cpu_bound.c

```c

////////////////////////////////////////////////
// cpu_bound.c
//Programa CPU-Bound: Cálculos Intensivos

////////////////////////////////////////////////

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

// O tamanho do vetor
#define VECTOR_SIZE 20000

// Função com cálculos intensivos
double intensive_calculation(double val) {
    double result = val;
    for (int i = 0; i < 100000; i++) { // Repetir para aumentar a carga
        result = sin(result) * cos(val) + sqrt(fabs(result)) * tan(val / 2.0);
    }
    return result;
}

int main() {
    double *data = (double*)malloc(VECTOR_SIZE * sizeof(double));
    if (data == NULL) {
        fprintf(stderr, "Erro ao alocar memória.\n");
        return 1;
    }

    // Inicializa o vetor
    #pragma omp parallel for
    for (long i = 0; i < VECTOR_SIZE; i++) {
        data[i] = (double)i / 1000.0;
    }

    printf("Programa CPU-Bound: Cálculos Intensivos\n");
    printf("Tamanho do Vetor: %ld doubles\n", (long)VECTOR_SIZE);
    printf("----------------------------------------\n");

    // Loop para testar com diferentes números de threads
    for (int num_threads = 1; num_threads <= 16; num_threads *= 2) {
        omp_set_num_threads(num_threads);

        double start_time = omp_get_wtime();

        // Seção paralela. O gargalo aqui é a capacidade de processamento da CPU.
        #pragma omp parallel for
        for (long i = 0; i < VECTOR_SIZE; i++) {
            data[i] = intensive_calculation(data[i]);
```

```
        }

        double end_time = omp_get_wtime();
        printf("Threads: %2d | Tempo de Execução: %f segundos\n", num_threads, end_time - start_time);
    }

    free(data);

    return 0;
}
```