paralelo_reduction.c

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

// Definição global do número de passos para consistência
const long NUM_PASSOS = 100000000;

long pi_paralel_for_reduction() {
    long pontos_no_circulo = 0;

    #pragma omp parallel for reduction(+:pontos_no_circulo)
    for (long i = 0; i < NUM_PASSOS; i++) {

        unsigned int seed = time(NULL) ^ omp_get_thread_num();

        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) {
            pontos_no_circulo++;
        }
    }

    return pontos_no_circulo;
}

int main() {
    double start_time, end_time;
    long total_pontos_no_circulo;

    printf("Iniciando analise de desempenho para %ld passos com reduction.\n",
NUM_PASSOS);

    start_time = omp_get_wtime();

    // Chama a função e armazena o valor retornado
    total_pontos_no_circulo = pi_paralel_for_reduction();

    end_time = omp_get_wtime();

    double tempo_paralelo = end_time - start_time;

    // Usa a variável local da main para calcular o Pi
    double pi_estimado = 4.0 * total_pontos_no_circulo / NUM_PASSOS;

    printf("\nEstimativa paralela de pi = %f\n", pi_estimado);
    printf("Tempo Paralelo: %f segundos\n", tempo_paralelo);

```

```
50      return 0;
51  }
```