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 paralelizada de forma que não foi alterada a lógica do programa. 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 implementada na 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. Com as alterações necessárias a `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 o método `gettimeofday` da biblioteca `sys/time.h` para que possamos comparar seus resultados.

3 Resultados

Sequencial

n - Tempo de Execução

100 - 0,000011 1000 - 0,000183 10000 - 0,002914 100000 - 0,057376 1000000 - 0,292285 10000000 - 5,107619 100000000 - 140,143158

Paralelo com 4 threads

n - Tempo de Execução

100 - 0,000011 1000 - 0,000142 10000 - 0,001346 100000 - 0,050420 1000000 - 0,271619 10000000 - 4,971940 100000000 - 142,141867