Universidade Federal do Rio Grande do Norte
Departamento de Engenharia da Computação e Automação
DCA3703 - Programação Paralela
Tarefa 5 - Comparação entre programação sequencial e paralela
Aluno: Daniel Bruno Trindade da Silva

1 Introdução

Nesta prática, buscamos comparar a programação paralela com a sequencial, para isso desenvolvemos um programa capaz de contar quantos números primos existem entre 2 e um dado n e o implementamos em uma versão sequencial e outra paralela, mantendo a lógica original do programa para garantir uma comparação justa de desempenho. Por fim comparamos o desempenho das versões do código para entender o impacto da paralelização no tempo de execução.

2 Metodologia

Para a versão de execução sequencial, implementamos o código com uma função chamada `ehPrimo()` que recebe um número e retorna `true` caso seja primo, `false` caso não:

```
bool ehPrimo(int num) {
    if (num < 2) return false;
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) return false;
    }
    return true;
}
```

Na `main()` foi implementado um laço de repetição que passa por todos os inteiros de 2 a n testando o valor com a função `ehPrimo()`. Nesse caso temos dois laços de repetição aninhados que testam os números um a um.

```
int main() {
    int contador = 0;
    struct timeval inicio, fim;
    double time_lapsed;
    gettimeofday(&inicio, NULL);
    for (int i = 2; i <= N; i++) {
        if (ehPrimo(i)) {
            contador++;
        }
    }

    gettimeofday(&fim, NULL);
    time_lapsed = get_time(inicio, fim);
    printf("Quantidade de números primos eh: %d\n", contador);
    printf("Tempo gasto: %f segundos\n", time_lapsed);

    return 0;
}
```

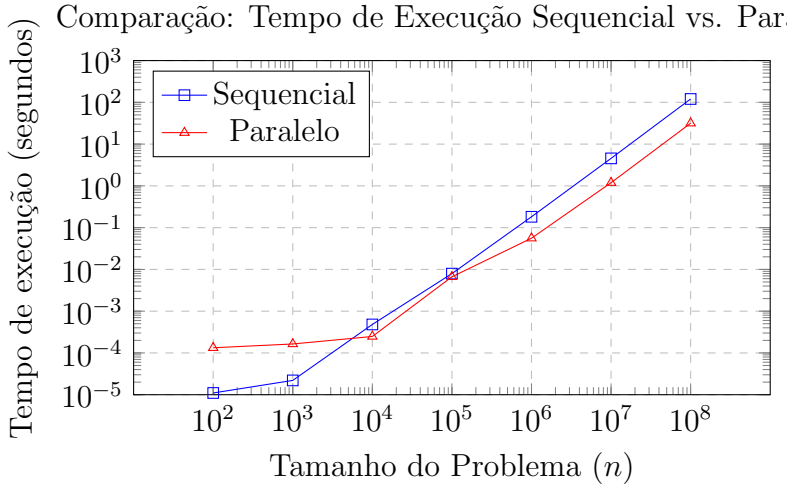
Para a versão do código paralela, mantivemos a lógica da versão sequencial, porém acrescentamos a diretiva `#pragma omp parallel for` que paraleliza a execução do laço de repetição, distribuindo entre os threads disponíveis. Devido a condição de corrida estabelecida pelo paralelismo do laço precisamos adicionar uma área crítica na variável contador, e nesse caso adicionamos a diretiva `reduction(+:contador)` que protege a variável desse problema. CCom as alterações necessárias, a função `main()` ficou da seguinte forma:

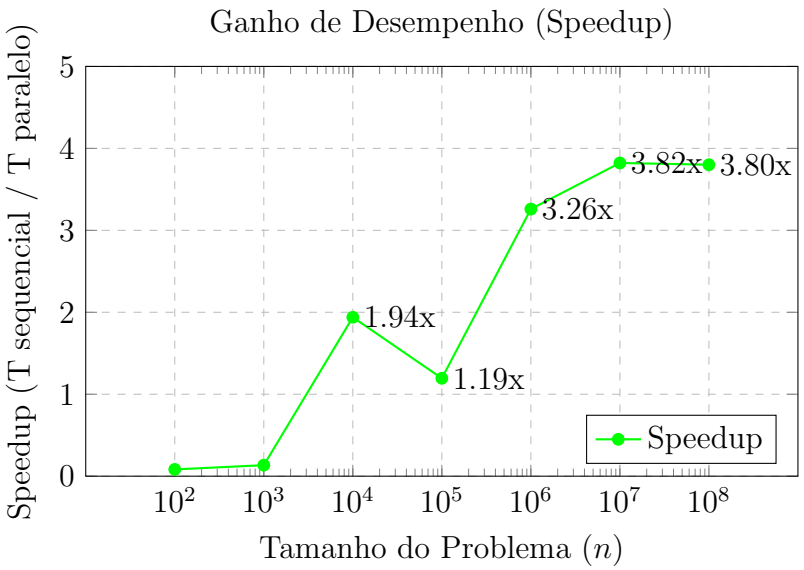
```
int main() {
    int contador = 0;
    struct timeval inicio, fim;
    double time_lapsed;
    gettimeofday(&inicio, NULL);

    #pragma omp parallel for reduction(+:contador)

    for (int i = 2; i <= N; i++) {
        if (ehPrimo(i)) {
            contador++;
        }
    }

    gettimeofday(&fim, NULL);
    time_lapsed = get_time(inicio, fim);
    printf("Quantidade de números primos eh: %d\n", contador);
    printf("Tempo gasto: %f segundos\n", time_lapsed);

    return 0;
}
```

Em ambas as versões o tempo de execução foi calculado utilizando a função `gettimeofday()` da biblioteca `sys/time.h` para que possamos comparar seus resultados.

3 Resultados

Vamos realizar a comparação dos tempos de execução de cada uma das versões do código:





Com isso, podemos observar que, para valores pequenos de n (até 10.000), a versão paralela apresenta desempenho igual ou inferior ao da versão sequencial. Isso ocorre porque a criação, gerenciamento e sincronização das threads introduzem um custo computacional adicional, conhecido como *overhead*, que não compensa o paralelismo nesse intervalo. No entanto, para valores maiores (acima de 100.000), a versão paralela começa a superar a execução sequencial, alcançando uma melhoria de desempenho de até 3,8 vezes na execução da tarefa.

4 Conclusão

A comparação entre as abordagens sequencial e paralela demonstrou que, para tarefas computacionalmente simples ou com baixo volume de dados, a paralelização pode não trazer benefícios devido ao overhead de gerenciamento das threads. Entretanto, conforme o tamanho do problema aumenta, os ganhos de desempenho se tornam significativos, alcançando um speedup de até 3,8 vezes em nossa análise. Essa prática evidencia a importância da escolha criteriosa da abordagem de programação, considerando o custo-benefício do paralelismo.