Universidade Federal do Rio Grande do Norte - UFRN
Departamento de Engenharia da Computação e Automação - DCA
DCA3703 - Programação Paralela

Tarefa 04: Aplicações limitadas por memória ou CPU

Discente: Quelita Míriam
Docente: Samuel Xavier de Souza

I. INTRODUÇÃO

Este relatório tem como objetivo analisar o impacto da paralelização com OpenMP sobre dois aspectos em um programa: limitação de desempenho por acesso à memória (*memory-bound*) e limitação por carga computacional (*compute-bound*). A motivação central é compreender como o número de threads influencia o tempo de execução desse programa, além de explorar as consequências da utilização de *multithreading* por *hardware* em arquiteturas modernas de processadores.

Este estudo serve como exemplo prático do *gargalo de Von Neumann*, que ocorre quando o desempenho de um programa é limitado pela velocidade com que os dados são transferidos entre a memória e a CPU. Isso acontece porque, na maioria dos computadores, memória e processador compartilham o mesmo caminho para troca de informações. Em programas que acessam muita memória, como o de somas vetoriais (*memory-bound*), mesmo com vários núcleos trabalhando em paralelo, o tempo gasto esperando dados da memória pode se tornar o principal limitador de desempenho. Já em programas *compute-bound*, que exigem muito cálculo, o gargalo está mais relacionado à capacidade de processamento da CPU do que ao acesso à memória.

II. METODOLOGIA

Utilizando a biblioteca OpenMP, foi desenvolvido um programa[1] em C para explorar *memory-bound*, baseado em operações simples de soma entre vetores, e outro *compute-bound*, com cálculos matemáticos mais intensivos utilizando funções como *sin* e *cos*. Ambos foram implementados em um único arquivo e paralelizados com a diretiva "*#pragma omp parallel for*". A medição do tempo de execução de cada trecho foi feita com a função "*omp_get_wtime()*", apropriada para esse tipo de experimento por fornecer tempo de parede com boa precisão e integração nativa com OpenMP.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#include <sys/time.h>

#define N 10000000
#define ITERATIONS 10000000

double get_time() {
```

[1] Código fonte para a tarefa em: Github: Quelita2 – <https://github.com/quelita2/programacao-paralela/blob/main/topico01/tarefa04/src/main.c>

```
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec + tv.tv_usec / 1e6;
}

void memory_bound(int num_threads) {
    double *a = (double *)malloc(N * sizeof(double));
    double *b = (double *)malloc(N * sizeof(double));
    double *c = (double *)malloc(N * sizeof(double));

    for (int i = 0; i < N; i++) {
        a[i] = i * 1.0;
        b[i] = (i + 1) * 1.0;
    }

    double start = get_time();

    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        c[i] = a[i] + b[i];
    }

    double end = get_time();
    printf("Memory-bound (Threads: %d) -> Tempo: %f s\n", num_threads, end - start);

    free(a);
    free(b);
    free(c);
}

void compute_bound(int num_threads) {
    double result = 0.0;
    double start = get_time();

    #pragma omp parallel for
    for (int i = 0; i < ITERATIONS; i++) {
        result += sin(i) * cos(i) / (sqrt(i + 1.0));
    }

    double end = get_time();
    printf("Compute-bound (Threads: %d) -> Tempo: %f s\n", num_threads, end - start);

    printf("Resultado final (ignore): %f\n", result);
  }

int main() {
    int thread_counts[] = {1, 2, 4, 8, 12};
    int num_tests = sizeof(thread_counts) / sizeof(thread_counts[0]);

    for (int i = 0; i < num_tests; i++) {
        omp_set_num_threads(thread_counts[i]);

