

Resumo

Este relatório detalha a análise de desempenho de um algoritmo para contagem de números primos em uma matriz, comparando uma implementação serial com uma implementação paralela baseada em múltiplas threads. O objetivo é avaliar o ganho de performance (Speedup) e a eficiência do paralelismo sob diferentes configurações de número de threads e granularidade de tarefas. Os experimentos foram conduzidos em um ambiente controlado para garantir a fidedignidade dos resultados. A análise revelou um ganho de performance máximo de 3.5x, com resultados notáveis sobre o comportamento do escalonador do sistema operacional em cenários de supersaturação de threads.

1 Introdução

A programação concorrente é um pilar da computação moderna, permitindo a execução de tarefas complexas em uma fração do tempo através da utilização de múltiplos núcleos de processamento. Este trabalho implementa e avalia uma solução paralela para o problema de contagem de números primos em uma matriz de grandes dimensões, um problema inerentemente paralelizável e ideal para o estudo de conceitos como seção crítica, divisão de trabalho e overhead de paralelismo.

2 Metodologia

Para garantir a consistência e a reprodutibilidade dos resultados, foi definida uma metodologia rigorosa, detalhando o ambiente de hardware e as configurações da simulação.

2.1 Ambiente de Teste

Todos os experimentos foram executados na mesma máquina, cujas especificações estão listadas abaixo:

- **CPU:** Intel Core i5-2500 @ 3.30GHz
- **Núcleos Físicos / Lógicos:** 4 / 4 (sem Hyper-Threading)

- **Memória RAM:** 16 GB
- **Sistema Operacional:** Windows 10 Pro
- **Processos em Execução:** Navegador Edge (5 abas), Visual Studio Code, Gerenciador de Tarefas.

2.2 Configuração da Simulação

Os parâmetros a seguir foram mantidos fixos durante toda a campanha de testes para assegurar uma base de comparação justa.

- **Dimensões da Matriz:** 8000 x 8000 elementos.
- **Semente Aleatória:** 123456789.
- **Consumo de Memória:** A matriz consumiu aproximadamente 256 MB, um valor seguro para os 16 GB de RAM disponíveis.
- **Justificativa da Configuração:** A escolha por uma matriz de dimensões 8000x8000 atende ao requisito de desempenho definido no projeto, o qual especifica um tempo de execução para o algoritmo serial superior a 40 segundos.

2.3 Execução de Referência (Single Thread)

A execução serial do algoritmo serve como nossa linha de base para todos os cálculos de performance.

- **Tempo de Execução Serial (T_S):** 57748 ms.
- **Resultado (Primos Encontrados):** 4.252.198. Este valor serve como gabarito para validar a correção das execuções paralelas.

3 Resultados

Foram conduzidos dois experimentos principais: o primeiro variando o número de threads com um tamanho de tarefa fixo, e o segundo variando o tamanho da tarefa com um número de threads fixo.

3.1 Experimento A: Análise de Escalabilidade por Threads

Neste teste, o tamanho da subtarefa (chunk) foi fixado em 100 linhas, enquanto o número de threads foi variado para medir o impacto no Speedup e na Eficiência. Os resultados estão compilados na Tabela 1 e visualizados nas Figuras 1 e 2.

Tabela 1: Resultados da variação do número de threads (Chunk Fixo = 100).

Nº Threads	**Tempo (ms)**	**Resultado**	**Speedup**	**Eficiência (%)**
2	41505	4252198	1.39x	69.57%
4	22603	4252198	2.55x	63.87%
6	19822	4252198	2.91x	48.56%
8	18205	4252198	3.17x	39.65%
10	18081	4252198	3.19x	31.94%
16	16469	4252198	3.51x	21.92%
32	19099	4252198	3.02x	9.45%

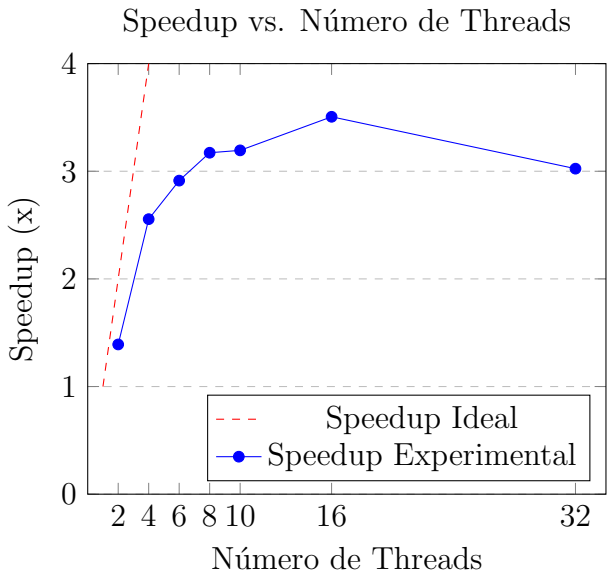

Figura 1: Ganho de performance em relação à execução serial.

