Nesta tarefa o objetivo foi implementar dois programas paralelos em C utilizando a biblioteca OpenMP. Um dos programas deveria ser limitado por acesso à memória (memory-bound), e o outro limitado pelo uso intensivo da CPU (compute-bound). O desafio envolveu medir o tempo de execução desses programas utilizando diferentes quantidades de threads e, com base nos resultados, analisar o comportamento do desempenho, os ganhos com paralelismo e as limitações que surgem.

No primeiro programa, focado em ser limitado por memória, foram alocados cinco vetores grandes do tipo double. Quatro desses vetores foram preenchidos com o valor 1.0, e o quinto foi preenchido com a soma dos quatro vetores anteriores. O tamanho de cada vetor foi de 100 milhões de posições, o que garante um uso considerável de memória. A ideia desse teste é observar como o acesso à memória se torna o principal gargalo, já que o cálculo da soma é simples, mas a movimentação de dados entre RAM e CPU pode afetar o desempenho.

Esse código foi paralelizado com a diretiva #pragma omp parallel for e o tempo de execução foi medido com a função gettimeofday, conforme orientação. Os testes foram realizados com 1, 2, 3 e 4 threads, e os tempos medidos foram:

- 1 thread: 1.854526 segundos
- 2 threads: 0.668052 segundos
- 3 threads: 0.562622 segundos
- 4 threads: 0.336096 segundos

Com esses resultados, fica evidente que o tempo de execução caiu bastante ao usar 2 threads em vez de 1, mostrando um ganho significativo com o paralelismo. No entanto, com 3 e 4 threads, o ganho foi menor e começou a estabilizar. Isso acontece porque, apesar de existirem mais threads, todas acabam compartilhando o mesmo barramento de memória. Com isso, elas disputam o acesso à RAM, o que limita o aproveitamento total das threads. Esse comportamento é típico de programas memory-bound, em que o acesso à memória é mais lento do que o tempo necessário para processar os dados.

Já no segundo programa, o objetivo foi simular um código compute-bound, ou seja, que exige bastante do processador. Para isso, foi criado um laço de iteração muito grande, onde cada iteração realiza diversas operações matemáticas pesadas, como seno, cosseno, tangente, raiz quadrada, logaritmo e exponencial. Todas essas operações foram feitas sobre o mesmo valor (1.0), apenas para simular carga de processamento. O resultado final foi acumulado em uma variável global, também paralelizada com #pragma omp parallel for.

Os tempos de execução medidos com 1 até 4 threads foram os seguintes:

- 1 thread: 182.014061 segundos
- 2 threads: 94.957542 segundos
- 3 threads: 66.058959 segundos
- 4 threads: 53.957309 segundos

Nesse caso, o ganho de desempenho foi mais evidente do que no exemplo anterior. O tempo caiu praticamente pela metade ao passar de 1 para 2 threads, e continuou caindo com 3 e 4 threads. Isso mostra que o paralelismo ajuda bastante quando o gargalo está nos cálculos e não no acesso à memória. Como cada thread pode calcular sua parte de forma independente, sem necessidade de acessar constantemente a RAM, o desempenho se beneficia mais do multithreading.

Para entender melhor os limites desse ganho, também testei o programa compute-bound com 8 e 9 threads. Os tempos obtidos foram:

- 8 threads: 53.305365 segundos
- 9 threads: 53.449908 segundos

Esses novos resultados mostram que, a partir de um certo ponto, aumentar a quantidade de threads não traz mais ganhos. O tempo com 9 threads foi até um pouco pior do que com 8. Isso é um reflexo do uso de hyper-threading: quando o número de threads ultrapassa o número de núcleos físicos do processador, elas começam a dividir os mesmos recursos internos (como cache e registradores), gerando concorrência. Com isso, o desempenho se estabiliza ou até piora.

Essa observação mostra a importância de entender a arquitetura da máquina onde o programa está sendo executado. O paralelismo, apesar de muito útil, tem um limite prático que depende tanto do tipo de problema (se é limitado por memória ou por CPU) quanto da capacidade do hardware (quantidade de núcleos físicos, largura de banda da RAM, cache etc).

