Universidade Federal do Rio Grande do Norte
Departamento de Engenharia da Computação e Automação
DCA3703 - Programação Paralela
Tarefa 4 - Aplicações limitadas por memória ou CPU
Aluno: Daniel Bruno Trindade da Silva

1 Introdução

A computação paralela tem se tornado uma estratégia essencial para melhorar o desempenho de programas, especialmente com a popularização de processadores com múltiplos núcleos. O OpenMP (Open Multi-Processing) é uma API amplamente utilizada para programação paralela em sistemas com memória compartilhada, permitindo a distribuição de tarefas entre múltiplas threads de forma simples e eficiente. Este relatório apresenta a implementação e análise de dois programas paralelos desenvolvidos em linguagem C com OpenMP: um limitado por memória (memory-bound), baseado em somas simples entre vetores, e outro limitado por processamento (compute-bound), com operações matemáticas intensivas.

O objetivo principal é avaliar o impacto da paralelização com a diretiva `#pragma omp parallel for` no tempo de execução dos programas, variando o número de threads utilizadas. A partir dos experimentos, são observadas as situações em que o desempenho melhora, se estabiliza ou até mesmo piora, permitindo uma reflexão sobre os efeitos do multithreading de hardware em diferentes cenários. Essa análise contribui para a compreensão dos limites e vantagens da programação paralela, considerando o perfil computacional das aplicações.

2 Metodologia

Realizamos a implementação de dois programas em C, o primeiro deles tem como objetivo realizar somas simples para simular uma carga de trabalho limitada pelo acesso a memória (memory-bound). Ele aloca dinamicamente um vetor de 100 milhões de elementos do tipo double e realiza uma operação simples de escrita: atribuir a cada posição do vetor o valor i + 2, onde i é o índice da iteração. A paralelização é feita com a diretiva `#pragma omp parallel` for, permitindo que múltiplas threads realizem essas atribuições em paralelo.

O tempo de execução é medido com a função `omp_get_wtime()`, antes e depois do laço paralelo. Como a operação executada em cada iteração é computacionalmente simples, o desempenho do programa depende principalmente da taxa de leitura e escrita na memória principal. Esse tipo de programa evidencia os limites do ganho de desempenho com o aumento do número de threads, já que o gargalo está na largura de banda da memória, e não na capacidade de cálculo da CPU.

Assim ficou nosso código:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define SIZE 100000000

int main() {

 double *a = (double *)malloc(SIZE * sizeof(double));
 double start, end;
```

```
  start = omp_get_wtime();
  #pragma omp parallel for
  for (int i = 0; i < SIZE; i++) {
   a[i] = i+2;
  }
  end = omp_get_wtime();

  printf("Tempo de execução (Memory-bound): %f segundos\n", end - start);

  free(a);

  return 0;
}
```

Para execução desse programa memory bound utilizamos a diretiva `OMP_NUM_THREADS=1` alterando a quantidade de treads progressivamente e observando o comportamento

Programa CPU-bound O segundo programa desenvolvido simula uma carga de trabalho limitada pela capacidade de processamento da CPU (CPU-bound). Ele realiza 100 milhões de operações matemáticas intensivas, envolvendo funções como sin, cos e sqrt, seguidas de uma divisão. Essas operações são realizadas dentro de um laço for paralelo, utilizando a diretiva `#pragma omp parallel for` com uma cláusula de reduction para somar corretamente os resultados parciais de cada thread na variável result.

O tempo de execução também é medido com `omp_get_wtime()`. Diferente do programa memory-bound, aqui o gargalo está no tempo necessário para realizar os cálculos matemáticos complexos, que demandam mais ciclos de CPU. Esse tipo de programa permite observar como o desempenho pode escalar com múltiplas threads até certo ponto, mas também evidencia como o aumento de concorrência pode gerar competição por recursos internos da CPU, como unidades de ponto flutuante e cache, o que pode limitar os ganhos ou até causar perda de desempenho.