MPI Programming for Julia Sets

Jiayao Pang, Kaihang Liu, Yao Luo

Wilfred Laurier University

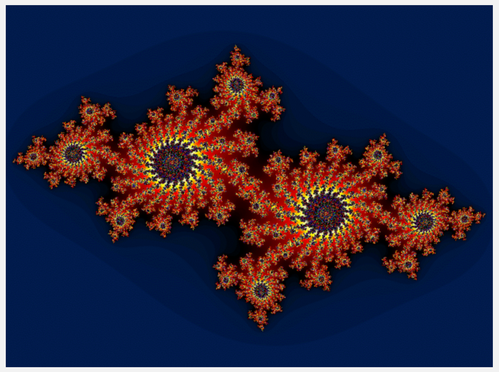
Objectives

The purpose of this project is to produce high-resolution pictures of Julia sets using C/MPI programming to compute the corresponding points and OpenGL to render the pictures.

Julia sets are fractal sets that are defined via the iteration of functions, such as: $Q_c(z) = z^2 + c$ and $T_c(z) = z^3 + c$, where z; c are complex numbers.

Introduction

Julia sets, named after Gaston Julia (1893-1978), arise from analysing the dynamics of complex functions. We first fix a function f : C → C and then consider the local behaviour of f around points in the complex plane. A function f is said to display sensitive dependence at a point z C if, roughly speaking, there exists a constant > 0, such that no matter how small a neighbourhood of z we consider there will be a point w in that neighbourhood such that the respective images of z and w under f are at least apart.



Load Balancing

Our solution is to divide the 1000 × 1000 matrix into different matrices according to the rows, and assign these split matrices to different processors.

The processor processes the points in the assigned matrix, and after the Orb (z0) operation, the final result of each point is obtained.

Implementation

We implement Orb(z0) by equation: $Q_c(z) = z^2 + c$. During the iteration, if the points obtained after the iteration are outside the specified range, the iteration is stopped, and the number of iterations is stored in the one-dimensional array of the processor. If after 50 iterations, the obtained point is still not out of range, stop the iteration of this point, and store 0 in the corresponding position of dimensional array.

Gather & Store Data

1. **Gather:**We use MPI_Gatherv to gather data obtained by the processors to PROCESSOR_0 and let it integrate all result.

```
MPI_Gatherv(local_colors, local_size, MPI_INT, colors, cc

if (rank == 0)
{

   array_to_file(colors);

   free(colors);
   free(color_indices);
   free(color_part_lengths);
}
```

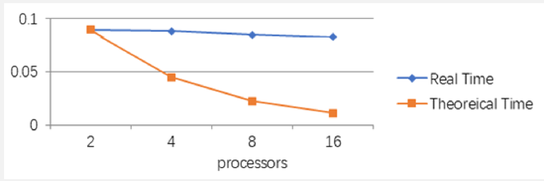
2. **Store:** Store the data in an array and write to a file.

Draw graph

Decide color based on the #iterations. Integrate and draw the picture based on the obtained colors.

Use the “FreeImage” library and create the “grab” function to store the graph as a ’.jpg’ file.

Benchmark



- 2 Processors: 0.089466 s
- 4 Processors: 0.087926 s
- 8 Processors: 0.084527 s
- 16 Processors: 0.082583 s

Conclusion

As the graph shows, the execution time decreases as we add processors. And it’s a “perfect” speed improvement theoretically. But in reality, due to the reduction operations, the theoretical and real execution time are different and the difference grows with adding processors.

Result