

## paralelo\_atomic\_comp.c

```
1  /*
2   * pi_critical_compartilhado.c
3   * * Estimativa de Pi (Monte Carlo) usando um contador compartilhado
4   * protegido por uma diretiva #pragma omp critical.
5   * Esta abordagem sofre de alta contenção e baixo desempenho.
6   *
7   * Compilação: gcc -o paralelo_atomic -fopenmp paralelo_atomic.c -lm
8   * Execução:   ./paralelo_atomic
9   */
10
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <omp.h>
15 #include <time.h>
16 #include <math.h>
17
18 const long NUM_PASSOS = 100000000;
19
20 int main() {
21     long pontos_no_circulo = 0;
22
23     printf("Executando Versão: Contador Compartilhado + atomic\n");
24     printf("Calculando Pi com %ld passos...\n", NUM_PASSOS);
25
26     double start_time = omp_get_wtime();
27
28     #pragma omp parallel
29     {
30         // Garante uma semente única por thread para rand_r
31         unsigned int seed = time(NULL) ^ omp_get_thread_num();
32
33         #pragma omp for
34         for (long i = 0; i < NUM_PASSOS; i++) {
35             double x = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;
36             double y = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;
37
38             if (x * x + y * y < 1.0) {
39                 // A seção atômica protege o bloco de código.
40                 // Apenas uma thread pode executar este bloco por vez.
41                 #pragma omp atomic
42                 pontos_no_circulo++;
43             }
44         }
45     } // Fim da região paralela
46
47     double end_time = omp_get_wtime();
48     double tempo_execucao = end_time - start_time;
49     double pi_estimado = 4.0 * pontos_no_circulo / NUM_PASSOS;
50 }
```

```
51     printf("\nPi estimado = %f\n", pi_estimado);
52     printf("Tempo de execucao: %f segundos\n", tempo_execucao);
53
54     return 0;
55 }
```