13X007: Assignment #5

Due on 04.12.2023

Parallelism

UNIVERSITÉ DE GENÈVE

CHRISTOFOROU Anthony

Abstract

This report delves into the detailed implementation and analysis of the Mandelbrot set generation using OpenMP for parallel computing. We examine the effects of various thread counts and computational regions on performance, providing insights into parallelization strategies and their implications on complex mathematical computations.

Contents

1 Introduction

The Mandelbrot set is a cornerstone in fractal geometry, epitomizing the intricate structures that emerge from simple mathematical rules. Its computation, inherently complex and resource-intensive, presents an ideal case for exploring the capabilities of parallel computing. This report presents a deep dive into the generation of the Mandelbrot set using OpenMP, a widely-used parallel programming framework in C++. We aim to explore how parallel computation can be leveraged to optimize the rendering of this fractal, analyzing the performance across different thread counts and regions of computation.

2 Methodology

2.1 Theory

The Mandelbrot set is a set of complex numbers defined by a simple iterative formula: For a given complex number c, the sequence $f_c(0)$, $f_c(f_c(0))$, ..., with $f_c(z) = z^2 + c$, remains bounded. The beauty of the Mandelbrot set lies in the complexity and variety of patterns that emerge from this simple definition.

2.2 Mandelbrot Set Computation

Our implementation in C++ leverages complex number arithmetic to iterate over points in the complex plane. We evaluate the boundedness of each point under the iterative function, thereby determining its membership in the Mandelbrot set.

```
// Function to check if a point is in the Mandelbrot set
int isInMandelbrotSet(std::complex<double> c, int maxIterations) {
    std::complex<double> z = 0;
    int n = 0;
    for (n = 0; n < maxIterations; ++n) {
        if (std::abs(z) > 2.0) {
            break;
        }
        z = z * z + c;
    }
    return n;
}
```

This function takes a complex number `c` and an iteration limit `maxIterations` as parameters. It initializes a complex number `z` to zero and iterates `z` as `z = z * z + c`. If the magnitude of `z` exceeds 2, the function terminates, indicating that `c` does not belong to the Mandelbrot set. The number of iterations before divergence is returned as an indication of how quickly the points diverge.

2.3 Parallelization

Using OpenMP, we distribute this computation across multiple threads. This section will discuss the specific OpenMP directives employed, their configuration, and the rationale behind their use in optimizing the computation process.

```
// Parallel computation of Mandelbrot set
#pragma omp parallel for private(x, y, c, n)
for (int i = 0; i < imageHeight; ++i) {
    for (int j = 0; j < imageWidth; ++j) {
        c = std::complex<double>(x_min + j * pixel_size_x,
                                 y_min + i * pixel_size_y);
        n = isInMandelbrotSet(c, maxIterations);
    }
}
```

In this snippet, the `#pragma omp parallel for` directive is used to distribute the computation of each pixel in the Mandelbrot set image across multiple threads. Each thread computes a subset of the total pixels, thereby dividing the workload and reducing the overall execution time.

2.4 Image Rendering

The computed set is rendered into a BMP image using a custom function, `write_to_bmp`. This function maps the number of iterations required for divergence to a color scale, effectively visualizing the fractal pattern of the Mandelbrot set.

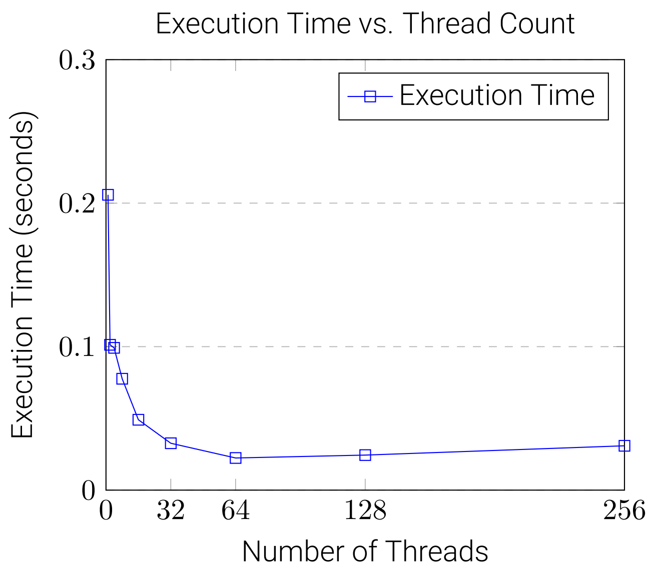
3 Results

3.1 Execution Time vs. Thread Count

We conducted experiments with varying numbers of threads to analyze the performance of our parallel algorithm. This section provides a detailed examination of the execution times across different thread counts, highlighting the scalability and efficiency of the parallelized implementation.

```
Execution Time with 1 threads: 0.205807 seconds
Fractal image saved as BMP.
Execution Time with 2 threads: 0.101268 seconds
...
Execution Time with 128 threads: 0.0244127 seconds
Fractal image saved as BMP.
Execution Time with 256 threads: 0.0308905 seconds
Fractal image saved as BMP.
```

The execution times show a clear trend: as the thread count increases from 1 to 64, there is a significant reduction in execution time, demonstrating the benefits of parallel computation. However, post 64 threads, there is a slight increase in execution time, indicating a point of diminishing returns.



