Lebanese American University

Department of Computer Science & Mathematics

CSC447-Parallel Programming

Assignment 1

Student Name: Samir Khansa

Student ID: 202001781

Instructor: Dr Hamdan Abdelllatef

Table of Contents

1. How I will parallelize the computation.

The Mandelbrot set is defined as the set of all complex numbers ‘c’ for which the function f_c(z) = z^2 +c does not diverge when iterated from an initial value of z=0. In other words, it’s the collection of complex numbers for which a certain mathematical process remains bounded (does not go to infinity) as the iteration proceeds. Importantly, each complex number within the Mandelbrot set, represented by a unique point in the complex plane, undergoes a different number of iterations before it is determined whether it belongs to the set or not.

This variation in the number of required iterations for each point in the set highlights the suitability of the Mandelbrot set for parallelization using MPI (Message Passing Interface) or similar parallel computing techniques. Because different points can be processed independently, this computational task can be efficiently divided among multiple processors or computing units, leading to faster and more efficient calculations.

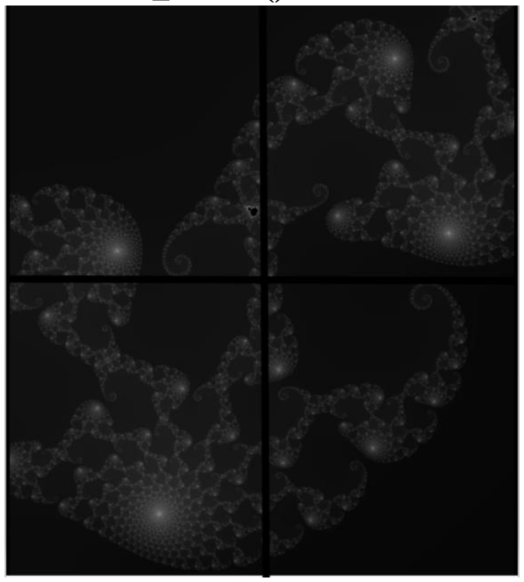
To parallelize this computation we would divide the computation into 4 equal size of chunks where every process would receive a one of the four chunks to execute it which is called Static paralyzing technique. Moreover, we are also able to parallelize the computation using the

Dynamic parallelization technique where we would divide the computation into multiple parts and whenever a process is ready to take a small chunk the root processor would assign it another chunk until it finished the image. After all the processors are finished, the result will be gathered into a single output image.

The parallelization is achieved using the library MPI (Message Passing Interface) in the C programming language. The algorithm would calculate the Mandelbrot set in parallel across multiple processes. Moreover, every process is tasked to a specific portion of the image to execute where the results are gathered between all the processes and combine them into one image.

The procedure of parallelization using MPI (Message Passing Interface) is as follows.

i. Initialization of MPI using the function MPI_Init().
ii. Dividing the program into chunks and assigning each chunk to a specific processor.
iii. Every processor would compute the Mandelbrot set calculations for its portion of the image.
iv. The results are then gathered and combined into a final image using the MPI_Gather().
v. Finally, I would finalize my MPI using the function MPI_Finalize().



In this diagram, the master process would be dividing the image into equal chunks and assigning each chunk to different processor this is Static computation. Moreover, every process would do their Mandelbrot set calculations for only the chunk that is assigned to it. At the moment where every process finishes execution, the results are gathered by the master process using the MPI function MPI_Gather() function to combine the results and create the final image. The execution time will be measured using the MPI_Wtime() function, The total time is acquired by reducing the parallel time across the processes using the function MPI_Reduce() function. The sequential time is calculated by subtracting the total time from the elapsed time.

