Fractal Assignment
Evaluation Report

1. PURPOSE OF THIS DOCUMENT

The purpose of this document is to describe a series of tests that accurately and with a reasonable degree of repeatability demonstrates that the Mandelbrot image creation has been implemented using multithreading to achieve Parallelism.

In this experiment, we are computing the Mandelbrot image by dividing it into n parts so that each part can be assigned to a single thread, leading to creating n number of threads, and combining them parallelly to form a desired Mandelbrot image.

2. Command Line Arguments

Command line argument used to compile the code is

gcc -o mandel mandel.c bitmap.c -pthread

Command line arguments used to execute the program is

mandel -x 0.2869325 -y 0.0142905 -s .0001 -W 1024 -H 1024 -m 1000

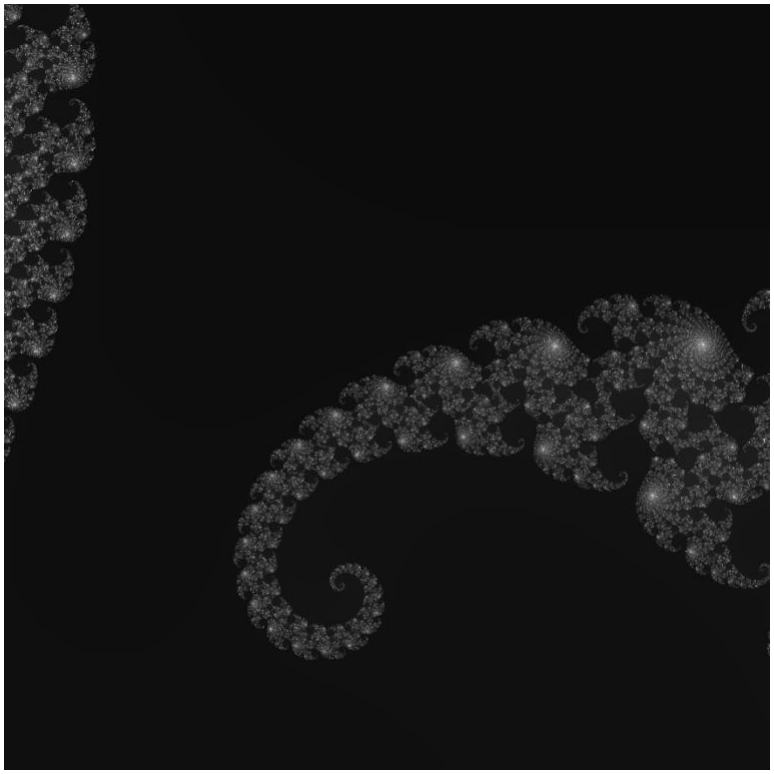
The above command used a single thread to execute the program.

```
[[nxr2126@omega ~]$ gcc -o mandel mandel.c bitmap.c -pthread
[[nxr2126@omega ~]$ mandel -x 0.2869325 -y 0.0142905 -s .0001 -W 1024 -H 1024 -m 1000
mandel: x=0.286932 y=0.014290 scale=0.000100 max=1000 outfile=mandel.bmp numberofthreads=1
This code took 859329 microseconds to execute
```

In addition to above command, argument -n has been added with different values starting from 2 to 6 to note duration that the program took to compute same part of the image.

```
[[nxr2126@omega ~]$ mandel -x 0.2869325 -y 0.0142905 -s .0001 -W 1024 -H 1024 -m 1000 -n 2
mandel: x=0.286932 y=0.014290 scale=0.000100 max=1000 outfile=mandel.bmp numberofthreads=2
This code took 444766 microseconds to execute
[[nxr2126@omega ~]$ mandel -x 0.2869325 -y 0.0142905 -s .0001 -W 1024 -H 1024 -m 1000 -n 3
mandel: x=0.286932 y=0.014290 scale=0.000100 max=1000 outfile=mandel.bmp numberofthreads=3
This code took 344122 microseconds to execute
[[nxr2126@omega ~]$ mandel -x 0.2869325 -y 0.0142905 -s .0001 -W 1024 -H 1024 -m 1000 -n 4
mandel: x=0.286932 y=0.014290 scale=0.000100 max=1000 outfile=mandel.bmp numberofthreads=4
This code took 270593 microseconds to execute
[[nxr2126@omega ~]$ mandel -x 0.2869325 -y 0.0142905 -s .0001 -W 1024 -H 1024 -m 1000 -n 5
mandel: x=0.286932 y=0.014290 scale=0.000100 max=1000 outfile=mandel.bmp numberofthreads=5
This code took 237592 microseconds to execute
[[nxr2126@omega ~]$ mandel -x 0.2869325 -y 0.0142905 -s .0001 -W 1024 -H 1024 -m 1000 -n 6
mandel: x=0.286932 y=0.014290 scale=0.000100 max=1000 outfile=mandel.bmp numberofthreads=6
This code took 197312 microseconds to execute
```

Resultant output image of the above computation of Mandelbrot image is as below.



