Paralelização da Contagem de Números Primos

Um Estudo sobre Condições de Corrida e Speedup com OpenMP

Werbert Arles de Souza Barradas

Universidade Federal do Rio Grande do Norte (UFRN)
Disciplina de Programação Paralela - DCA3703

20 de agosto de 2025

Introdução

O Desafio da Contagem de Primos

A contagem de números primos em um grande intervalo é uma tarefa **computacionalmente intensiva**, ideal para testar os limites do paralelismo.

Execução Sequencial

Serve como nossa **linha de base (baseline)** para medir a corretude e o desempenho.

Execução Paralela

Busca reduzir drasticamente o tempo de execução, mas introduz desafios de sincronização.

Objetivo do Estudo

Analisar o ganho de desempenho (*speedup*) e demonstrar experimentalmente o problema da **condição de corrida** e sua solução em OpenMP.

Metodologia: As Três Implementações

1. Versão Sequencial (Baseline)

```
int total_primes_seq = 0;
start_time = omp_get_wtime();

for (int i = 2; i <= n; i++) {
    if (is_prime(i)) {
        total_primes_seq++;
    }
}
```

Metodologia: As Três Implementações

2. Versão Paralela Ingênua

```
    int total_primes_par = 0;
    start_time = omp_get_wtime();

    // A diretiva #pragma omp parallel for divide as
itera  es do la o
    // entre as threads dispon veis.
    #pragma omp parallel for
    for (int i = 2; i <= n; i++) {
        if (is_prime(i)) {
            // CUIDADO: Esta linha causa um problema!
            total_primes_par++;
        }
    }

    end_time = omp_get_wtime();
```

Metodologia: As Três Implementações

3. Versão Paralela Corrigida

```
    total_primes_par = 0; // Reinicia a contagem
    start_time = omp_get_wtime();
    // Usei a clausula 'reduction' para evitar a
condicao de corrida.
    // Cada thread tera sua propria copia local de '
total_primes_par'.
    // No final, o OpenMP soma os valores de todas as
copias locais.
    #pragma omp parallel for reduction(+:
total_primes_par)
    for (int i = 2; i <= n; i++) {
        if (is_prime(i)) {
            total_primes_par++;
        }
    }

end_time = omp_get_wtime();
```

Por que a Versão Ingênua Falha?

A Operação Não-Atômica

A instrução ‘x++‘ não é uma única operação. O processador a executa em três passos:

1. **Ler** o valor de ‘x‘ da memória.
2. **Incrementar** o valor no registrador.
3. **Escrever** o novo valor de volta na memória.