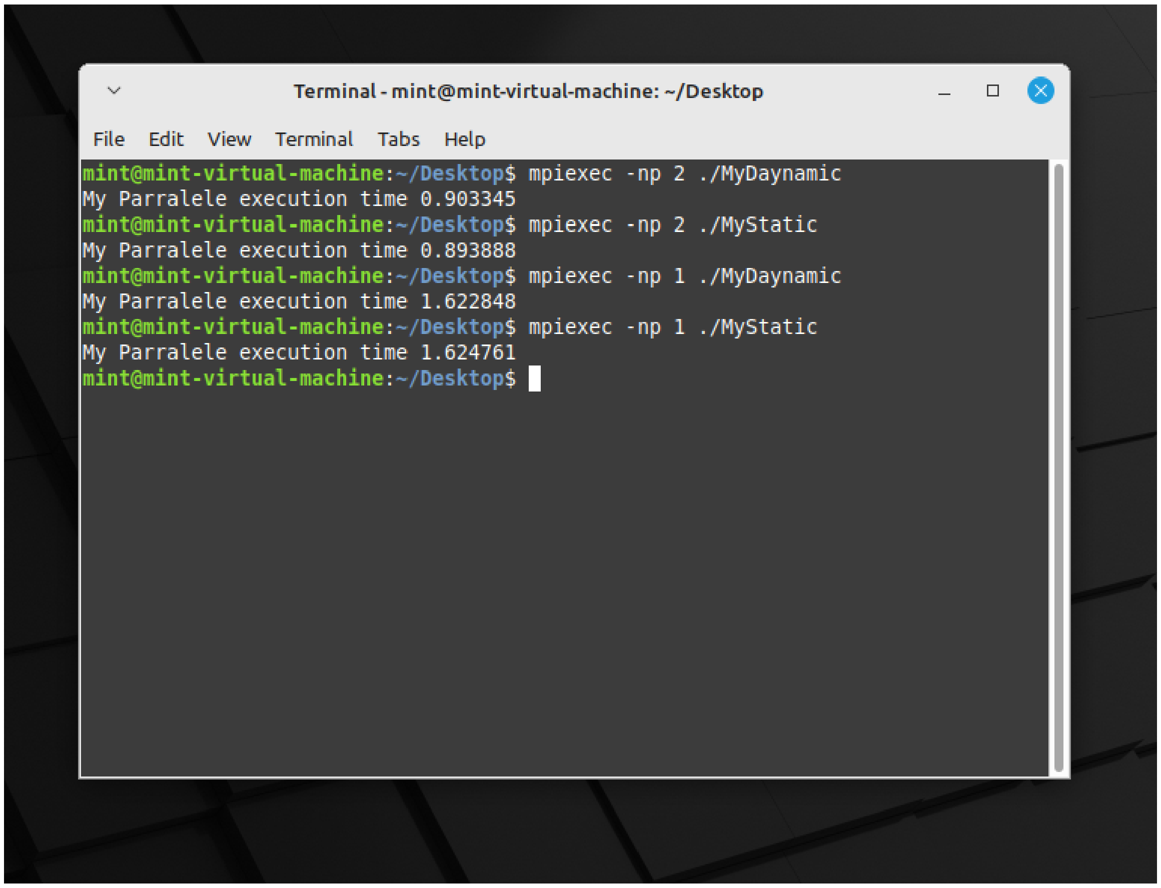
2. The setup I used

The setup that I used is the linux Mint and I have 8GB of RAM on my laptop. Moreover, I installed the library MPI (Message Passing Interface) for the parallel computation. MPI is a message passing system that allows multiple processes or nodes to communicate with each other at the same time. The code use MPI functions such as MPI_Init() Initialization of the MPI, MPI_Comm_rank() to acquire rank of the process, MPI_Comm_size() to acquire the size of the process, MPI_Wtime() to measure the time, MPI_Reduce() to reduce the parallel time across all processes, MPI_Gather to gather the data from all of the processes. However, unfortunately I was not able to allow MPI to use more than 2 processors at the same time.