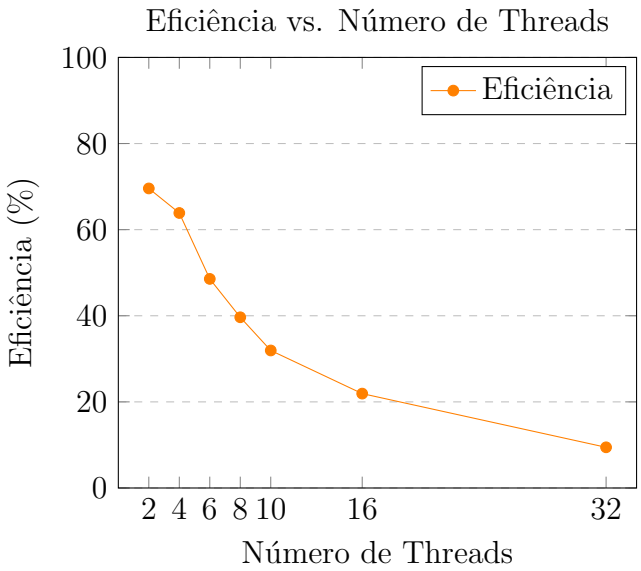

Figura 2: Percentual de utilização dos recursos de processamento.

3.2 Experimento B: Análise de Sensibilidade à Granularidade

Com base no resultado anterior, o número de threads foi fixado em 16 (o ponto de melhor desempenho) para avaliar o impacto do tamanho do chunk no tempo de execução.

Tabela 2: Resultados da variação do tamanho do chunk (Threads Fixo = 16).

Tamanho do Chunk	**Tempo (ms)**
1	17847
10	18470
50	17938
100	16785
250	18523
500	18017
1000	18207
2500	25347

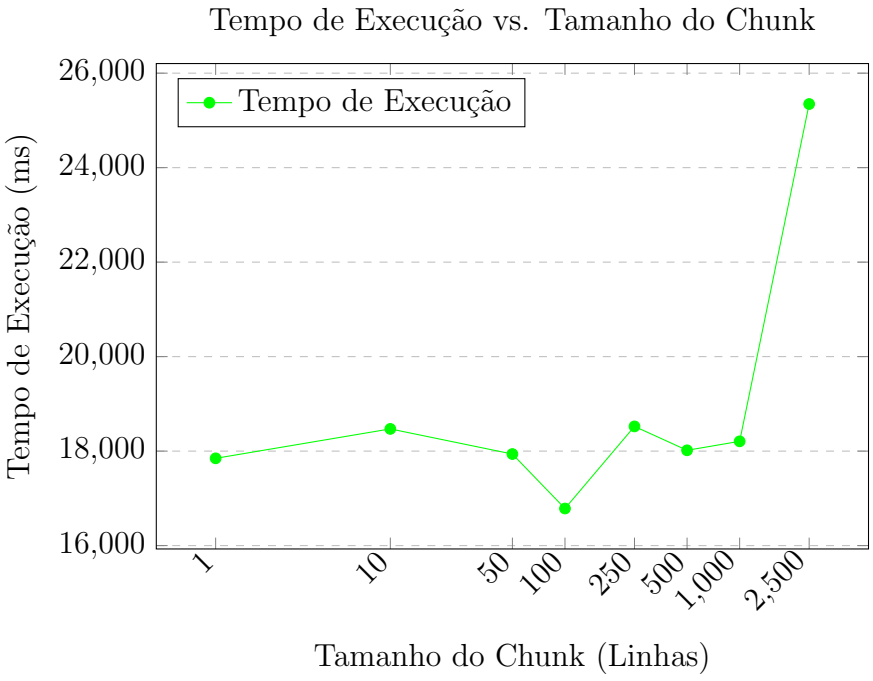

Figura 3: Impacto da granularidade da tarefa no desempenho.

4 Análise e Discussão

A análise dos dados coletados revela insights importantes sobre a dinâmica da computação paralela neste cenário específico. Primeiramente, a consistência do resultado final em todas as execuções (‘4.252.198‘) valida a correção da implementação concorrente, indicando que os mecanismos de sincronização foram eficazes em prevenir condições de corrida.

O resultado mais notável do Experimento A foi o pico de performance ter ocorrido com **16 threads** em uma CPU de 4 núcleos. A teoria convencional sugere que o ótimo seria com 4 threads. Este comportamento pode ser atribuído à capacidade do escalonador

do Sistema Operacional de "esconder a latência". Com um número elevado de threads prontas para executar, o SO pode alternar o contexto rapidamente, garantindo que os núcleos da CPU permaneçam produtivos mesmo durante micro-paradas de uma thread (ex: cache miss). No entanto, como visto na Figura 1, a performance decai com 32 threads, quando o custo do chaveamento de contexto supera os benefícios.

O Experimento B (Figura 3) validou a hipótese da "curva em U". Chunks muito pequenos resultaram em tempos maiores devido ao alto **overhead** de gerenciamento, com as threads disputando o acesso à seção crítica da fila de tarefas. Chunks muito grandes, por outro lado, levaram a um severo **desbalanceamento de carga**, onde threads que recebiam tarefas computacionalmente mais leves ficavam ociosas esperando a conclusão da thread mais sobrecarregada. O ponto ótimo foi encontrado em um chunk de tamanho 100.

5 Conclusão

O estudo demonstrou com sucesso a eficácia da paralelização para o problema da contagem de números primos, alcançando um **speedup máximo de 3.51x**. A análise revelou que o desempenho ótimo não depende apenas do número de núcleos físicos, mas também da interação complexa entre o algoritmo, a granularidade das tarefas e o escalonador do sistema operacional. A configuração de melhor desempenho foi identificada com **16 threads** e um **tamanho de chunk de 100 linhas**, uma conclusão não trivial que ressalta a importância da experimentação empírica na otimização de software concorrente.