paralelo_critical_comp.c

```c
/*
 * pi_critical_compartilhado.c
 * * Estimativa de Pi (Monte Carlo) usando um contador compartilhado
 * protegido por uma diretiva #pragma omp critical.
 * Esta abordagem sofre de alta contenção e baixo desempenho.
 *
 * Compilação: gcc -o pi_critical -fopenmp pi_critical_compartilhado.c -lm
 * Execução:   ./pi_critical
 */


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

const long NUM_PASSOS = 100000000;

int main() {
    long pontos_no_circulo = 0;

    printf("Executando Versão: Contador Compartilhado + critical\n");
    printf("Calculando Pi com %ld passos...\n", NUM_PASSOS);

    double start_time = omp_get_wtime();

    #pragma omp parallel
    {
        // Garante uma semente única por thread para rand_r
        unsigned int seed = time(NULL) ^ omp_get_thread_num();

        #pragma omp for
        for (long i = 0; i < NUM_PASSOS; i++) {
            double x = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;
            double y = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;

            if (x * x + y * y < 1.0) {
                // A seção crítica protege o bloco de código.
                // Apenas uma thread pode executar este bloco por vez.
                #pragma omp critical
                {
                    pontos_no_circulo++;
                }
            }
        }
    } // Fim da região paralela

    double end_time = omp_get_wtime();
    double tempo_execucao = end_time - start_time;
```

```c
    double pi_estimado = 4.0 * pontos_no_circulo / NUM_PASSOS;

    printf("\nPi estimado = %f\n", pi_estimado);
    printf("Tempo de execucao: %f segundos\n", tempo_execucao);

    return 0;
}
```