Name: Bhupendra Ramdam
Id no: 1001370027

Mandelbrot Experiment using multi-threading

Objective:

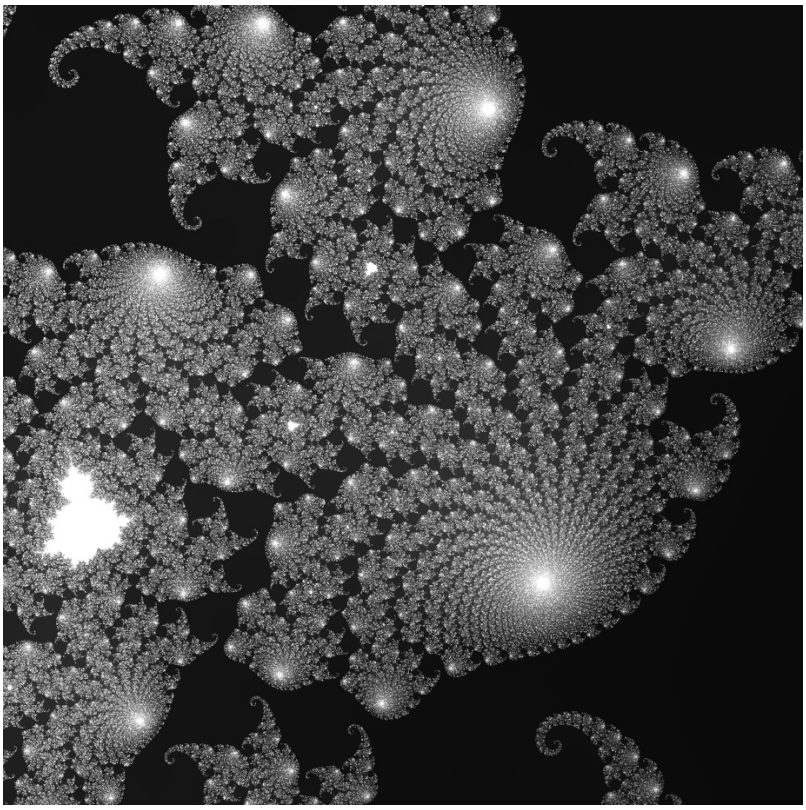
To understand the working of multi-threading and use it to reduce the run time for the Mandelbrot images.

Experimental setup:

We are given the program that can compile and compute the images with single threading and save them as BMP files. We need to reduce the execution time to compute those images by modifying the given code. We are also provided with two more configuration that we need to keep track of.

Procedure and description:

The image I found, took about 5 seconds to run. For this, I choose the configuration "./mandel -x 0.286922 -y 0.015297 -s 0.0001 -m 1500 -W 1250 -H 1250 -o mandel3.bmp". Here is the image for this particular configuration:



I choose pthread.h library for this experiment. pthread_create function creates the new thread with the attribute that contain in the structure “mystruct” in my experimental code. Whereas the function “pthread_join” will allow the program to wait until the created threads gets terminated. Default thread number is assigned as 1 but the user can thread it by more than 1. For that, I found the image height which is “new_height= height/thread_num”. If the thread number is 1, it will produce the whole image but if thread number is more than one than the total height of images will be divided into number of bands as required by the user. Dividing the threads into number of bands that will allow the compiler to work on those numbers of bands separately. Ultimately makes the code run faster with multiple threading and produce image in shorter time.

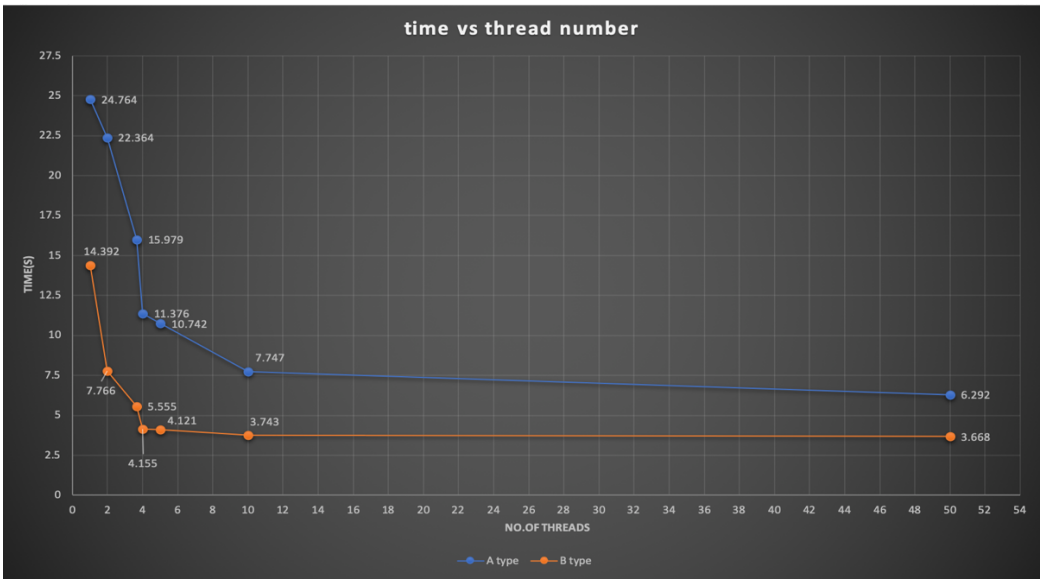
I compiled the code for the given configuration:

A: mandel -x -.5 -s 1 -m 2000 -W 2048 -H 2048

B: mandel -x 0.2869325 -y 0.0142905 -s .000001 -W 2048 -H 2048 -m 1000

for 10 times with the numbers of threads 1,2,3,4,5,10,50 and track the run time and choose the one that will be the achieved faster.

The following graphs explains the execution time for different thread with different time.



Observation:

The curves I found after plotting time vs number of threads looks like an inverted parabola. There is an inverted relation with time and number of threads. With the increase in number of threads, the compilation time decreases. In the graph, A (represented by blue) the compilation time for thread one is 24.764s which drops simultaneously as we increase the number of threads. But when the thread number is increased past 10, there is a slight variation of change in time. Finally, for the 50 threads, the execution time is 6.292s. Similarly, in the graph, B (represented by orange), the execution time for default thread is 14.392s which also falls as we increase the number of threads. With six thread, the run time is 3.743s and after that, time changes with increase in thread number gives just slight variation in time.

Conclusion:

By analyzing the graph, we can assume the optimal number of threads is 10 because there are no such drastic changes in the run time even after increasing the number of threads.

There is no extreme difference in the shape of the curve between type A and type B except in some cases like:

1.When we thread with 2, type A has lesser time drop than type B,

2.From thread 5 to 10, we can see more time drop in type A in comparison to type B.

3.Comparitively, on thread 10-50, we can see more time drop in type A than on type B.

4.Both cases are not running on same time with consecutive increase in number of threads.

The reasons behind those changes in shape is due to different in command line argument that we passed. Different band of images may have to wait for different amount of times.