Introduction

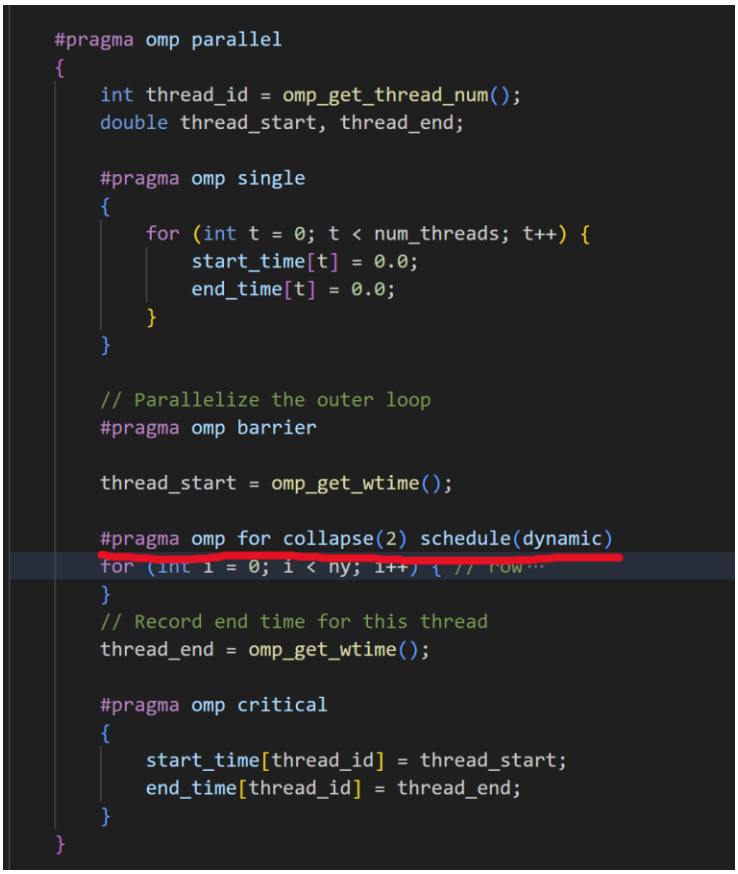
In this assignment, we parallelized the computation of the Mandelbrot set using OpenMP to improve performance. The Mandelbrot set is a complex mathematical set of points that produces a fractal when plotted. This task is challenging due to the nature of the calculation, which requires efficient parallel processing to handle large data sets and ensure load balancing among threads.

Problem Description

The Mandelbrot set is defined by iterating a complex function and checking if the result remains bound. Each point in the complex plane is tested, and if it escapes beyond a certain threshold, it is classified based on the number of iterations it took to escape.

Parallelization with OpenMP

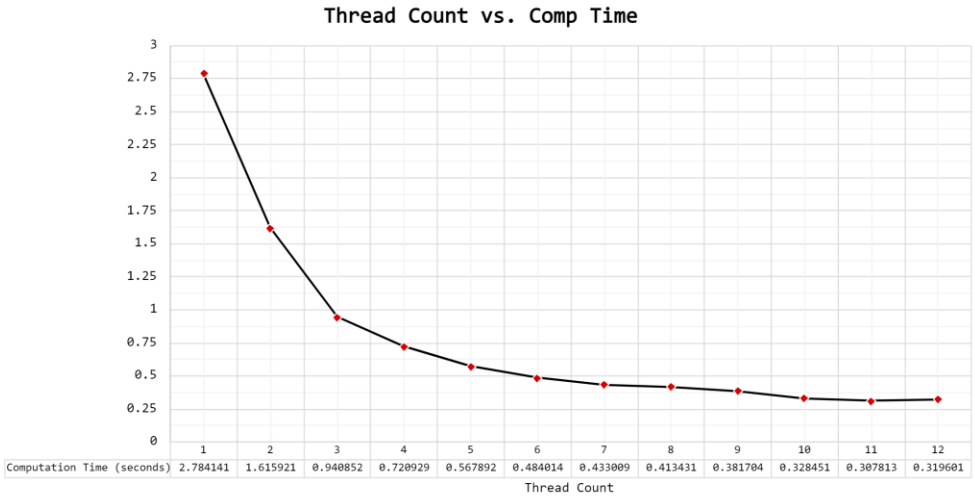
To parallelize the computation, we use OpenMP's `#pragma omp parallel for` directive. This allows us to distribute iterations of the nested loops across multiple threads:



```
#pragma omp parallel
{
    int thread_id = omp_get_thread_num();
    double thread_start, thread_end;

    #pragma omp single
    {
        for (int t = 0; t < num_threads; t++) {
            start_time[t] = 0.0;
            end_time[t] = 0.0;
        }
    }

    // Parallelize the outer loop
    #pragma omp barrier

    thread_start = omp_get_wtime();

    #pragma omp for collapse(2) schedule(dynamic)
    for (int i = 0; i < ny; i++) { // row···
    }
    // Record end time for this thread
    thread_end = omp_get_wtime();

    #pragma omp critical
    {
        start_time[thread_id] = thread_start;
        end_time[thread_id] = thread_end;
    }
}
```

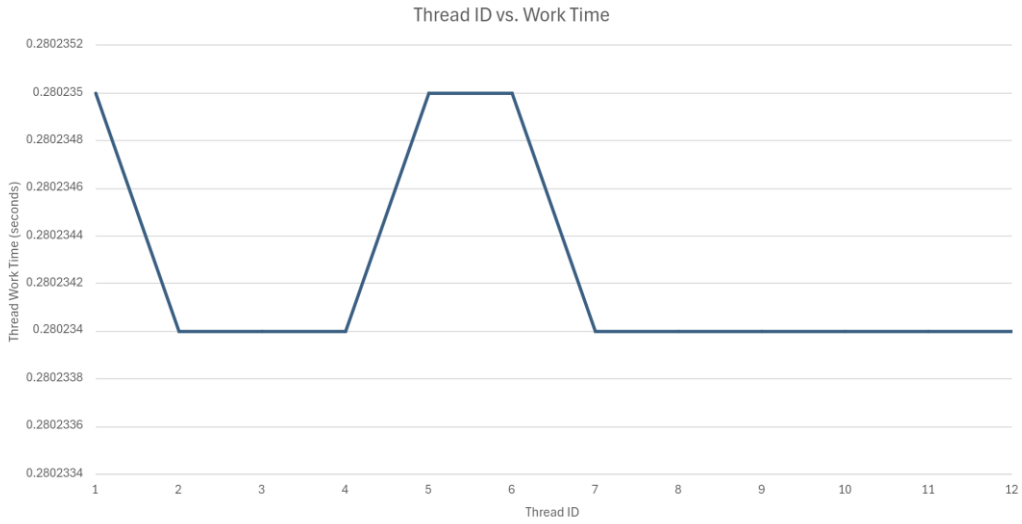
Strong Scaling

The strong scaling plot will illustrate how the computation time decreases as the number of threads increases.



Load Balancing

The load balancing plot will show the distribution of computation times among threads. This demonstrates whether each thread completed its assigned work in roughly the same amount of time.



Conclusion

The parallelization of the Mandelbrot set computation using OpenMP significantly improved performance, demonstrating the efficiency of multi-threading for this type of problem. The strong scaling and load balancing plots confirm that the computation was well-distributed among threads, with each thread handling its workload effectively.