Códigos usados e resultados obtidos:

memoria.c > main(int, char * [])

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

#define tam 100000000

double tempoAtual() {
    struct timeval tempo;
    gettimeofday(&tempo, NULL);
    return tempo.tv_sec + tempo.tv_usec / 1000000.0;
}

int main(int argc, char *argv[]) {
    int numThreads = atoi(argv[1]);
    omp_set_num_threads(numThreads);

    double *vetor1 = malloc(sizeof(double) * tam);
    double *vetor2 = malloc(sizeof(double) * tam);
    double *vetor3 = malloc(sizeof(double) * tam);
    double *vetor4 = malloc(sizeof(double) * tam);
    double *vetorResultado = malloc(sizeof(double) * tam);

    for (int i = 0; i < tam; i++) {
        vetor1[i] = 1.0;
        vetor2[i] = 1.0;
        vetor3[i] = 1.0;
        vetor4[i] = 1.0;
    }

    double tempoInicio = tempoAtual();

    #pragma omp parallel for
    for (int i = 0; i < tam; i++) {
        vetorResultado[i] = vetor1[i] + vetor2[i] + vetor3[i] + vetor4[i];
    }

    double tempoFim = tempoAtual();
    printf("Tempo de execucao (limitado por memoria) com %d threads: %f segundos\n", numThreads, tempoFim - tempoInicio);

    free(vetor1);
    free(vetor2);
    free(vetor3);
    free(vetor4);
    free(vetorResultado);

    return 0;
}
```

```
JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ gcc -fopenmp memoria.c -o memoria.exe

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./memoria 1
Tempo de execucao (limitado por memoria) com 1 threads: 1.854526 segundos

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./memoria 2
Tempo de execucao (limitado por memoria) com 2 threads: 0.668052 segundos

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./memoria 3
Tempo de execucao (limitado por memoria) com 3 threads: 0.562622 segundos

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./memoria 4
Tempo de execucao (limitado por memoria) com 4 threads: 0.336096 segundos
```

cpu.c > main(int, char * [])

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

#define tam 100000000

double tempoAtual() {
    struct timeval tempo;
    gettimeofday(&tempo, NULL);
    return tempo.tv_sec + tempo.tv_usec / 1000000.0;
}

int main(int argc, char *argv[]) {
    int numThreads = atoi(argv[1]);
    omp_set_num_threads(numThreads);

    double resultado = 0.0;
    double tempoInicio = tempoAtual();

    #pragma omp parallel for reduction(+:resultado)
    for (int i = 0; i < tam; i++) {
        double x = 1.0;
        for (int j = 0; j < 10; j++) {
            x = sin(x) * cos(x) + tan(x) + log(x + 1.0) + sqrt(x + 2.0) + exp(x / 100.0);
        }
        resultado += x;
    }

    double tempoFim = tempoAtual();
    printf("Resultado: %f\n", resultado);
    printf("Tempo de execucaoo (limitado por CPU) com %d threads: %f segundos\n", numThreads, tempoFim - tempoInicio);

    return 0;
}
```

```
JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ gcc -fopenmp cpu.c -o cpu.exe

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./cpu 1
Resultado: 851194432.407356
Tempo de execucaoo (limitado por CPU) com 1 threads: 182.014061 segundos

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./cpu 2
Resultado: 851194433.858747
Tempo de execucaoo (limitado por CPU) com 2 threads: 94.957542 segundos

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./cpu 3
Resultado: 851194433.109097
Tempo de execucaoo (limitado por CPU) com 3 threads: 66.058959 segundos

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./cpu 4
Resultado: 851194433.138637
Tempo de execucaoo (limitado por CPU) com 4 threads: 53.957309 segundos
```

```
JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./cpu 8
Resultado: 851194433.509864
Tempo de execucaoo (limitado por CPU) com 8 threads: 53.305365 segundos

JP@PCGilmara MINGW64 /c/Users/JP.PCGilmara/Documents/AltoDesempenho/testes/ex4
$ ./cpu 9
Resultado: 851194433.465106
Tempo de execucaoo (limitado por CPU) com 9 threads: 53.449908 segundos
```