3. Testing

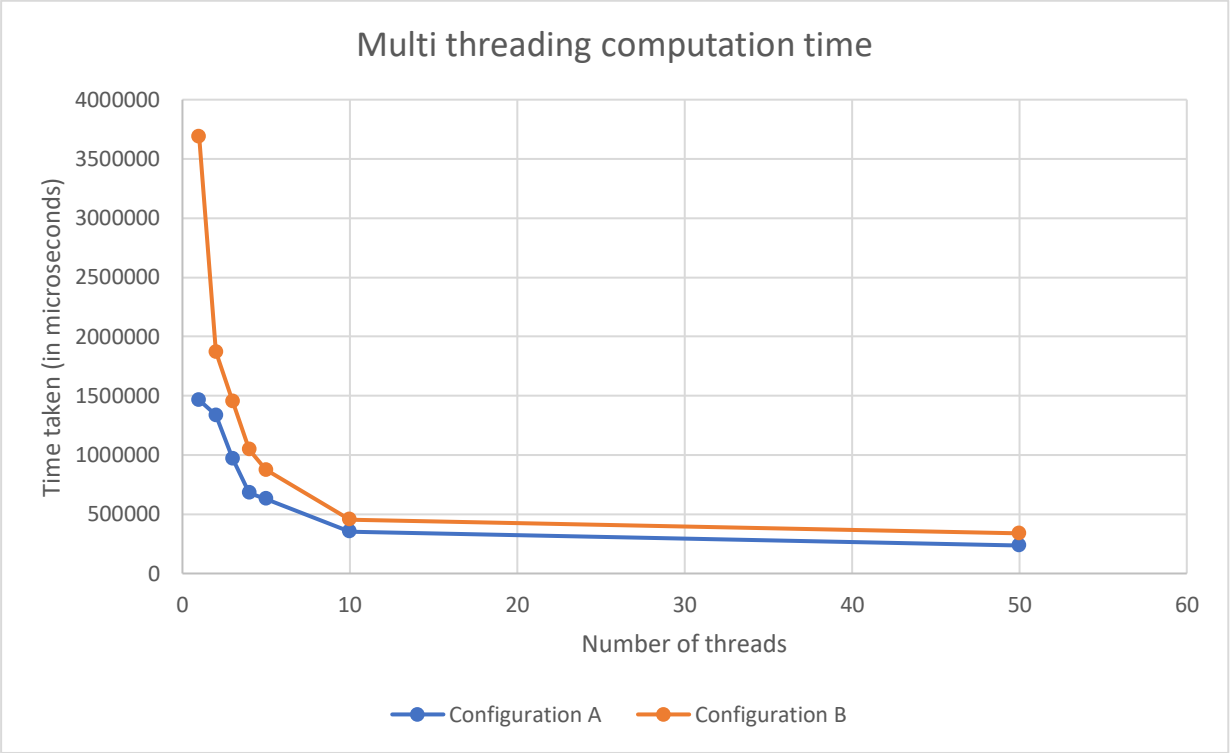
Testing has been performed for two different configurations and running with different number of threads. Below are the readings for each configuration

Configuration A:

Number of threads	Time (in microseconds)
1	1463490
2	1337244
3	970665
4	680793
5	630053
10	355408
50	236313

Configuration B:

Number of threads	Time (in microseconds)
1	3685726
2	1872050
3	1455292
4	1047177
5	875922
10	454291
50	338927



As from the above graph, the time taken to compute both the configurations of Mandelbrot image decrease exponentially with increasing number of threads.

The optimal number of threads that can be used to compute the image is 10 since the time taken by the program reduce by around 50% when 10 threads are initiated.

The shape of the curves for configuration A and B is different because configuration B is trying to generate a high-quality image because of which it takes long to execute the program compared to curve A.