Taha Al Khashmany
Rene Sorger
Gregor Zeman

12.03.2023

Our approach involves parallelizing the nested for loops that iterate over each pixel in the image. We use OpenMP's default scheduling policy of dynamic scheduling with chunk size 1 to ensure load balancing between threads. Additionally, we use a VIEW_SHIFT constant to shift the point that the program is zooming in on to the left, allowing the program to explore different parts of the Mandelbrot set.

Results:

We tested the performance of our parallel solution and compared it to the sequential implementation on a 1024x1024 image. We measured the pure computation time, without writing to disk and averaged over multiple runs. We used a release build and warm-up to ensure accurate measurements. Running on one thread, the load increases with each iteration significantly. Running on 24 threads, the program speedup is 5-6 times faster, and the load however increases significantly slower as well.
1 thread: average 130ms
24 threads: average 20ms

Additional findings and interesting remarks:

- The OpenMP parallelization significantly reduces the execution time of generating the Mandelbrot set, allowing for more exploration of the set or higher resolution images.
- The speedup of the parallel solution depends on the number of threads, with diminishing returns after a certain number of threads. It is important to experiment with the number of threads to find the optimal performance for a given system.

Conclusion:

In conclusion, we successfully implemented a parallel solution for generating the Mandelbrot set using OpenMP. Our performance measurements showed that the parallel solution achieves a significant speedup compared to the sequential implementation, with diminishing returns after around 8 threads. Overall, our approach provides a more efficient way to generate the Mandelbrot set and explore its visual features.