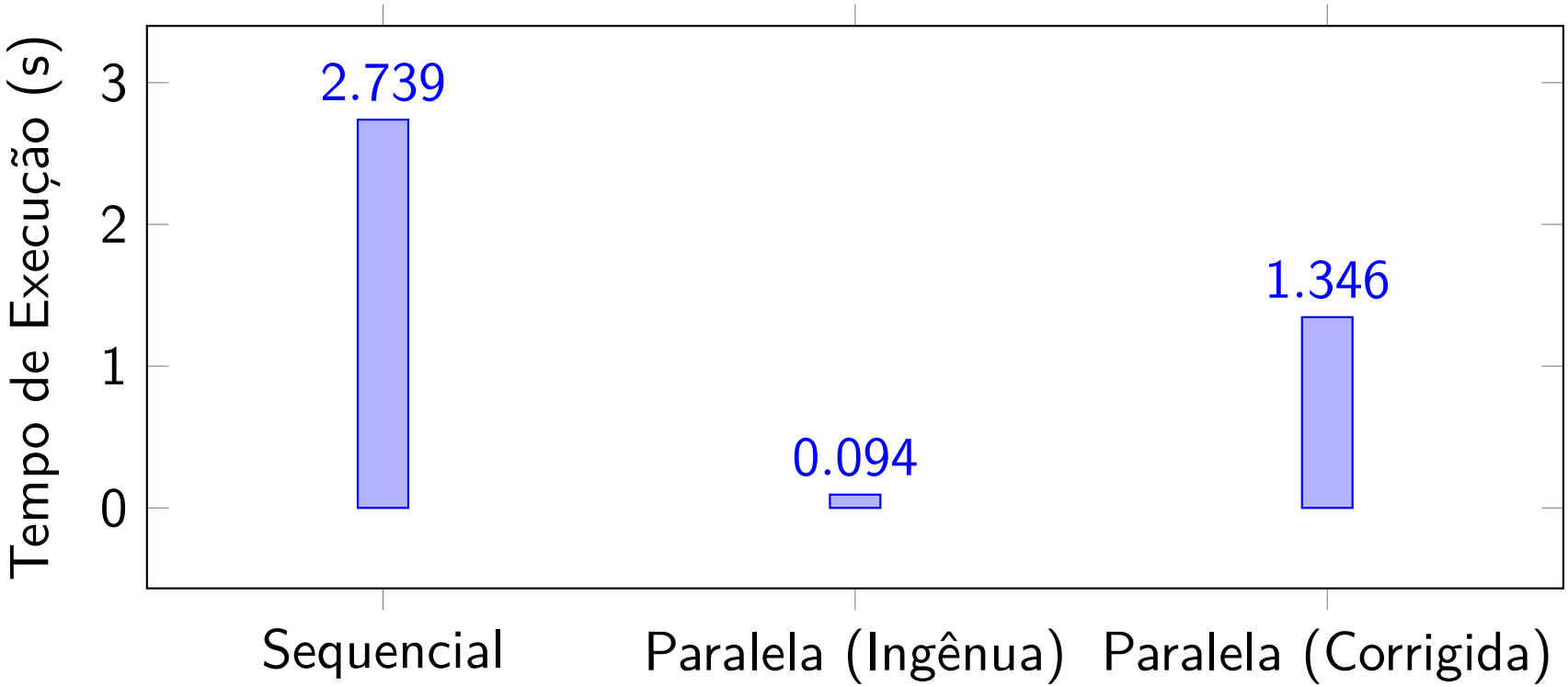
Múltiplas threads podem executar o passo 1 antes que qualquer uma chegue ao passo 3.

Cenário de Conflito

- Thread A **lê** ‘total‘ (ex: 100).
- Thread B **lê** ‘total‘ (ainda 100).
- Thread A **escreve** ‘101‘.
- Thread B **escreve** ‘101‘.

Resultado: Dois primos foram encontrados, mas o contador foi incrementado apenas uma vez.

Comparativo de Tempo de Execução para N = 20.000.000



Análise dos Resultados

Análise de Correção

A versão **Paralela (Ingênua)** é a mais rápida, mas produz um resultado incorreto (1.239.634 primos), provando o efeito da condição de corrida. Somente as versões **Sequencial** e **Paralela (Corrigida)** chegam ao valor correto de 1.270.607 primos.

Análise de Desempenho (Speedup)

Comparando as duas versões corretas:

$$S = \frac{T_{sequencial}}{T_{paralelo}} = \frac{2.739s}{1.346s} \approx 2.04$$

A paralelização correta resultou em um programa **2.04 vezes mais rápido**.

Conclusão

Resultados do Estudo

O experimento demonstrou claramente tanto o potencial de ganho de desempenho do OpenMP quanto os riscos da programação concorrente.

- A paralelização do problema, quando feita corretamente, gerou um **speedup significativo** de 2.04x.
- O acesso não sincronizado a recursos compartilhados leva a **resultados incorretos** e inconsistentes, mesmo que o programa pareça executar mais rápido.

Implicação Prática

Compreender e aplicar mecanismos de sincronização, como a cláusula ‘reduction‘, é **fundamental** para o desenvolvimento de software paralelo que seja não apenas rápido, mas também correto.