3. Github Link

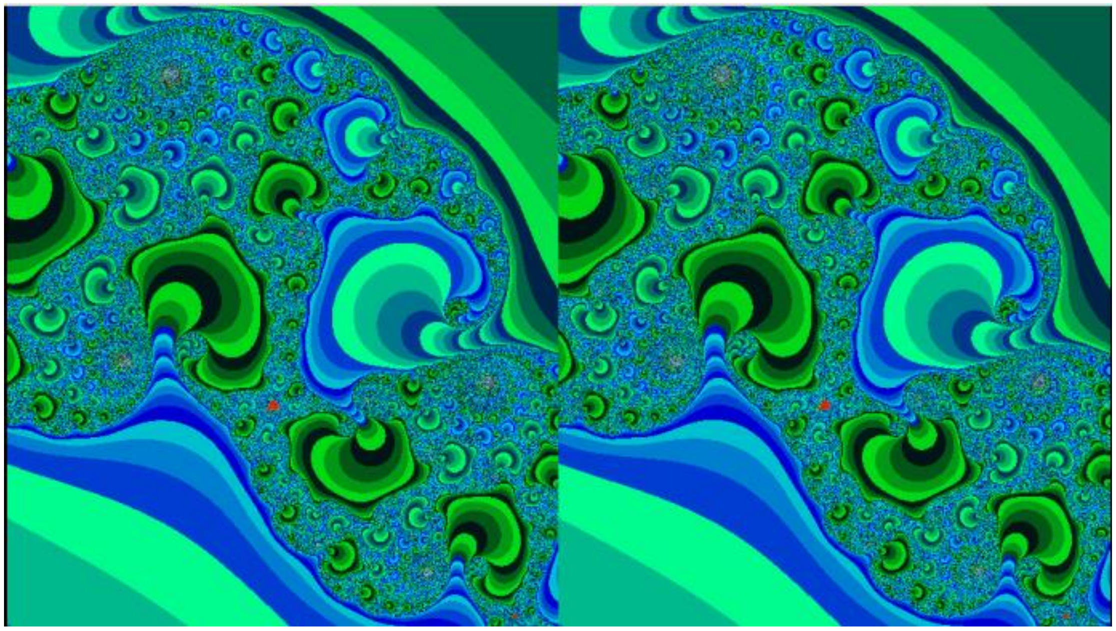
https://github.com/SamirKhansa

4. Screenshots Showing that the implementation is working and the resulting image.


Terminal - mint@mint-virtual-machine: ~/Desktop
File Edit View Terminal Tabs Help

```
mint@mint-virtual-machine:~/Desktop$ mpiexec -np 2 ./MyDaynamic
My Parralele execution time 0.903345
mint@mint-virtual-machine:~/Desktop$ mpiexec -np 2 ./MyStatic
My Parralele execution time 0.893888
mint@mint-virtual-machine:~/Desktop$ mpiexec -np 1 ./MyDaynamic
My Parralele execution time 1.622848
mint@mint-virtual-machine:~/Desktop$ mpiexec -np 1 ./MyStatic
My Parralele execution time 1.624761
mint@mint-virtual-machine:~/Desktop$
```

The first output in the photo below shows the time taken for a Dynamic paralyzing technique and the second output shows the time taken using the Static paralyzing technique. Unfortunatly, I was not able to use more than 2 processors therefore I showed the output of Dynamic using 1 processor and the output of Static using 1 processor. Therefore the Sequential time is 1.624761 and the parallel time is 0.903345 for Daynamic and 0.893888 for static.



5. The performance of the paralyzed code:

Speedup:

The speedup ratio measures how quickly one process executes compared to several processes. It stands for the performance in using one processor from many processors. Calculating the speedup factor is as follows:

SpeedUp=Sequential Time/ Parralel Time=1.624761/0.893888=1.817633753

Therefore the speedup factor is 1.817633753

Efficiency

Efficiency refers to how well a system uses its available processors. It is described as the speedup divided by the total number of processors being utilized.

The efficiency may be determined as follows:

Efficiency = Speedup / p= 1.817633753/2=0.903168765x100=90.317%

Scalability:

Scalability measures how well a program can take advantage of increased resources to solve large or more complex problems without a significant decrease in performance or efficiency.

Scalability = (Sequential time / (Number of processors * Parallel time)) = (1.624761/ (2*0.893888))=0.9088168764

Computation to Communication Ratio (CCR):

Computation to Communication Ratio (CCR) is a performance metric used in parallel and distributed computing to evaluate the balance between computational work and communication overhead in parallel program or algorithm. It quantifies the efficiency of a parallel application by measuring the ratio of computation time to communication time.

- Computation Time: The total time spent on performing computational tasks.
- Communication Time: The total time spent on communication and synchronization between processors.

CCR = Computation time / Communication time

Computation time= Parallel time / Number of processors=0.893888/2=0.446944 Seconds

Communication time = Parallel time - Computation time=0.893888-0.446944=0.446944

CCR = Computation time / Communication time=0.446944/0.446944=1. Which means that for every time spend on communication, 1 second is spent on computation. Unfortunately I am not able to use more than 2 processors to be able to have better results.

6. Conclusions:

Based on the calculations, the code for paralyzing the Mandelbrot set appears to exhibit acceptable scalability, efficiency, and speedup. Scalability, with a value of 0.9088168764, indicates the program's capacity to leverage increased resources for solving larger or more complex problems without a significant decrease in performance or efficiency. An efficiency score of 90.317% suggests that the program effectively utilizes approximately 90.317% of available resources. Additionally, a speedup factor of 1.817633753 demonstrates that the program runs nearly as fast as two separate programs combined. However, it's important to note that these values may vary on different computers.

Furthermore, I encountered a limitation when attempting to utilize more than two processors, as I received an error indicating that there were no available slots when I entered the command line "mpiexec –np 3 ./MyStatic" on this computer.