        memory_bound(thread_counts[i]);
        compute_bound(thread_counts[i]);
```

```
        printf("----------------------------------------\n");
    }

    return 0;
}
```

Os experimentos foram executados em um sistema com 4 núcleos físicos e suporte a *hyper-threading*, totalizando 8 *threads* lógicas disponíveis. Ainda assim, foram realizadas execuções com até 12 *threads*, o que é possível graças à capacidade do OpenMP de criar mais *threads* do que a quantidade de unidades de processamento disponíveis. Nesse cenário, o sistema operacional distribui as 12 *threads* entre os 8 *threads* lógicos através de escalonamento, intercalando sua execução. Esse comportamento permite avaliar como o desempenho se comporta em condições de saturação de recursos, especialmente em programas *compute-bound*, nos quais o excesso de *threads* pode provocar competição interna por cache, registradores e unidades de execução da CPU.

As execuções foram feitas variando o número de *threads* (1, 2, 4, 8 e 12), permitindo observar como o desempenho se altera conforme a carga paralela aumenta. A análise comparativa dos tempos de execução dos dois tipos de programas permitiu explorar os impactos do paralelismo, da arquitetura do processador e do tipo de carga computacional sobre o desempenho final.

III. RESULTADOS E DISCUSSÕES

Utilizando 10 milhões de elementos e iterações, os resultados observados mostram um comportamento coerente com a natureza dos programas.

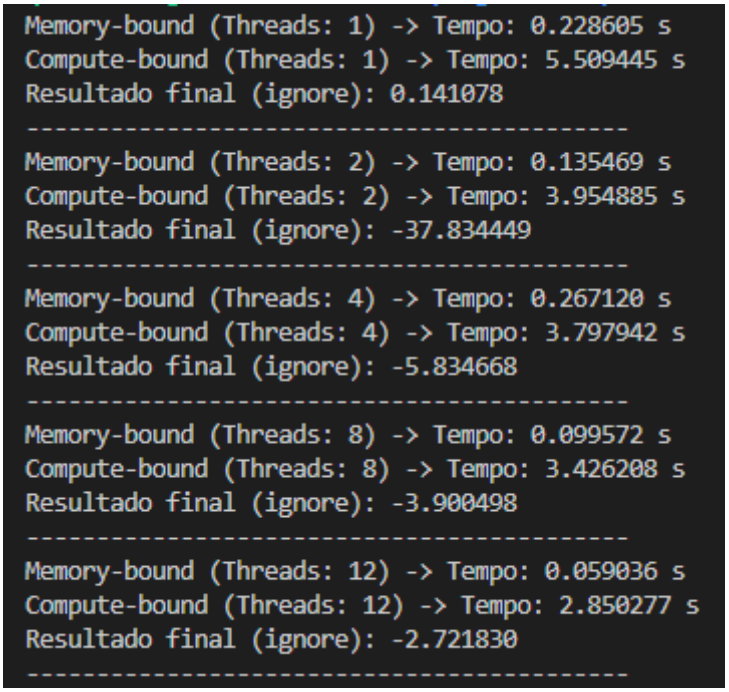

Imagem 1: Resultados de compilação

No caso do *memory-bound*, o tempo de execução diminuiu consideravelmente ao aumentar o número de *threads*, especialmente de 1 para 2, e continuou melhorando até 12 *threads*, ainda que de forma menos linear. Isso ocorre porque múltiplos *threads* ajudam a explorar melhor a largura de banda da memória, mesmo com alguma competição por *cache* e canais de acesso.

Já no *compute-bound*, também houve melhora no desempenho com mais *threads*, mas de forma mais gradual. O ganho foi limitado pela quantidade de núcleos físicos disponíveis, já que, ao ultrapassar esse limite, a competição por recursos internos da CPU reduz a eficiência do paralelismo.

Em resumo, o paralelismo com OpenMP foi benéfico em ambos os casos. Programas limitados por memória se beneficiam mais do *multithreading* por *hardware*, enquanto os limitados por CPU podem ser prejudicados por excesso de *threads*, devido à competição por recursos de cálculo.

IV. CONCLUSÃO

A análise demonstrou que o comportamento da paralelização depende fortemente do tipo de limitação enfrentada pelo programa. O programa de somas vetoriais é claramente *memory-bound*, pois sua performance está associada ao gargalo de Von Neumann — a limitação da taxa de transferência entre memória e CPU. Já o programa de cálculos matemáticos intensivos é *compute-bound*, limitado pela capacidade de processamento da CPU.

A *multithreading* de *hardware* pode ser benéfica para programas *memory-bound*, pois permite manter a unidade de execução ocupada enquanto threads aguardam dados da memória, aproveitando melhor o tempo ocioso. Por outro lado, para programas *compute-bound*, o uso excessivo de *threads* pode ser prejudicial. Isso ocorre porque múltiplos threads passam a disputar os mesmos recursos internos da CPU, o que pode diminuir o desempenho ao invés de melhorá-lo.

Portanto, é fundamental considerar a natureza da aplicação ao aplicar técnicas de paralelização. O uso eficiente de *threads* depende não apenas do número de núcleos disponíveis, mas também do tipo de tarefa executada. Essa análise ajuda a compreender melhor o impacto real da programação paralela e da arquitetura de *hardware* no desempenho de aplicações