3.2 Visualization

Here, we present the rendered images of the Mandelbrot set for different iteration counts. The section includes an analysis of the fractal patterns, discussing how they evolve with the increase in iterations and threads.

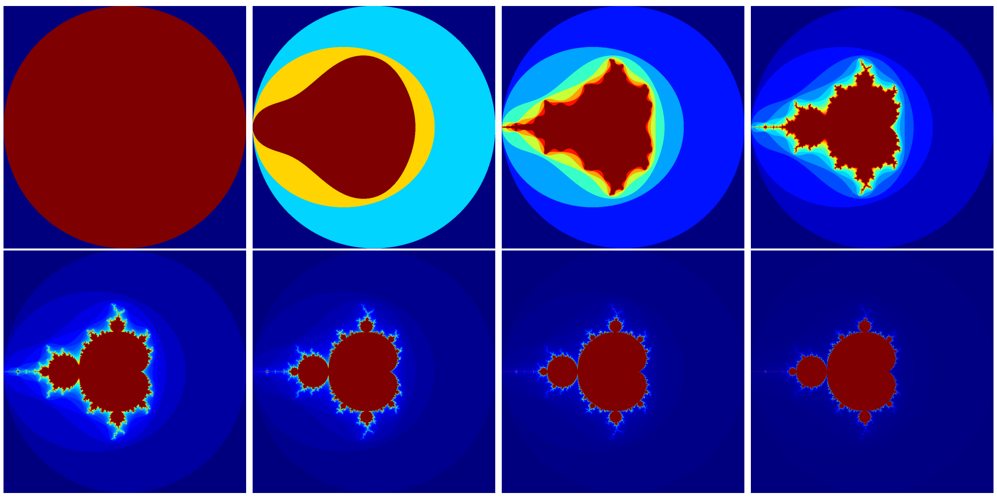


Figure 1: Evolution of Mandelbrot Set from 2 to 256 iterations

3.3 Region Variation

We also explored how varying the region of computation in the complex plane affects the execution time and the resulting fractal patterns. This section provides insights into the relationship between the complexity of the region and the computational effort required.

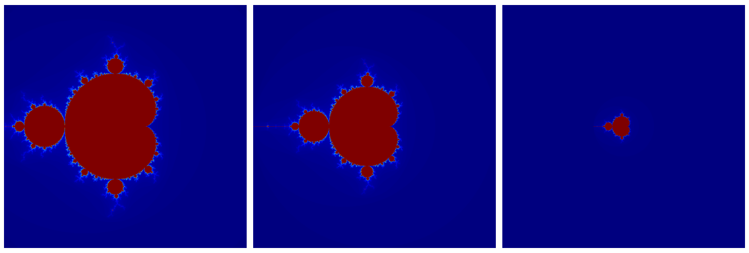


Figure 2: Different regions of computation

3.3.1 Zone Visualization

3.3.2 Time Variation

- Region: -1.5 -1.5 1.5 1.5, Time: 0.0705245 seconds
- Region: -2 -2 2 2, Time: 0.0479166 seconds
- Region: -8 -8 8 8, Time: 0.00753379 seconds

4 Discussion

The results indicate an initial trend of decreasing execution time with increased thread counts, demonstrating the efficiency of parallel computing. However, the slight increase in execution time beyond 64 threads can be attributed to several factors, including thread management overhead, memory bandwidth limitations, and the inherent complexity of the algorithm.

The overhead of managing a large number of threads can outweigh the benefits of parallelization. As more threads are added, the cost of creating, scheduling, and synchronizing these threads becomes significant. This phenomenon is often observed in parallel computing and is a critical factor in determining the optimal number of threads for a specific computation.

Memory bandwidth limitations also play a role. With more threads, the contention for memory access increases, potentially leading to bottlenecks that impede overall performance. This is particularly relevant for algorithms like the Mandelbrot set computation, where each thread requires access to memory for reading and writing data.

Furthermore, the complexity of the Mandelbrot set itself influences execution time. Regions with more complex patterns require more iterations to resolve, leading to longer computation times. This aspect was evident in our experimentation with different computation regions, where more intricate areas of the set demanded more computational resources.

5 Conclusion

The exploration of the Mandelbrot set using OpenMP underscores the potential and challenges of parallel computing in handling complex mathematical computations. This report concludes with a reflection on our findings, emphasizing the balance between computational resources and efficiency, and suggesting avenues for future research in this domain.